Bakalárska práca

České
vysoké
učení technické
v Praze

F3

Fakulta elektrotechnická
Katedra počítačov

Návrh a tvorba backend strany platformy pre podporu gastronómie

Michal Šalaga

Vedúci: Ing. Pavel Náplava, PhD.
Odbor: Softwarové inžinierstvo a technológie
Máj 2022

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

I. OSOBNÍ A STUDIJNÍ ÚDAJE

Příjmení: **Šalaga** Jméno: **Michal** Osobní číslo: **483730**

Fakulta/ústav: **Fakulta elektrotechnická**

Zadávající katedra/ústav: **Katedra počítačů**

Studijní program: **Softwarové inženýrství a technologie**

II. ÚDAJE K BAKALÁŘSKÉ PRÁCI

Název bakalářské práce:

Návrh a tvorba backend strany platformy pro podporu gastronomie

Název bakalářské práce anglicky:

Design and implementation of a gastronomy platform backend

Pokyny pro vypracování:

Navrhněte a implementujte backend část platformy pro podporu gastronomie, založenou na soutěžení zákazníků. Soutěže a jejich parametry si budou moct definovat registrované podniky. Zákazníci se budou moct do soutěže přihlásit a v době platnosti soutěže sbírat kredity, počítané z načtených účtenek. Účastníci budou soutěžit mezi sebou a po ukončení soutěže se rozdělí výhry.
Postupujte následovně:
1) Proveďte business analýzu platformy, definujte požadavky na funkčnosti backendu.
2) Navrhněte architekturu backendu.
3) Vyberte vhodnou technologii pro realizaci a způsob nasazení.
4) Během všech návrhů reflektujte požadavky frontendu, realizovaného v bakalářské práci Martina Nešněry. Navrhněte a popište vhodný způsob a rozhraní komunikace.
5) Navrženou část realizujte.
6) Navrhněte vhodný způsob ověření fungování vytvořeného backendu. Následně backend ověřte.
7) Ve spolupráci s frontend částí proveďte uživatelské testování celé platformy.
8) Proveďte nákladovou analýzu údržby a provozu backendu.

Seznam doporučené literatury:

1. Dokumentace k frameworku Node.js: https://nodejs.org/en/docs/
2. Dokumentace k MongoDb databázi: https://docs.mongodb.com/manual/
3. Dokumentace a manuál k službě Amazon S3: https://docs.aws.amazon.com/AmazonS3/latest/userguide/Welcome.html
4. Dokumentace k technologií Socket.io: https://socket.io/docs/v4/
5. Dokumentace a manuál k frameworku Expressjs: https://expressjs.com/en/4x/api.html

Jméno a pracoviště vedoucí(ho) bakalářské práce:

Ing. Pavel Náplava, Ph.D. Centrum znalostního managementu FEL

Jméno a pracoviště druhé(ho) vedoucí(ho) nebo konzultanta(ky) bakalářské práce:

Datum zadání bakalářské práce: **02.02.2022** Termín odevzdání bakalářské práce: **20.05.2022**

Platnost zadání bakalářské práce: **30.09.2023**

Ing. Pavel Náplava, Ph.D. podpis vedoucí(ho) práce	podpis vedoucí(ho) ústavu/katedry	prof. Mgr. Petr Páta, Ph.D. podpis děkana(ky)

III. PŘEVZETÍ ZADÁNÍ

Student bere na vědomí, že je povinen vypracovat bakalářskou práci samostatně, bez cizí pomoci, s výjimkou poskytnutých konzultací. Seznam použité literatury, jiných pramenů a jmen konzultantů je třeba uvést v bakalářské práci.

. _____
Datum převzetí zadání

_____
Podpis studenta

Poďakovanie

Predovšetkým by som chcel poďakovať svojmu vedúcemu Ing. Pavlovi Náplavovi, PhD. za cenné rady a podporu pri spoločných konzultáciach. Takisto by som chcel za podporu poďakovať svojej rodine a kamarátom.

Prehlásenie

Prehlasujem, že som predloženú bakalársku prácu vypracoval samostatne, a že som uviedol všetku použitú literatúru.

V Prahe, mája 13, 2022

Abstrakt

Cieľom tejto bakalárskej práce bolo analyzovať a navrhnúť backend časť platformy pre podporu podnikov v oblasti gastronómie. Ďalej bolo nutné zvoliť vhodné technológie pre implementáciu tejto aplikácie. Ďalším bodom bakalárskej práce bola samotná implementácia a následné testovanie funkčnosti užívateľskými a automatizovanými testami. Táto práca je prepojená s bakalárskou prácou zaoberajúcou sa frontendovou časťou platformy vypracovanou kolegom Martinom Nešněrom.

Kľúčové slová: Node.js, AWS, Mongoose, MongoDB, gastronómia, REST, Heroku

Vedúci: Ing. Pavel Náplava, PhD.

Abstract

The aim of this bachelor's thesis was to analyze and design backend part of the platform for support of businesses in gastronomy. Next goal was to choose suitable technologies for implementation of this application. Next goal of this bachelor thesis was implementation followed by testing with user and automated unit tests. This thesis is linked with bachelor thesis dedicated to frontend part of the platform created by my colleague Martin Nešněra.

Keywords: Node.js, AWS, Mongoose, MongoDB, gastronomy, REST, Heroku

Obsah

Obrázky

Tabuľky

Kapitola 1
Úvod

Najvýznamnejším faktorom ovplyvňujúcim životy ľudí v posledných rokoch bola bezpochyby pandémia ochorenia Covid-19. Toto ochorenie ovplyvnilo napríklad vzdelávanie, šport, priemysel alebo gastronómiu. Dá sa povedať, že nezostalo odvetvie, ktoré nebolo týmto ochorením do určitej miery zasiahnuté. Pri prechádzaní mestom si nešlo nevšimnúť jednotlivé podniky, ktoré zanikli, alebo zívali prázdnotou v dôsledku nariadených opatrení. V čase, keď je nutné dodržiavať odstup od osôb minimálne dva metre je prakticky nemožné, aby podniky ako sú napríklad bary a reštaurácie boli v prevádzke v plnom rozsahu. Tieto podmienky spôsobili odliv zákazníkov a tým pádom aj tržieb jednotlivých podnikov. Cieľom tejto práce je snaha pomôcť podnikom z oblasti gastronómie prekonať ťažké obdobie a následky spôsobené ochorením Covid-19 zviditeľnením sa, udržaním momentálnej a získaním novej klientely. Táto snaha by mala prebiehať v období po pandémii, alebo v období medzi jednotlivými vlnami ochorenia keď opatrenia neobmedzujú fungovanie gastronomických podnikov.

Spolu s kolegom Martinom Nešněrom sme začali v rámci predmetu PRO pracovať na projekte „DrinKing“. Jedná sa o súťažnú platformu pomocou ktorej je možné vytvárať jednorázové akcie v jednotlivých spolupracujúcich podnikoch. Cieľom akcie je získať odmenu vo forme kreditu do daného podniku, ktorý bude môcť výherca uplatniť k zakúpeniu sortimentu. Účastníkom akcie sa stane zákazník, ktorý do mobilnej aplikácie nahrá neskôr potvrdenú účtenku v dobe trvania akcie. Jednotlivé pridané účtenky budú uložene do databázy a neskôr ich analyzuje správca aplikácie v oddelenej webovej aplikácií, kde má právo účtenku schváliť alebo zamietnuť. Po schválení, sa následne účastníkovi pripočítajú body slúžiace ako ukazateľ umiestnenia v rebríčku súťažiacich v danej akcii. História pridaných účteniek v danej akcii bude pre účastníka takisto dostupná. Po skončení doby vyhradenej na trvanie akcie a po schválení alebo odmietnutí všetkých účteniek správcami aplikácie bude daná udalosť uvedená do stavu dokončená a následne budú automaticky priradené dopredu dohodnuté odmeny.

Cieľom tejto bakalárskej práce je preto návrh a implementácia backend časti platformy, ktorá pomôže zvýšiť návštevnosť a zviditeľniť podniky v oblasti gastronómie.

Súčasťou projektu je frontend mobilná aplikácia, správcovská webová aplikácia slúžiaca na správu užívateľov a účtov jednotlivých podnikov a nakoniec backend aplikácia zastrešujúca biznis logiku a komunikáciu s mobilnou a webovou aplikáciou projektu. Na prvých dvoch spomenutých aplikáciach pracuje v rámci svojej bakalárskej práce kolega Martin Nešněra [29]. Táto bakalárska práca je venovaná backend aplikácií. Postup práce na bakalárskej práci je rozdelený do nasledujúcich častí:

- Analytická časť venovaná špecifikácií požiadavkou na aplikáciu z technického pohľadu. Takisto bude popísaný napríklad doménový model a stavové diagramy jednotlivých entít.
- Návrh riešenia a porovnanie technológií, ktoré je možné použiť na realizáciu daného riešenia. Prebehne analýza modulov ktoré budeme implementovať pomocou vlastných zdrojov a modulov pre ktoré využijeme dostupné integračné služby. Súčasťou bude nakoniec aj návrh riešenia nasadenia aplikácií vo fáze vývoja a v produkčnej fáze.
- Popis implementácie navrhnutého riešenia. Súčasťou bude aj nákladová analýza behu a údržby aplikácie.
- Spôsob testovania aplikácie. Súčasťou bude popis jednotkových testov, záťažových testov vykonávaných pomocou aplikácie Postman, a takisto aj popis procesných testov na ktorých sa podieľalo niekoľko osôb vykonávajúcich testovacie scenáre.

Nasleduje kapitola, v ktorej sú v krátkosti popísané jednotlivé časti platformy a ich význam. Ďalej sa zameriame na analýzu konkrétnej časti platformy a tou je backend časť.

Kapitola 2
Popis aplikácií v projekte

V nasledujúcej kapitole sa venujem popisu jednotlivých častí systému v projekte DrinKing, a to z dôvodu poskytnutia prehľadu funkcionality a zmyslu týchto častí. V neskorších kapitolách sa budem venovať výhradne backend časti systému.

2.1 Frontend mobilná časť aplikácia

Mobilná aplikácia vyvinutá pomocou technológie React Native mojím kolegom Martinom Nešněrom[29] slúži ako komponenta View v MVC architektúre, ktorá bude popísaná neskôr. Typickým príkladom použitia aplikácie je prihlásenie užívateľa do aplikácie, vyfotografovanie účtenky ku konkrétnej jednorázovej akcii. Ďalším použitím môže byť otvorenie záložky kredity a následná aktivácia odmeny u konkrétneho podniku. Aplikácia takisto disponuje komunikáciou v reálnom čase, implementovanú pomocou knižnice Socket.io.

2.2 Frontend správcovská časť aplikácia

Webová aplikácia slúžiaca predovšetkým pre správu podnikového profilu ale takisto na správu užívateľov správcom aplikácie. Je tu poskytnuté jednoduché grafické rozhranie pre tvorbu jednorázových akcií, úpravu informácií o podniku ale nachádza sa tu možnosť validácie pridaných účteniek. Táto validácia prebieha pomocou komunikácie v reálnom čase. Vývoju aplikácie sa venuje kolega Martin Nešněra[29].

2.3 Backend časť aplikácie

Serverová aplikácia pokrývajúca autentizáciu a biznis logiku pre frontend aplikácie. Zabezpečuje spoľahlivú komunikáciu pomocou vystaveného komunikačného rozhrania.

2.4 Záver kapitoly

Z dôvodu pomerne vysokej komplexnosti systému bolo nutné rozdeliť jeho funkcionalitu do troch častí. Do mobilnej aplikácie pre užívateľov, kde sa môžu zúčastňovať na súťažiach pridávaním účteniek. Do správcovskej aplikácie pre manažérov podniku a pre správcov aplikácie kde môžu napríklad jednotlivé udalosti pridávať. A nakoniec do backend časti aplikácie, ktorá zastrešuje komunikačné rozhranie a biznis logiku pre obidve spomínané časti aplikácie. Ak by boli predmetom bakalárskej práce všetky časti systému, táto bakalárska práca by bola rozsiahla. Práve preto sa táto bakalárska práca zameriava na backend časť systému a druhá bakalárska práca, na ktorej pracuje kolega Martin Nešněra sa zaoberá frontend časťou. V nasledujúcej kapitole sa zameriam na to, čo projekt dokáže potencionálnym spolupracujúcim podnikom a užívateľom priniesť, a zároveň analyzujem existujúce riešenia s podobným prínosom.

Kapitola 3

Analýza existujúcich riešení

nasledujúca kapitola popisuje prínos projektu a následne analyzuje existujúce riešenia s podobnou funkcionalitou a prínosom.

3.1 Očakávaný prínos projektu

Cieľom projektu je zvýšenie návštevnosti zákazníkov a zvýšenie tržieb v spolupracujúcich podnikoch postihnutých následkami pandémie ochorenia Covid-19 formou jednorázových akcií s možnosťou získania kreditu na nákup sortimentu v konkrétnom podniku. Kritickou analytickou činnosťou ktorú sme spoločne s kolegom Martinom Nešněrom vykonávali bolo zisťovanie záujmu podnikov o takýto typ projektu. Tento záujem bol zisťovaný pomocou osobných návštev v jednotlivých podnikoch s dopredu dohodnutým opisom a vysvetlením fungovania projektu. Výsledky týchto návštev sme spoločne s kolegom zaznamenávali do dokumentu na Google Docs. Navštívili sme spolu 14 podnikov a z toho 4 potvrdili, že majú záujem o takýto typ projektu, 3 podniky vyjadrili pozitívne stanovisko s projektom s tým, že keď projekt začne je potrebné aby sme sa im ozvali, v piatich podnikoch sme nezastihli manažéra alebo vlastníka podniku, a nakoniec 2 podniky vyjadrili nezáujem s našim projektom. Z týchto výsledkov bolo usúdené, že projekt má predispozíciu na to aby bol pre podniky prínosný.

Takisto je tento projekt prínosný pre užívateľov, ktorí si môžu pomocou aplikácie vyhľadať udalosti v jednotlivých podnikov a vychutnať si v nich svoje obľúbené nápoje alebo jedlo s možnosťou získania odmeny vo forme kreditu do tohoto podniku na ďalší nákup.

Výhodou projektu DrinKing je nulová vstupná investícia zo strany spolupracujúceho podniku a na druhej strane potenciálne navýšenie zisku pre daný podnik. Ďalšou výhodou pre spolupracujúce podniky je, že sa dokážu vďaka našej platforme zviditeľniť. Neúspech projektu neznamená pre podnik riziko, pretože podnik platí províziu vo výške desať percent jedine za reálne preukázaný nákup sortimentu zákazníkom v priebehu akcie. Taktiež sa nevyžaduje zložitá integrácia s ostatnými systémami.

3.2 Existujúce riešenia

Nasledujúce projekty sa svojou funkcionalitou a prínosom podobajú na náš projekt. Porovnania výhod a nevýhod s naším projektom vychádzajú z biznis modelu popísaného v bakalárskej práci kolegu Martina Nešněru[29]. V prípade BeerSportu je to hlavne podoba v odmeňovaní užívateľov vo forme cashbacku a vernostným programom. Pri Wolte ide o možnosť podniku zviditeľniť sa na trhu a poskytnúť zákazníkovi rôzne akcie. Trifft sa snaží o prilákanie nových a udržanie stálych zákazníkov vo forme vernostného programu. Všetky spomínané projekty boli vyhľadávané pomocou služieb Google Play, App Store a podľa referencií od ostatných osôb.

3.2.1 BeerSport

Projekt s názvom BeerSport poskytuje možnosť rozpoznávania načapovaných pív pomocou aplikácie využívajúcej umelú inteligenciu. Po následnej konzumácií, aplikácia odmení užívateľa vo forme cashbacku. BeerSport sa zameriava výhradne na pivo a jedná sa o úspešný projekt s priemerným počtom mesačných užívateľov až štyristotisíc. Zdrojom príjmov projektu sú dva poplatky, ktoré musí výrobca piva zaplatiť. Jedná sa o vstupný poplatok v radoch stoviek tisíc korún a následne poplatok, ktorý reprezentuje provízie za reálne a prekázané aktivácie užívateľov[1].

Na obrázku 3.1, môžeme vidieť screenshot z hlavnej stránky aplikácie BeerSport, umožňujúca zvoliť si typ piva, ktorý chceme kúpiť za kredit. V dolnej časti sa nachádza menu umožňujúce zobrazenie priateľov a momentálnej výšky kreditu.

Výhody :

- Jednoduchosť.
- Zaujímavé technológie.
- Pridaná hodnota pre používateľov.

Nevýhody :

- Vysoká vstupná investícia pre podniky v stovkách tisíc korún.
- Zameranie jedine na segment s výrobcami pív.
- Vysoká cena vývoja aplikácie.

Obrázok 3.1: Screenshot z aplikácie BeerSport

[1]

3.2.2 Wolt

Umožňuje rýchlo a jednoducho objaviť jedlo z reštaurácií s následnou možnosťou vytvorenia objednávky cez túto platformu. Zdroje príjmov Woltu pochádzajú hlavne z provízií, ktoré musí podnik zaplatiť za sprostredkovanie objednávky a doručenia službami Woltu. Aplikácia obsahuje u niektorých podnikových profilov vernostné programy za ktoré je možné napríklad získať zľavy na ďalšiu objednávku. Podobnosť s naším projektom je vidieť v spôsobe získavania príjmov a motivácie zákazníkov používať danú službu spojenú s možnosťou získania odmeny[2].

Na obrázku 3.2, môžeme vidieť screenshot z aplikácie Wolt, ktorá nám umožňuje rozkliknúť si profil podniku s následnou možnosťou objednania jedla alebo pitia. Aplikácia v dolnej časti poskytuje menu s možnosťou vyhľadania podnikov v okolí, alebo napríklad aj podľa typu jedla na ktoré máme momentálne chuť. V záložke profil si môžeme pozrieť momentálne kupóny s dočasnou platnosťou, ktoré sme schopný použiť na objednávku v konkrétnom podniku.

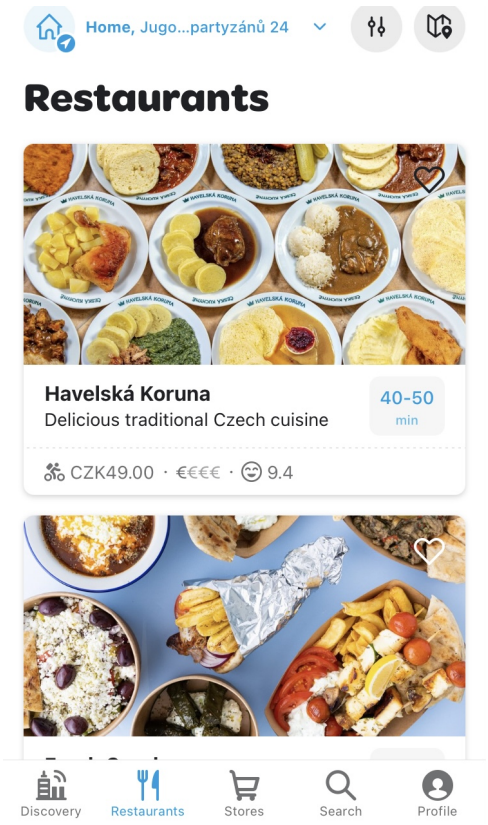
[1] Screenshot 3.1 z aplikácie BeerSport vyhotovený autorom bakalárskej práce.

Výhody :

- Jednoduchý spôsob doručenia jedla.
- Možnosť získať zaujímavé zľavy.

Nevýhody :

- Vysoká provízia platená podnikom až tridsať percent.
- Vysoká konkurencia na trhu vedie k nižším ziskom.



Obrázok 3.2: Screenshot z aplikácie Wolt

[2]

3.2.3 Trifft

Aplikácia Trifft slúži na získavanie nových, a udržanie stálych zákazníkov pomocou vernostných programov. V aplikácií je možné vidieť menu v konkrétnom podniku. Zákazník za nákup získava body do vernostného programu a pri splnení cieľového počtu bodov dostane odmenu. Aplikácia funguje od roku 2019[3].

[2]Screenshot 3.2 z aplikácie Wolt vyhotovený autorom bakalárskej práce.

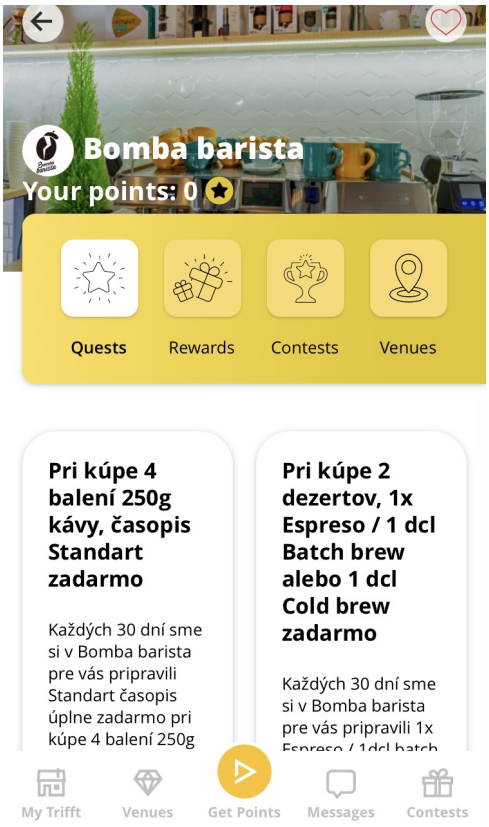
Na obrázku 3.3, môžeme vidieť screenshot z aplikácie Trifft. Na tejto stránke sa nám zobrazili vybrané ponuky v podnikoch. Máme tu znovu možnosť zobrazenia podnikov v okolí.

Výhody :

- Pomocou aplikácie je možné vidieť menu v podniku.
- Možnosť získať zaujímavé zľavy.

Nevýhody :

- Vysoká vstupná investícia podniku až 43000 Kč a potom mesačné poplatky.
- Aplikácia v poslednej dobe stráca na popularite a prichádzajú o zákazníkov.
- Neintuitívne používanie aplikácie.



Obrázok 3.3: Screenshot z aplikácie Trifft

3

[3]Screenshot 3.3 z aplikácie Trifft vyhotovený autorom bakalárskej práce.

3.3 Záver kapitoly

V tejto kapitole sme zistili, aké riešenia podporujúce podniky v oblasti gastronómie s podobnou funkcionalitou momentálne na trhu fungujú. Všetky tieto riešenia požadujú vysokú vstupnú investície pre podniky, ktoré chcú spolupracovať s týmito spoločnosťami. Avšak, nie každý podnik si takúto investíciu môže dovoliť. Tu vidím výhodu u nášho projektu, kde takáto investícia nie je vyžadovaná. Provízie s predaja sú taktiež nižšie ako u konkurenčných riešení. Nevýhodou môže byť, že jednotlivé riešenia majú dostatočne vybudovanú základňu so spolupracujúcimi podnikmi.

Kapitola 4

Analýza požiadaviek na backend časť aplikácie

Zber požiadaviek je kľúčovou analytickou činnosťou pre správne pochopenie očakávaní budúcich užívateľov od aplikácie. Správne definovanie požiadaviek na začiatku projektu môže pomôcť v dodržiavaní stanovených dátumov doručenia softvéru, šetrí zdroje vyhradené na vývoj softvéru a keďže máme k dispozícií jasne definovanú funkcionalitu , existuje menšia pravdepodobnosť, že do systému zanesieme kód náchylný na chyby alebo kód s nejasnou funkcionalitou. Preto sa v tejto kapitole venujem definícií požiadaviek na backend časť aplikácie. Budú rozdelené na funkčné a nefunkčné požiadavky.

4.1 Funkčné požiadavky

Dávajú prehľad o softvérovej funkcionalite, ktorá musí byť implementovaná. Funkčné požiadavky sa zaoberajú napríklad funkcionalitou modulov autorizácie, autentizácie, externého komunikačného rozhrania, alebo aj notifikačných služieb[4]. V nasledujúcom zozname sú uvedené funkčné požiadavky na backend časť aplikácie.

- **FR1 Registrácia užívateľa:** Systém poskytne rozhranie pre registráciu užívateľa.
- **FR2 Prihlásenie užívateľa:** Systém poskytne rozhranie pre prihlásenie užívateľa.
- **FR3 Zablokovanie užívateľa:** Systém poskytne rozhranie pre zablokovanie prístupu užívateľa.
- **FR4 Odmeny užívateľa:** Systém poskytne rozhranie pre získanie prehľadu odmien užívateľa.
- **FR5 Dáta o užívateľoch:** Systém poskytne rozhranie pre získanie dát o užívateľoch.
- **FR6 Zmena profilovej fotky:** Systém poskytne rozhranie pre zmenu profilovej fotky užívateľa.

- **FR7 Získanie profilovej fotky:** Systém poskytne rozhranie pre získanie profilovej fotky užívateľa.
- **FR8 Vymazanie profilovej fotky:** Systém poskytne rozhranie pre vymazanie profilovej fotky užívateľa.
- **FR9 Zmena hesla:** Systém poskytne rozhranie pre zmenu hesla užívateľa.
- **FR10 Vytvorenie podnikového profilu:** Systém poskytne rozhranie pre vytvorenie podnikového profilu.
- **FR11 Získanie podnikového profilu:** Systém poskytne rozhranie pre získanie dát o podnikovom profile.
- **FR12 Pridanie fotky podniku:** Systém poskytne rozhranie pre pridanie podnikovej profilovej fotky.
- **FR13 Zmazanie fotky podniku:** Systém poskytne rozhranie pre zmazanie podnikovej profilovej fotky.
- **FR14 Vytvorenie udalosti:** Systém poskytne rozhranie pre vytvorenie novej udalosti.
- **FR15 Získanie aktívnych udalostí:** Systém poskytne rozhranie pre získanie prehľadu aktívnych udalostí.
- **FR16 Získanie udalosti:** Systém poskytne rozhranie pre získanie konkrétnej udalosti.
- **FR17 Dokončenie udalosti:** Systém poskytne rozhranie pre dokončenie udalosti.
- **FR18 Vymazanie udalosti:** Systém poskytne rozhranie pre vymazanie udalosti.
- **FR19 Pridanie účtenky:** Systém poskytne rozhranie pre pridanie účtenky.
- **FR20 Schválenie účtenky:** Systém poskytne rozhranie pre schválenie účtenky.
- **FR21 Zamietnutie účtenky:** Systém poskytne rozhranie pre zamietnutie účtenky.
- **FR22 Získanie účteniek v udalosti:** Systém poskytne rozhranie pre získanie prehľadu účteniek v rámci udalosti.
- **FR23 Získanie účteniek od užívateľa:** Systém poskytne rozhranie pre získanie prehľadu účteniek pridaných konkrétnym užívateľom.
- **FR24 Vytvorenie transakcie:** Systém poskytne rozhranie vytvorenie transakcie užívateľa.

- **FR25 Schválenie transakcie:** Systém poskytne rozhranie schválenie transakcie užívateľa.
- **FR26 Zamietnutie transakcie:** Systém poskytne rozhranie zamietnutie transakcie užívateľa.
- **FR27 Vymazanie transakcie:** Systém poskytne rozhranie vymazanie transakcie užívateľa.
- **FR28 Získanie transakcie:** Systém poskytne rozhranie získanie transakcie užívateľa.
- **FR29 Získanie prehľadu transakcií:** Systém poskytne rozhranie pre získanie prehľadu všetkých transakcií.
- **FR30 Odosielanie informačných emailov:** Systém automaticky odošle emailové notifikácie v rôznych prípadoch.
- **FR31 Zmena stavu udalosti:** Systém automaticky aktualizuje stav udalosti.

4.2 Nefunkčné požiadavky

Zaoberajú sa kvalitatívnymi vlastnosťami softvéru ako napríklad spoľahlivosť systému, ktorá udáva pravdepodobnosť zlyhania systému alebo výkon, ktorým opisujeme správanie systému v rôznych situáciach ako je napríklad vysoká záťaž systému[4]. V nasledujúcom zozname sú uvedené nefunkčné požiadavky na backend aplikáciu.

- **NFR1 Bezpečnosť:** Systém bude ukladať heslo v bezpečnej zašifrovanej forme a pri každom dotaze prebehne validácia vstupov.
- **NFR2 Autorizácia na základe rolí:** Systém bude rozpoznávať role užívateľov a tým sa zabezpečí obmedzenie prístupových práv k používaniu rozhrania.
- **NFR3 Komunikácia v reálnom čase:** Systém zabezpečí komunikáciu v reálnom času pre zobrazenie transakcií.
- **NFR4 Spoľahlivosť:** Systém zabezpečí spoľahlivú odpoveď a návratovú hodnotu klientovi.

4.3 Záver kapitoly

V tejto kapitole som najprv opísal načo nám funkčné a nefunkčné požiadavky slúžia, a ďalej som jednotlivé typy požiadaviek na aplikáciu definoval. V nasledujúcej kapitole tieto požiadavky využijeme na definovanie entít potrebných v aplikácií.

Kapitola 5

Analýza doménového modelu a entít

V nasledujúcej kapitole sa venujem analýze a modelovaniu jednotlivých doménových entít a ich stavov. Poznať všetky vlastnosti entít a ich vzťahov s inými entitami nám neskôr pomôže napríklad pri modelovaní konkrétnej databázovej schémy.

5.1 Doménový model

Doménový model je reprezentáciou objektov z reálneho sveta. Predstavuje určitú mieru abstrakcie, kde nezávisí na konkrétnej implementácií a teda nejedná sa o softvérové objekty. Tento model vzniká premenou požiadaviek na systém. Obsahuje atribúty, konceptuálne triedy, vzťahy medzi nimi a ich kardinality[5]. Postup pri tvorbe doménového modelu bol nasledovný:

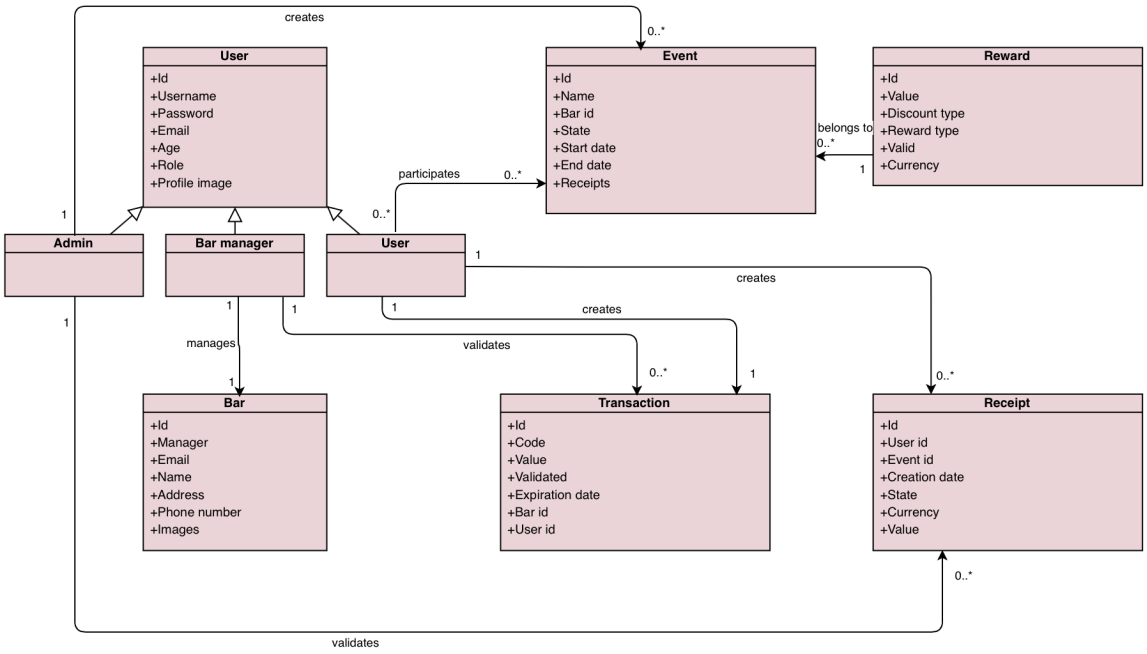
1. **Identifikácia konceptuálnych tried :** K identifikácií všetkých tried došlo na základe funkčných požiadaviek definovaných v kapitole 3.3.

2. **Nakreslenie týchto tried :** Na nakreslenie diagramu som použil online nástroj Visual Paradigm.

3. **Pridanie vzťahov a ich kardinalít k triedam :** Pridanie kardinalít a vzťahov medzi entitatmi znovu vyplynulo z definovaných požiadaviek a z predchádzajúcej analýzy.

4. **Pridanie atribútov k triedam :** Nakoniec došlo k definovaniu jednotlivých atribútov k triedam.

Na nasledujúcom obrázku 5.1 môžeme vidieť doménový model nášho systému. Zobrazuje všetky domény v systéme, pričom môžeme vidieť , že existujú tri typy užívateľa, ktorý každý z nich môže vykonávať iné funkcie v systéme. Ďalšou dôležitou skutočnosťou je, že entita Odmena nemôže existovať bez pridelenej udalosti.



Obrázok 5.1: Doménový diagram

5.2 Popis jednotlivých entít

V tejto sekcii popíšem jednotlivé entity a význam ich atribútov, ktoré boli definované v predchádzajúcej sekcii.

5.2.1 Užívateľ

Užívateľ v projekte môže zastávať tri typy rolí, a to užívateľ, správca alebo podnik. V aplikácií sa bude jednať o rovnaký typ triedy rozlíšený vlastnosťou rola. Záznamy o užívateľoch obsahujú nasledujúce dáta.

- id - automaticky generovaný identifikátor objektu.
- username - jedinečná prezývka užívateľa v aplikácií.
- password - zašifrované a osolené heslo zvolené. užívateľom.
- email - emailová adresa užívateľa.
- role - rola užívateľa od ktorej sa odvíjajú sa od ňej prístupové práva.
- profile_image - po príhlasení si môže užívateľ zmeniť svoju profilovú fotku.
- banned - identifikátor odobratia prístupu do aplikácie správcom

- prizes - objekt obsahujúci informácie o počte výhier a ich type.
  - gold
  - silver
  - bronze
- events - pole obsahujúce identifikátory akcií na ktorých sa užívateľ zúčastnil.
- total_events _no - celkový počet akcií na ktorých sa užívateľ zúčastnil.
- total_credits _gained - celkový počet bodov, ktoré užívateľ doposiaľ získal
- best_score - najlepšie skóre, ktoré užívateľ na akcii dosiahol.

5.2.2 Podnik

Entita obsahujúca všeobecné informácie o podniku a kto je jeho správcom. Nejedná sa o samotného užívateľa typu podnik.

- id - automaticky generovaný identifikátor objektu.
- manager - identifikátor užívateľa s rolou podnik, ktorý je správcom podniku.
- address - objekt obsahujúci informácie o adrese podniku
  - street
  - building_number
  - city
  - postal_code
- images - pole identifikátorov fotografií podniku.
- phone_number - kontaktné telefónne číslo podniku.

5.2.3 Udalosť

Udalosť predstavuje jednorázovú akciu v podniku s vymedzeným časovým oknom kedy je možné k tejto udalosti pridávať účtenky. Obsahuje nasledovné dáta.

- id - automaticky generovaný identifikátor objektu.
- bar_id - identifikátor podniku v ktorom sa udalosť koná.
- state - stav v ktorom sa udalosť nachádza.
- start_date - dátum začiatku udalosti.

- end_date - dátum ukončenia udalosti.
- participants - pole objektov nesúcich informáciu o súťažiacich užívateľoch.
  - user_id - identifikátor užívateľa.
  - score - skóre, ktoré užívateľ doposiaľ dosiahol.
- receipts - pole identifikátorov účteniek priradených k udalosti.

5.2.4 Odmena

Táto entita predstavuje odmenu, ktorú je možné získať za popredné umiestnenie v akcii. Počet odmien, ich typ a hodnotu určuje správca podniku. Obsahuje nasledovné dáta.

- id - automaticky generovaný identifikátor objektu.
- value - hodnota odmeny vyjadrená číslom.
- discount_type - typ odmeny, môže mať hodnotu percentá alebo hotovosť.
- reward_index - poradie odmeny v rámci udalosti.
- valid - hodnota vyjadrujúca platnosť odmeny. Ak je odmena neplatná, znamená to, že už bola priradená užívateľovi.
- currency - typ meny v ktorej bude hodnota priradená užívateľovi.
- event_id - identifikátor udalosti ku ktorému je odmena priradená.
- bar_id - identifikátor podniku ku ktorému je odmena priradená.

5.2.5 Transakcia

Entita vyjadrujúca transakciu vytvorenú užívateľom pri využití odmien v konkrétnom podniku. Platnosť transakcie je obmedzená na desať minút, počas ktorých môže správca podniku transakciu potvrdiť alebo zrušiť. Obsahuje nasledujúce dáta.

- id - automaticky generovaný identifikátor objektu.
- code - automaticky generovaný kód transakcie.
- value - hodnota kreditu, ktoré chce užívateľ v danom podniku využiť.
- validated - hodnota vyjadrujúca či bola transakcia potvrdená.
- expiration_date - dátum a čas expirácie transakcie.
- user_id - identifikátor užívateľa, ktorý danú transakciu vytvoril.
- bar_id - identifikátor podniku ku ktorému je transakcia priradená.

5.2.6 Účtenka

Predstavuje účtenku vyfotografovanú a vytvorenú užívateľom k danej udalosti. Účtenka sa následne musí manuálne vyhodnotiť a ak je účtenka reálna, bude užívateľovi pripočítané skóre v rámci udalosti. Obsahuje nasledujúce dáta.

- id - automaticky generovaný identifikátor objektu.
- user_id - identifikátor užívateľa, ktorý danú účtenku vytvoril.
- receipt_id - kód, nachádzajúci sa na reálnej papierovej účtenke.
- event_id - identifikátor udalosti ku ktorému je účtenka priradená.
- creation_date - dátum a čas vytvorenia účtenky.
- state - stav účtenky.
- value - hodnota účtenky, ktorá je uvedená na reálnej papierovej účtenke.
- currency - mena v ktorej je reálna účtenka uvedená

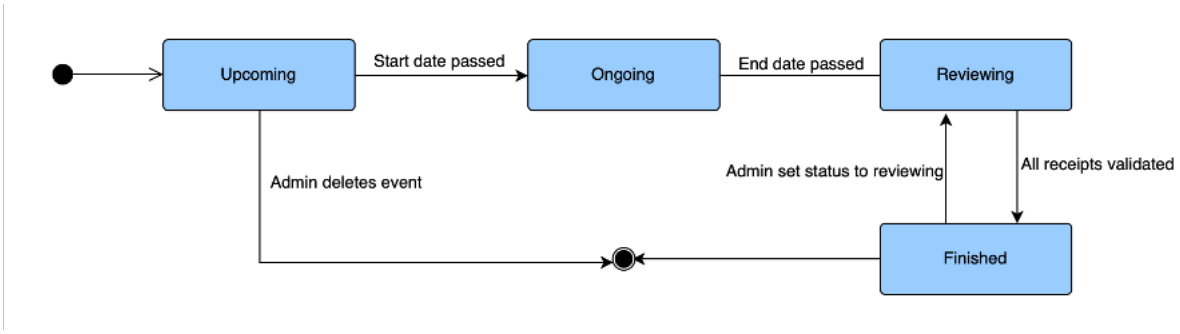
5.3 Stavové diagramy entít

Stavový diagram používame na zobrazenie stavov, v ktorých sa entita môže počas svojho života nachádzať, a ďalej zobrazuje dynamiku prechodu medzi týmito stavmi. Nasledovné entity považujem za najdôležitejšie a určite je vhodné zaznamenať v akom stave sa môžu nachádzať[6].

5.3.1 Udalosť

Entita Udalosť sa môže nachádzať v nasledujúcich štyroch stavoch popísaných aj na obrázku 5.2:

- **Nadchádzajúca** (Upcoming) : V tomto stave sa udalosť nachádza okamžite po vytvorení správcom aplikácie a platí, že dátum začiatku akcie ešte nenastal.
- **Prebiehajúca** (Ongoing) : Do tohto stavu sa entita udalosť dostane vtedy, keď nastane dátum začiatku akcie. Tento prechod zabezpečuje systém automaticky.
- **Vyhodnocovanie** (Reviewing) : Do tohto stavu sa entita udalosť dostane vtedy, keď nastane dátum ukončenia akcie. Tento prechod zabezpečuje systém automaticky.
- **Ukončená** (Finished) : V tomto stave sa udalosť nachádza v tom prípade keď správca aplikácie schválil alebo zamietol všetky pridané účtenky vzťahujúce sa k danej udalosti. Tento prechod neprebieha automaticky ale je nutné daný stav odkliknúť v správcovskej webovej aplikácií.

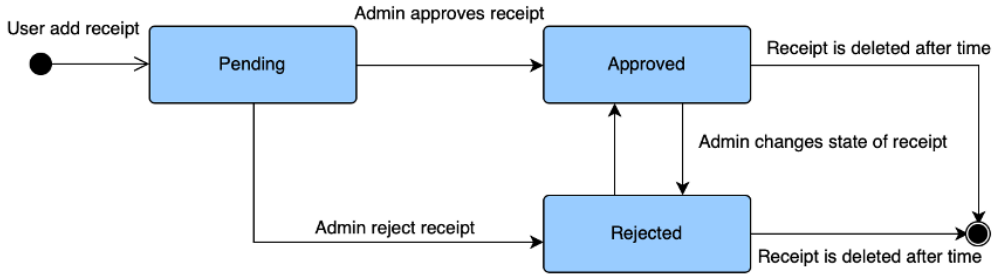


Obrázok 5.2: Stavový diagram udalosti

5.3.2 Účtenka

Entita účtenka sa môže nachádzať v nasledujúcich troch stavoch popísaných aj na obrázku 5.3 :

- **Nerozhodnutá** (Pending) : Počiatočný stav účtenky hneď po pridaní užívateľom ku konkrétnej akcii. Skóre užívateľa v akcii zatiaľ nie je navýšené.
- **Schválená** (Approved) : Do tohto stavu sa účtenka dostane schválením v správcovskej webovej aplikácií správcom. V tomto okamihu je užívateľovi pripočítané skóre v danej akcii. Užívateľ bude o tejto skutočnosti informovaný e-mailom.
- **Zamietnutá** (Rejected) : Do tohto stavu sa účtenka dostane zamietnutím v správcovskej webovej aplikácií správcom. Aj o tejto skutočnosti je užívateľ informovaný e-mailom.



Obrázok 5.3: Stavový diagram účtenky

5.4 Záver kapitoly

V tejto kapitole došlo k definovaniu doménového modelu, jednotlivých atribútov entít, a k vzťahom medzi jednotlivými entitami. Tieto informácie

budú potrebné pre určenie typu dát, ktoré bude musieť backend časť aplikácie spracovávať, a zároveň boli upresnené operácie, ktoré bude medzi jednotlivými entitami možné vykonávať.

Kapitola 6
Návrh riešenia

V predchádzajúcich kapitolách boli vydefinované požiadavky na aplikáciu, doménový model a aj samotné popisy entít a ich atribútov. Tieto informácie nám v nasledujúcej kapitole pomôžu v návrhu architektúry aplikácie a samotnému výberu technológií, ktoré budú využité na implementáciu. Dôjde takisto k výberu modulov, ktoré budú implementované pomocou vlastných zdrojov a k výberu modulov, na ktoré použijeme existujúce spoľahlivé služby tretích strán.

6.1 Architektúra backend časti aplikácie

nasledujúca sekcia sa venuje výberu architektúry vývoja aplikácie. Medzi parametre ovplyvňujúce výber patrí čas nutný na vývoj, aktuálne skúsenosti s vývojom danej architektúry aplikácie ale aj to, či daná aplikácia bude niekedy integrovaná v rámci iných systémov. Do úvahy spadajú dve najznámejšie typy architektúry aplikácie a to monolit a mikroslužby.

6.1.1 Monolit vs. mikroslužby

Momentálne najznámejšie a najpoužívanejšie architektúry aplikácie sú monolitická architektúra a mikroslužby. V krátkosti si popíšeme každú z nich. Následne porovnáme ich výhody a nevýhody.

- **Monolitická architektúra** : Aplikácia je koncipovaná ako celok, to znamená, že všetká funkcionalita a moduly sa nachádzajú v jednom jar, war alebo zip súbore, ktorý je následne nasadený na lokálny alebo vzdialený server. Jedná sa o moduly a komponenty zabezpečujúce autorizáciu, validáciu, biznis logiku ale aj perzistentnú vrstvu zabezpečujúcu prístup do databázy[7].

  Výhody:

  - Jednoduché nasadenie aplikácie ako celku pri vývoji aplikácie a v skorých fázach produkčnej fáze.
  - Začiatok vývoja rýchlejšie ako u mikroslužieb kde je nutné dôkladnejšie plánovanie.

- Jednoduché testovanie aplikácie.
- Jednoduchšie hľadanie v aplikácií kde nie je nutné agregovať výpisy z viacerých nezávislých mikroslužieb.
- Jednoduché škálovanie spustením viacerých kópií s vyvažovačom zaťaženia.

Nevýhody:

- Ťažká orientácia v kóde pri narastajúcej zložitosti aplikácie.
- Mála zmena implementácie znamená, že sa celá aplikácia musí znovu nasadiť.
- Používa sa jeden programovací jazyk a nie je možné túto skutočnosť nijak zmeniť a prejsť na iný jazyk.

- **Mikroslužby** : Jedná sa o aplikáciu, kde sú jednotlivé moduly a komponenty rozdelené do viacerých služieb nasadzovaných samostatne a nezávislo do ostatných modulov. Každý z modulov poskytuje precízne vydefinované rozhranie na komunikáciu s ostatnými mikroslužbami. Na rozdiel od monolitickej aplikácie, každá z mikroslužieb komunikuje s rozdielnou databázou pre dosiahnutie čo najvačšej nezávislosti od ostatných modulov[7].

  Výhody:

  - Prehľadnosť kódu.
  - Jednoduchá zmena implementácie bez nutnosti znovunasadenia celej aplikácie. Stačí nasadiť konkrétnu mikroslužbu.
  - Každá mikroslužba môže byť napísaná v inom jazyku.
  - Lepšia flexibilita pri škálovaní aplikácie. Možnosť duplikácie konkrétnych mikroslužieb.

  Nevýhody:

  - Zložitejšie testovanie aplikácie z dôvodu nutnosti volania ďalších mikroslužieb.
  - Zložité hľadanie chýb v aplikácií.
  - Zdĺhavé a zložité nasadzovanie aplikácie napríklad pri novej verzii aplikácie.
  - Vysoká miera potrebných skúsenosti s vývojom mikroslužieb.

Pri výbere architektúry aplikácie bolo brané do úvahy, že s vývojom mikroslužieb nemám dostatočné skúsenosti a takisto čas na tvorbu aplikácie nie je možné predĺžiť ak by došlo k neočakávaným problémom. Preto je v tomto ohľade logickejšou voľbou monolitická architektúra. Ako ďalší dôvod pre výber tejto architektúry je obmedzené množstvo zdrojov, ktoré je možné vynaložiť na vývoj aplikácie. Mikroslužby z pravidla vyžadujú viacerých vývojárov.

6.1.2 Viacvrstvová architektúra

Dizajn aplikácie bude koncipovaný použitím viacvrstvovej architektúry, ktorá je najpoužívanejšia pri vývoji systému obsahujúceho prezentačnú vrstvu vo forme užívateľskej mobilnej aplikácie[8].

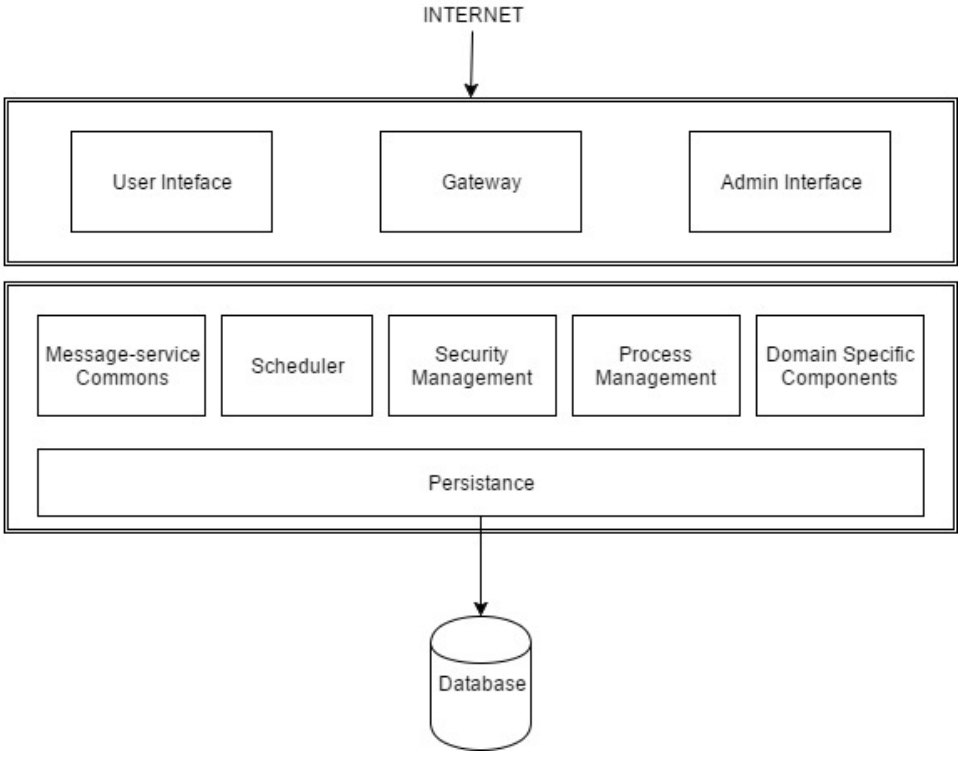
Viacvrstvová architektúra najčastejšie pozostáva z troch vrstiev.

- **Prezentačná vrstva :** V našom projekte predstavuje túto vrstvu užívateľská mobilná aplikácia ale aj správcovská webová aplikácia. Úlohou tejto vrstvy je umožniť užívateľovi interagovať so systémom ale hlavne zobrazovať obsah a dáta vďaka grafickému rozhraniu.

- **Aplikačná vrstva :** Predstavuje strednú vrstvu systému. Obsahuje biznis logiku a kontrolu prístup k perzistentným dátam.

- **Perzistentná vrstva :** Jedná sa o najnižšie položenú vrstvu systému, ktorá sa zaoberá ukladaním a načítaním aplikačných dát. Tieto dáta sú najčastejšie uložené v databázy.

Nasledujúci obrázok 6.1 zobrazuje jednotlivé vrstvy začínajúce zhora prezentačnou vrstvou, aplikačnou vrstvou obsahujúcou rôzne typy služieb, a nakoniec perzistentnou vrstvou komunikujúcou s databázou.



Obrázok 6.1: Schéma viacvrstvovej architektúry aplikácie[30]

6.2 Výber technológií

Ďalším krokom pred samotným vývojom aplikácie je výber technológií pre vývoj, ukladanie dát, dokumentáciu rozhrania ale aj hľadanie dostupných a spoľahlivých integračných služieb spoločností tretích strán, ktoré nám urýchlia vývoj a zamedzia zaneseniu chýb v dôležitých moduloch aplikácie.

6.2.1 Vývoj

Pre implementáciu serverových aplikácií existuje veľké množstvo dostupných technológií. Do úvahy pre mňa pripadá Java a JavaScript, a to z dôvodu skúsenosti s vývojom v týchto programovacích jazykoch. Z dôvodu, že použitie čistého JavaScriptu nie je vhodným adeptom na vývoj klient-server aplikácie a to z dôvodu, že nám neumožňuje využívať knižnice potrebné na implementáciu určitých modulov, a zároveň čistý Javascript nedokáže fungovať bez prehliadača. Preto do úvahy berieme použitie jedného z najznámejších JavaScriptových frameworkov Node.js. Nasleduje porovnanie týchto technológií a následné zdôvodnenie výberu.

- **Java :** Je vysoko úrovňový objektovo orientovaný programovací jazyk. Podporuje viac vláknové vykonávanie operácií. V dnešnej dobe sa najčastejšie používa na budovanie komplexných podnikových systémov. Nastavenie prostredia a samotná implementácia môže byť zdĺhavejšia[9].
- **Node.js :** Je JavaScriptový engine prekladajúci JavaScript do strojového kódu. Primárnym dôvodom k tvorbu tohoto frameworku bola tvorba škálovateľných serverových aplikácií. Jeho výhodou je rýchlosť implementácie, kde nie je nutné zdĺhavé nastavovanie prostredia. Ďalšou výhodou, ktorá nám Node.js poskytuje je tvorba svižných a nenáročných systémov a v neposlednom rade veľká komunita vývojárov podporujúcich a produkujúcich knižnice, ktoré nám uľahčujú implementáciu[9].

Vzhľadom na to, že cieľom projektu je implementácia svižnej serverovej aplikácie zastrešujúcej biznis logiku mobilnej a webovej aplikácie s obmedzenými časovými zdrojmi sa javí použitie Node.js ako vhodnej technológie na vývoj.

6.2.2 Databáza

Pri návrhu perzistentnej vrstvy a výbere databázy máme na výber široké spektrum typov databáz medzi ktorými sa musíme rozhodnúť pre tú, ktorá najlepšie spĺňa požiadavky na moju aplikáciu. Medzi dve základné skupiny databáz, ktoré budeme brať pri výbere do úvahy sú relačné SQL databázy a NoSQL databázy.

Relačné databázy:

- **PostgreSQL :** Je open source relačná databáza používajúca SQL jazyk. Poskytuje mnoho rozšírení napríklad vo forme indexov. Ukázala sa byť

vysoko škálovateľná v objeme dát, ktoré dokáže spracovať a takisto v počte užívateľov, ktorých dokáže zároveň obslúžiť. Dáta sa ukladajú vo forme tabuliek so striktne danými riadkami a stĺpcami. Nevýhodou je, že každá zmena modelu vedie k potrebe znovu nasadenia tohto modelu[10].

- **MySQL :** Jedná sa taktiež o relačnú databázu v ktorej sa ukladajú štrukturované kolekcie dát. Využíva sa väčšinou pre robustnejšie systémy a podniky ju využívajú pre kritické podnikové dátové úložiská[11].

NoSQL databázy:

- **MongoDB :** Dokumentový typ NoSQL databázy ukladá záznamy vo forme flexibilných dokumentov pripomínajúcim JSON formát. MongoDB je jednoduchá na použitie pre vývojárov, ale zároveň poskytuje možnosť škálovateľnosti a dostupnosti. Pri zmene modelu databázy nie je nutné znovu tento model nasadiť, dokumenty môžu mať v tej istej kolekcii rôznu štruktúru, čo poskytuje flexibilitu pre moju aplikáciu[12].

- **Cassandra :** Je takisto NoSQL distribuovaná databáza. Jej silnou stránkou je schopnosť analyzovať obrovské množstvo dát, čo sa v posledných rokoch ukázalo ako kľúčové pre Big Data[13].

Každý typ databázy má svoje výhody ale aj nevýhody. V našom prípade potrebujeme rýchlu databázu s dobrou podporou pre Node.js. Tieto parametre najviac spĺňa NoSQL databáza MongoDB[12].

6.2.3 Integrácie služieb

Cieľom tejto podkapitoly je identifikovať moduly, ktoré nebudem implementovať vlastnými zdrojmi ale použijem integrácie spoľahlivých cloudových služieb. Do úvahy beriem modul s autentifikáciou, ukladaním obrázkov, posielaním emailových notifikácií a logovaním.

Autentifikácia

Autentifikačné a autorizačné služby pre aplikácie poskytuje mnoho produktov na trhu. Jedným z najznámejších je služba Firebase Authentication, ktorá poskytuje jednoducho použiteľnú knižnicu pre Node.js. Podporuje autentifikáciu pomocou hesiel, telefónnych čísel, ale aj prihlásenie pomocou Google, Facebook a ďalších[14].

Jedná sa o skvelú službu, avšak mojím cieľom na tejto bakalárskej práci je získať nové, a overiť predošlé skúsenosti s implementáciou a preto modul s autentifikáciou a autorizáciou budem implementovať vlastnými zdrojmi. Požiadavkami na tento modul budú určite spoľahlivosť, ukladanie zašifrovaných a osolených hesiel, prístup k zdrojom aplikácie na základe role.

Ukladanie obrázkov a súborov

V aplikácií vznikne potreba ukladania veľkého množstva obrázkov s rôznym významom. Jedná sa konkrétne o profilové fotografie podnikov, užívateľov a fotografie účteniek. Tieto súbory môžu v budúcnosti predstavovať nezanedbateľný objem dát, ktoré je potrebné spravovať. Ukladanie na lokálnych úložiskách v dnešnej dobe cloudových úložísk nedáva veľký zmysel. Preto je výhodnejšie využiť služby spoločností, ktoré tieto riešenia poskytujú. Medzi najvýznamnejších poskytovateľov cloudových služieb patrí AWS od Amazonu a Azure od Microsoftu. Obe služby poskytujú podobnú funkcionalitu, výpočtovú kapacitu, úložisko a dokonca aj ceny za služby sú podobné[15]. Keď porovnávame konkrétne úložiskové služby S3 od AWS a Storage Server od Azure, zdá sa, že podpora ale aj dokumentácia pre Node.js je kvalitnejšia od AWS. Preto došlo k výberu služby S3 od AWS pre účely ukladania obrázkov z aplikácie.

Emailové notifikácie

Pre účely informovania užívateľov o nových udalostiach, výhrach ale aj o zvalidovaní transakcií je potrebné integrovať notifikačnú službu pre odosielanie emailov. Vzhľadom na predchádzajúci výber služby S3 od AWS, bolo najrozumnejšie ostať pri AWS službách a nájsť vhodnú službu na tento účel. Tou službou je SES. Existuje knižnica pre Node.js s dobrou podporou a dokumentáciou, zároveň bude jednoduchšie používať túto knižnicu vzhľadom na predchádzajúce skúsenosti so službou S3. Služba poskytuje možnosť posielať emaily konkrétnym užívateľom ale aj hromadné správy. Je možné vytvárať vzory emailov, a následne tieto vzory používať doplnením personalizovaných informácií priamo v kóde[16].

6.2.4 Dokumentácia rozhrania

Dôležitou súčasťou aplikácie je kvalitná dokumentácia. Keďže budeme potrebovať komunikačné rozhranie medzi klientom a serverom, je potrebné toto rozhranie zdokumentovať. Vzhľadom na predošlé skúsenosti s technológiou SwaggerUI, nevidím dôvod prechádzať na inú technológiu. Stačí vytvoriť jeden Swagger súbor s popisom rozhrania aplikácie, technológia následne automaticky vygeneruje prehľadné a zároveň prívetivé grafické rozhranie dokumentácie mojej aplikácie.

6.3 Návrh komunikačného rozhrania

Základnou vlastnosťou serverových aplikácií je vystavenie komunikačného rozhrania hlavne pre klientske aplikácie. Existuje mnoho komunikačných štandardov, preto je potrebné analyzovať a zvážiť, ktorý štandard vyhovuje mojej aplikácií. nasledujúca kapitola popisuje dostupné architektonické štýly tvorby komunikačného rozhrania aplikácie.

6.3.1 REST

Jedná sa o architektonický štýl komunikačného rozhrania, ktorý definuje komunikačné štandardy medzi aplikáciou a webom. Hlavnými znakmi tejto komunikácie je bezstavovosť a oddelenie záujmov klienta a serveru, to znamená, že implementácie týchto aplikácií môžu prebiehať nezávisle na sebe. Komunikácia prebieha posielaním požiadaviek z klientskej aplikácie na modifikáciu alebo získanie dát zo serverovej aplikácie. Tieto požiadavky sú postavené na HTTP protokole. V tomto požiadavku je potrebné definovať operáciu, ktorá serverovej aplikácií naznačí, o aký typ požiadavku sa jedná. Ide o štyri základné operácie a to: GET, POST, PUT alebo DELETE. Ďalej je potrebné definovať cestu k zdroju aplikácie vo forme url[17].

6.3.2 GraphQL

Rozdiel medzi REST-om a GraphQL je v tom, že kým REST predstavuje architektonický koncept definujúci pravidlá pre komunikačné rozhranie, GraphQL je konkrétny dotazovací jazyk. Pomocou GraphQL je možné opísať typ a tvar dát potrebných zo serverovej aplikácie. Cieľom je vytvoriť rýchle a flexibilné rozhranie[18]. Výhodou je, že typ dát v odpovediach serveru sú striktne dané. Ako nevýhody môžeme považovať vysokú náročnosť tvorby takéhoto rozhrania pre vývojárov bez predošlých skúseností.

6.3.3 SOAP

Je protokol založený na posielaní správ umožňujúcich komunikáciu medzi distribuovanými modulmi aplikácie. Môže byť postavený na rôznych nízko úrovňových protokoloch napríklad HTTP. Štruktúra dát SOAP-u je definovaná pomocou XML. Veľkú nevýhodu predstavuje použitie protokolu hlavne pre robustné systémy. Používanie tohto protokolu je na ústupe[19].

6.3.4 WebSocket

Sokety umožňujú komunikáciu medzi dvoma rozdielnymi procesmi na rovnakom alebo vzdialenom počítači. WebSocket protokol zabezpečuje obojsmernú komunikáciu, a v súčastnosti sa používa v aplikáciach kde je potrebné zabezpečiť komunikáciu v reálnom čase bez nutnosti znovunačítania stránky. Akokoľvek užitočná táto technológia je, netreba zabúdať, že pri stratení pripojenia neexistujú mechanizmy znovupripojenia[20].

6.3.5 Výber a návrh rozhrania

Pre hlavnú časť rozhrania som sa rozhodol použiť REST. Je to z dôvodu predošlých skúsenosti s týmto štandardom a preto, že nemám k dispozícií dostatočné časové kapacity na získanie skúsenosti s technológiou GraphQL. SOAP do úvahy nepripadá z dôvodu, že moja aplikácia nebude robustná do takej miery, v ktorej je rozumné používať SOAP. Vzhľadom na to, že sa jedná

o prvú verziu, je možné sa rozhranie bude časom meniť a dokonca, že zvolím inú technológiu pre vývoj rozhrania.

Časť aplikačného rozhrania bude implementovaná pomocou technológie WebSocket a to z dôvodu potreby komunikácie v reálnom čase v module s vytváraním a správou transakcií.

6.4 Nasadenie aplikácie

Vzhľadom na to, že je projekt vyvíjaný spoločne s kolegom Martinom Nešněrom, ktorý sa venuje vývoju frontend aplikácií, je potrebné vystavenie serverovej aplikácie na verejnom servery, aby bola dostupná bez nutnosti spustenia aplikácie na lokálnom počítači. Preto sa v tejto kapitole venujem dostupným riešeniam nasadenia aplikácie na server. Neskôr vyberiem riešenie pre momentálnu vývojovú fázu a následne pre fázu produkčnú, kde sa toto riešenie môže líšiť.

Začínajúce projekty a malé firmy si nemôžu dovoliť prevádzkovať vlastné servery pre beh a vystavenie ich aplikácií, a väčšina týchto projektov využíva Cloudové služby, ktoré nám umožňujú jednoducho nasadiť aplikácie na vzdialený server poskytovateľa Cloudových služieb. Tento typ služby sa nazýva PaaS.

6.4.1 Dostupné technológie

V nasledujúcej podkapitole porovnám najznámejšie PaaS služby, ktoré predstavujú potencionálnych kandidátov, pomocou ktorých prebehne nasadenie mojej aplikácie na server. Tieto technológie boli vybrané na základe predchádzajúcich skúsenosti s týmito technológiami, alebo na základe rozsiahleho analyzovania technológií na internete.

Azure

Pri narastajúcom dopyte po Paas službách sa Microsoft rozhodol vytvoriť vlastné riešenie. Platforma podporuje všetky populárne programovacie jazyky ako napríklad Java, PHP, Ruby. Nasadzovanie aplikácie prebieha pomocou Gitu, Dockeru alebo technológie Azure DevOps. Azure sľubuje dostupnosť služby až 99,95 percent. Nevýhodou tejto služby je neintuitívnosť, a tým pádom nutnosť strávenia veľkého počtu hodín učením sa tejto technológie.

Firebase

Firebase je PaaS služba od Googlu poskytujúca hosting, autentifikáciu alebo aj súborové úložisko. Integrovaním ostatných služieb od Googlu sa vývojárom otvárajú nové možnosti pre tvorbu aplikácií. Nevýhodou je vysoká cena používania služby pri narastajúcom množstve užívateľov[21].

Heroku

Zbavuje vývojárov potreby zdĺhavého nastavovania vlastného serveru. Heroku ponúka škálovateľné PaaS riešenie. Aplikácia je nasadená na takzvané dyno, čo je virtuálny kontajner bežiaci na AWS. Pri potrebe alokovať viac zdrojov pre aplikáciu, jednoducho stačí pridať nové dyno alebo zvýšiť kapacitu existujúcich. Heroku ponúka veľké množstvo vylepšení napríklad pre logovanie, zabezpečenie a mnoho ďalších. Je možná integrácia s Gitom čo umožní bezproblémové nasadenie aplikácie[21].

DigitalOcean

Táto služba má podobné špecifikácie ako Heroku. Cena za používanie služby je dokonca nižšia v porovnaní s Heroku. Veľkou nevýhodou je zdĺhavé spúšťanie aplikácie na servery, ktoré môže trvať niekedy aj 15 minút. Ďalším problémom je nedostatočná kvalita dokumentácie[21].

Zhrnutie

Dnes existuje obrovské množstvo kvalitných PaaS služieb súperiacich o užívateľov. Preto je nutné prehodnotiť, čo od tohto typu služby očakávame a v akej fáze projektu sa nachádzame. Nasleduje výber riešenia pre nasadenie mojej aplikácie vo vývojovej a produkčnej fáze projektu.

6.4.2 Vývojová fáza projektu

V tejto fáze je pre mňa dôležitá rýchlosť a jednoduchosť nasadenia aplikácie na server z dôvodu dostupnosti rozhrania pre frontend aplikáciu. Ďalej je to určite aj cena takéhoto riešenia, ktorá sa vyvíja od počtu užívateľov. To znamená, že ideálne by takéto riešenie malo mať aj bezplatnú verziu využívania služby.

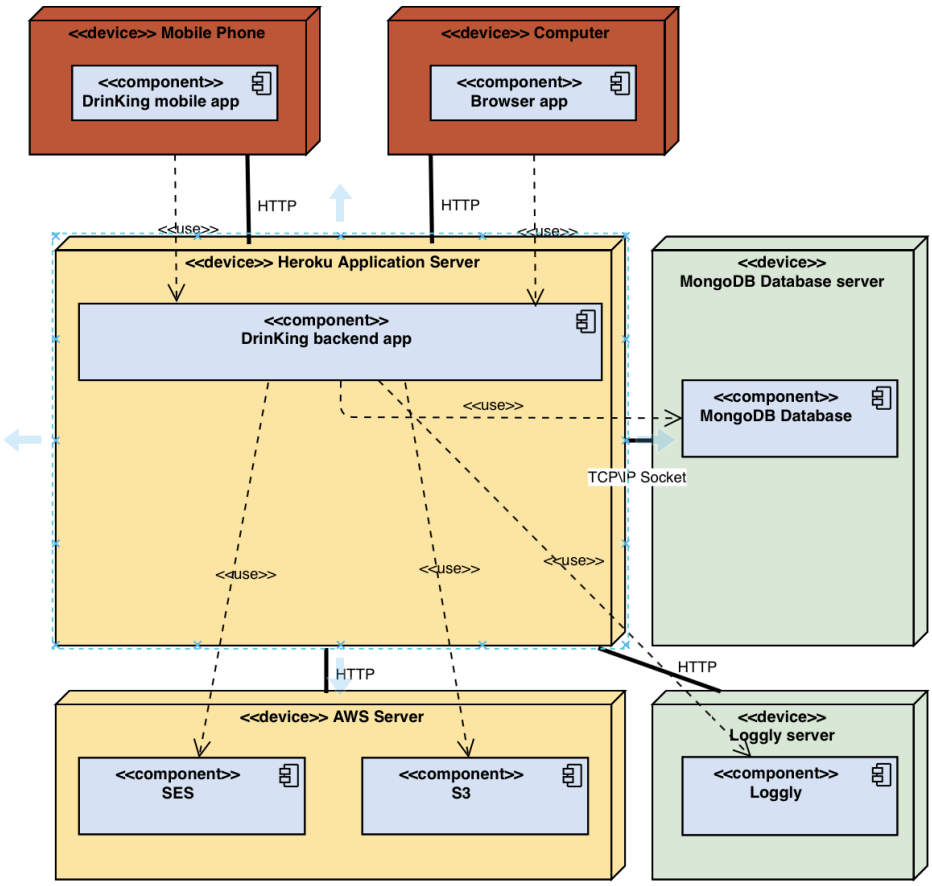
Aplikácia bude v tejto fáze fungovať vo forme Single-node aplikácie, čo znamená, že bude zároveň spustená jediná inštancia aplikácie na servery. Preto neberieme do úvahy použitie technológie Kubernetes na spravovanie zároveň bežiacich inštancií aplikácie. Najjednoduchšie rozhranie pre nasadenie aplikácie má Heroku. Zároveň je toto nasadzovanie rýchle, existuje tu bezplatná verzia využívania služby, a mám skúsenosti s používaním tejto technológie. Heroku sa teda javí ako najvhodnejšia služba pre nasadenie mojej aplikácie.

6.4.3 Diagram nasadenia

Nasledujúci diagram nasadenia na obrázku 6.2 zobrazuje rozloženie systému v súčasnej fáze, a znázorňuje akým spôsobom medzi sebou jednotlivé komponenty komunikujú.

Môžeme vidieť backend časť aplikácie, bežiacej na serveroch služby Heroku, ktorú využívajú mobilná a správcovská prehliadačová aplikácia komunikujúca

pomocou HTTP. Následne backend časť aplikácie podľa potreby využíva služby od AWS, Loggly alebo MongoDB.



Obrázok 6.2: Diagram nasadenia

6.4.4 Produkčná fáza projektu

Momentálne sa projekt nachádza vo vývojovej fáze. Je však dôležité premýšľať nad možným nasadením aplikácie do produkcie kde sa budú parametre a požiadavky na aplikáciu líšiť od vývojovej fáze. V produkčnej fáze projektu nám záleží na možnostiach škálovateľnosti a dostupnosti služby. Zároveň bude nasadzovanie aplikácie menej časte ako vo vývojovej fáze. Rozumná cena sa javí stále ako parameter na ktorý sa pri výbere služby zameriam.

Aplikácia bude v tejto fáze musieť zvládnuť obslúžiť veľké množstvo požiadaviek od užívateľov a preto bude potrebné prejsť na formu Multi-node aplikácie, čo znamená, že budú zároveň spustené viaceré inštancie aplikácie a vyvažovač zaťaženia bude jednotlivé požiadavky rozdeľovať medzi inštancie, aby nedošlo k nedostatku výpočtového výkonu. V tomto prípade začína používanie Heroku dávať menší zmysel hlavne kvôli cene, ktorá rastie rýchlejšie ako pri použití iných služieb.

V produkčnej fáze je najvhodnejšou PaaS službou pre moju aplikáciu služba Azure, a to z dôvodu rozsiahlej integrácie s technológiami Git a Docker, ďalej sa tu nachádza kvalitný monitoring aplikácií, a v neposlednom rade Azure garantuje dostupnosť služby až 99,95 percent.

6.5 Záver kapitoly

V tejto kapitole došlo k porovnaniam a výberu architektúry backend časti aplikácie, došlo k vhodnému zvoleniu technológií pre vývoj, a nakoniec sa vyriešila otázka nasadenia aplikácie na vzdialený server. V kapitole implementácia opíšem ako prebiehala implementácia jednotlivých modulov aplikácie.

Kapitola 7
Implementácia

Po analýze požiadaviek na aplikáciu, výbere technológií, ktoré budeme pri vývoji používať, nasleduje kapitola, kde sa priamo zaoberám opisom implementácie. Konkrétne, aké vývojové prostredie používam, ktoré knižnice potrebujem pre vývoj, popíšem, aké hlavné návrhové vzory boli použité a nakoniec popíšem najdôležitejšie moduly aplikácie a ich funkcionalitu.

7.1 Vývojové prostredie

Vzhľadom na vývoj aplikácie v Node.js a predchádzajúcich skúsenostiach s vývojovým prostredím WebStorm od spoločnosti JetBrains, som sa rozhodol pri používaní spomínaného prostredia ostať. Výhodou je, že fakulta ponúka licenciu tohoto nástroja pre študentov zdarma. Pre nastavovanie prístupových práv a kontrolu prítomných kolekcií v databázy používam webovú aplikáciu Atlas od MongoDB.

7.2 Štruktúra projektu

Nasledujúca podkapitola zobrazuje adresárovú štruktúru projektu. Vynechal som jednotlivé skripty z adresárov controllers, models, routes, units a konfiguračné súbory.

```
controllers ....... Adresár obsahuje jednotlivé kontroléry
                    reprezentujúce biznis logiku aplikácie.
docs
  swagger.json ... JSON súbor obsahujúci opis komunikačného
                   rozhrania aplikácie pre Swagger.
events
  eventEmmiter.js ... Skript slúžiaci na získanie novej inštancie
                      emitra udalostí.
logs
  loggly.js ...... Konfiguračný súbor logovacej služby Loggly.
```

```
├── middleware ..... Adresár obsahujúci middleware skripty validačnej
│                    a autorizačnej služby. Takisto sa tu nachádza
│                    skript na limitovanie počtu požiadaviek v určitom
│                    čase.
│   ├── validation
│   │   ├── socketValidationHandler.js
│   │   ├── utils.js
│   │   ├── validationHandler.js
│   │   └── validationSchemas.js
│   ├── authorization.js
│   └── limiters.js
├── models ......... Adresár obsahujúci konkrétne schémy definujúce
│                    dátový model systému pre MongoDB databázu.
├── notifications ... Adresár obsahujúci skripty využívajúce knižnicu
│   │                SES zodpovednú za vytváranie a posielanie
│   │                emailov.
│   ├── emailCreator.js
│   └── emailSender.js
├── routes ......... Adresár obsahujúci definície jednotlivých
│                    komunikačných bodov rozhrania. Miesto kde sa
│                    určuje použitie middlewarových služieb.
├── s3
│   └── s3.js ........ Zabalená služba využívajúca knižnicu S3 na
│                      ukladanie obrázkov.
├── schedulers ...... Adresár obsahujúci službu na plánovanie úloh.
│   ├── eventScheduler.js
│   └── transactionScheduler.js
├── sockets ......... Adresár obsahujúci modul zabezpečujúci soketovú
│   │                komunikáciu.
│   ├── barSocketHandler.js
│   ├── socketConsumer.js
│   └── transactionSocketHandler.js
├── test ............ Adresár obsahujúci modul s automatizovanými
│   │                testami.
│   ├── data ......... Adresár JSON súbory s testovacími dátami.
│   ├── db.js
│   │   └── mongoConnection.js ... Skript slúžiaci na pripojenie do testovacej
│   │                                databázy.
│   ├── units
│   └── wrapper.js ... Hlavný skript spúšťajúci jednotlivé testy modulov
│                      definované v adresári units.
├── .env
├── app.js .......... Hlavný skript obsahujúci logiku vykonávajúcu sa
│                     pri spúšťaní aplikácie.
├── enums.js
└── package.json
```

7.3 Použité knižnice

Pre správu knižníc v Node.js sa používa predinštalovaný manažér balíčkov npm. Vývojári sú pomocou neho schopný zdieľať vlastné, a používať ostatné knižnice od vývojárov z celého sveta.

Knižnice, ktoré sú použité v mojej aplikácií, prechádzajú kontrolou cez stránku npmjs.com, kde je možné vidieť počet stiahnutí za týždeň, ďalej objem daného balíčka na stiahnutie a nakoniec aj počet nahlásených chýb. Vývojár si dokáže na základe týchto údajov usúdiť, či je vhodné danú knižnicu použiť alebo nie. V nasledujúcich podkapitolách uvádzam najdôležitejšie knižnice použité v mojej aplikácii.

7.3.1 Nodemon

nasledujúca knižnica sa používa vo fáze vývoja ako monitor vykonanej zmeny na projekte. Eviduje sa každá zmena kódu a po uložení súboru sa aplikácia automaticky reštartuje, a tým nás odľahčí od nutnosti manuálne reštartu aplikácie[22]. Pre spustenie aplikácie pomocou Nodemonu je potrebné spustiť skript určený na vývoj. Vývojový skript označený na modro môžeme vidieť na nasledujúcom obrázku.

```
"scripts": {
  "start:prod": "NODE_ENV=dev node app.js",
  "start:dev": "NODE_ENV=dev nodemon app.js",
  "start:test": "NODE_ENV=test mocha --timeout 20000 --exit"
},
```

Obrázok 7.1: Screenshot skriptov v konfiguračnom súbore

Stránka npmjs.com zaznamenala štyri milióny stiahnutí tejto knižnice za týždeň, a eviduje doposiaľ 36 nahlásených chýb. Je licencovaná MIT a aj preto sa použitie knižnice Nodemon javí ako spoľahlivé a bezpečné.

7.3.2 Socket.io

Umožňuje obojsmernú komunikáciu v reálnom čase. Jednotlivé sokety je možné pridávať do miestností, a následne jednoducho správu poslať všetkým soketom, ktoré sa v tejto miestnosti nachádzajú. Výhodou je spoľahlivosť, keď k nadviazaniu spojenia dochádza aj za prítomnosti proxy servera alebo vyvažovača zaťaženia. Stránka npmjs.com eviduje štyri milióny stotisíc stiahnutí týždenne, pričom doposiaľ zaznamenala 81 nahlásených chýb. Licencovaná MIT[23].

7.3.3 Joi

Je knižnica slúžiaca na validáciu dát. V mojej aplikácií sa využíva na validáciu dát, ktoré sa nachádzajú v tele HTTP požiadaviek. Používam ju spôsobom, že

pre každý typ požiadavku vytvorím k nemu prislúchajúcu validačnú schému. Následne aplikácia na základe URI požiadavku vyhodnotí, ktorú validačnú schému je potrebné aplikovať. Ak dáta nie sú validné, aplikácia vráti príslušnú chybovú hlášku. Stránka npmjs.com eviduje pať miliónov šesťstotisíc stiahnutí týždenne, pričom doposiaľ zaznamenala 115 nahlásených chýb[24].

7.3.4 Express

Táto knižnica je základným stavebným prvkom mojej aplikácie. Je to minimalistický webový framework poskytujúci mnoho funkcionality pre webové a mobilné aplikácie. V mojej aplikácií sa stará hlavne o smerovanie jednotlivých HTTP požiadaviek na príslušné kontroléry. Stránka npmjs.com eviduje dvadsať miliónov tristotisíc stiahnutí týždenne a doposiaľ bolo nahlásených 101 chýb. Táto knižnica je licencovaná MIT[25].

7.3.5 Mongoose

Pomocou tejto knižnice je možné jednoducho namodelovať priamo v zdrojovom kóde schémy jednotlivých dokumentov, ktoré sa budú ukladať do MongoDB. Mongoose obsahuje zabudovanú kontrolu dátových typov a validáciu. Stránka npmjs.com eviduje jeden a pol milióna stiahnutí týždenne a doposiaľ bolo nahlásených 333 chýb. Znovu sa jedná o knižnicu licencovanú MIT[26].

7.4 Dôležité moduly aplikácie

nasledujúca podkapitola je venovaná opisu funkcionality a implementácie jednotlivých modulov, ktoré považujem za dôležité, a predstavujú najzaujímavejšiu funkcionalitu.

7.4.1 Autorizácia

Autentifikačný modul je do aplikácie zapojený pomocou middlewaru. To znamená, že akonáhle príde HTTP požiadavka na aplikáciu a router rozozná adresu na ktorú bola táto požiadavka poslaná, začne sa vykonávať middleware reťazec, kde sa nachádza aj autorizačná funkcia.

Na to aby som bol schopný obmedziť prístup k jednotlivým endpointom aplikácie, existujú tri typy autorizačnej funkcie, podľa toho aká rola má právo volať daný endpoint rozhrania. Na nasledujúcom obrázku 7.2 môžeme vidieť, že endpoint na zablokovanie užívateľa smie použiť jedine správca aplikácie.

```
router.post( path: "/ban/:userId",auth.authorize_a, UserController.setBanStatus);
```

Obrázok 7.2: Screenshot middleware reťazca

Konkrétna autorizačná funkcia kontroluje prítomnosť prístupového tokenu v HTTP požiadavku a následne zistí akú má užívateľ rolu. Ak splňuje požiadavok

na typ role, middleware reťazec pokračuje ďalej a užívateľovi je povolené daný endpoint rozhrania použiť.

V prípade ak je prístupový token expirovaný, je možné poslať požiadavok na obsahujúci platný refresh token. Na základe platnosti tokenu je následne klientovi zaslaný nový a platný prístupový token. Tento autorizačný model spĺňuje štandard OAuth 2.0[27].

7.4.2 Plánovač úloh

nasledujúci modul slúži na plánovanie úloh v čase. Konkrétne sa jedná o situácie, keď je nutné zavolať aktualizačnú funkciu nastavujúcu stav udalosti v čase jej začiatku a následne aj v čase konca udalosti. Ďalšou situáciou kedy potrebujeme plánovať úlohu je v čase expirácie transakcie.

Na plánovanie úloh využívam knižnicu node-schedule. Vytvoril som zabalenú službu využívajúcu funkcionalitu tejto knižnice a následne v kontrolóre v momente vytvorenia udalosti túto službu volám a nastavím dve úlohy na aktualizáciu stavu udalosti. Zabalená služba plánovača úloh využívajúca knižnicu node-schedule vyzerá následovne ako môžeme vidieť na obrázku 7.3.

```
const scheduler = require("node-schedule");
const eventController = require("../controllers/eventController");

module.exports.scheduleEventUpdateJob = (uniqueName, date) => {
  console.log("Scheduling job in scheduler module : " + uniqueName + "to date : "+ new Date(date));
  scheduler.scheduleJob(uniqueName, new Date(date), () => {
    eventController.updateEventStatusById(uniqueName).catch(err =>  console.log(err));
  });
};

module.exports.cancelEventUpdateJob = (uniqueName) => {
  let job = scheduler.scheduledJobs[uniqueName];
  if(job) job.cancel();
};
```

Obrázok 7.3: Screenshot zabalenej služby plánovača úloh

Využitie plánovača úloh v kontroléry udalostí môžeme vidieť na obrázku 7.4.

```
module.exports.createEvent = (req, res, next) => {
  eventModel
    .createEvent(req.body.event)
    .then((result) => {
      if (!!result) {
        console.log("Scheduling job for event : " + result._id.toString())
        scheduler.scheduleEventUpdateJob(
            result._id.toString(),
            result.start_date
        );
        req.body.event.event_id = result._id;
        next();
      }
    })
    .catch((err) => {
      console.log(err);
      res.status(500).send();
    });
};
```

Obrázok 7.4: Screenshot funkcie slúžiacej pre vytvorenie udalosti

7.4.3 Validácia vstupných hodnôt

Na validáciu vstupných hodnôt používam knižnicu Joi spomínanú v kapitole 7.3.3. Validácia vstupov funguje podobne ako autorizácia, a to znamená, že je vykonávaná v middleware reťazci.

Validačná služba najskôr poskladá celú URL adresu a následne k prislúchajúcej adrese získa validačnú schému. Ak je pri validácií zistená chyba, aplikácia chybový návratový kód s príslušným opisom chyby.

7.4.4 Emailové notifikácie

Notifikačný modul obsahuje zabalenú službu využívajúcu službu SES od AWS slúžiacu na posielanie emailov. Modul sa skladá celkovo z dvoch skriptov, kde prvý slúži na tvorbu emailov podľa potrebného typu emailu a personalizovaných dát o užívateľovi. Druhý skript slúži na samotné odosielanie emailov.

Odoslanie emailov nastáva pri vytvorení a ukončení udalosti, pri schválení alebo zamietnutí transakcie, a nakoniec pri schválení alebo zamietnutí účtenky.

7.5 Záver kapitoly

Na začiatku kapitoly som opísal aké vývojové prostredie bolo použité na vývoj, následne bola opísaná štruktúra projektu. Potom už nasledovalo opisovanie knižníc, ktoré som na vývoj použil, a následne aj opis jednotlivých modulov aplikácie.

Kapitola 8

Nákladová analýza behu aplikácie

Používanie cloudových služieb a služby Heroku na hostovanie aplikácie predstavuje finančné náklady. Množstvo služieb môžeme používať do určitej miery zdarma, avšak náklady rastú napríklad s počtom odoslaných mailov pri službe SES alebo od výpočtového výkonu pridelenej mojej aplikácií ak sa zameriame na Heroku. Preto je potrebné si tieto náklady vyčísliť a skúsiť analyzovať aké náklady nás môžu čakať v prípade nárastu užívateľov v produkčnej fáze.

Preto sa nasledujúca kapitola venuje vyčísleniu nákladov vo vývojovej fáze a následne vo fáze produkčnej pri odhadovanom počte užívateľov.

8.1 Vývojová fáza projektu

Vo vývoji sa jednotlivé služby používajú na testovanie funkčnosti aplikácie a nasadená aplikácia nemusí čeliť veľkému počtu požiadaviek. Preto všetky služby, ktoré momentálne využívame sú bezplatné.

- **Heroku:** Free and Hobby tarif, ktorý je zdarma. K dispozícií je 512MB RAM.
- **AWS Simple Email Service:** 1000 mailov mesačne je zdarma. Dostatočné množstvo na testovacie účely.
- **AWS S3 úložisko:** 5GB úložiska na 12 mesiacov zdarma. Dostatočná kapacita a časový priestor na testovacie účely.
- **Loggly:** Počiatočná verzia zdarma.

8.2 Produkčná fáza projektu

Nasledujúca podkapitola počíta s nasadením projektu do produkcie. Z výpočtov v práci kolegu Martina Nešněry[29] vychádza, že na to aby sa pokryli celkové náklady na projekt, je nutné aby sa konalo 22 udalostí mesačne s priemerným počtom účastníkov 30. V priemere by sa jednalo o 660 pridaných fotografií účteniek mesačne.

Ako prvé je nutné zistiť, koľko približne bude mesačne poslaných mailov, uložených obrázkov, a akú výpočtovú kapacitu budeme pre beh aplikácie potrebovať.

Pri počítaní so 660 užívateľmi (každý pridá mesačne jednu účtenku), sa dá očakávať nárast počtu požiadaviek na aplikáciu, no nie však do takej miery, aby to ovplyvnilo chod aplikácie. Preto nie je nutné navyšovať výpočetnú kapacitu pre aplikáciu. Je však potrebné prejsť na Heroku Hobby Plan, ktorý beží nepretržite aj napriek tomu, že na aplikáciu nejakú dobu neprichádzajú žiadne požiadavky.

Čo sa týka AWS SES služby, pri predpoklade vzniku 22 udalostí mesačne, budeme odosielať 37 400 emailov pre potrebu oboznámenia užívateľov o registrácií užívateľa, notifikácii o novej alebo ukončenej udalosti, a nakoniec o výsledku transakcie.

AWS S3 služba poskytuje bezplatné používanie 5 GB úložiska po dobu 12 mesiacov. Preto musíme počítať s tým, že naďalej túto službu bezplatne využívať nebudeme. Pri 660 užívateľoch, bude postačovať úložisko v objeme 3GB. Rátame s priemernou veľkosťou súboru 2 MB a jedná sa o 660 profilových fotiek užívateľov, 660 fotiek účteniek a zostáva nám rezerva na fotografie podnikov. Tento objem dát je však dostatočný len na prvý mesiac chodu aplikácie.

Pri vzniku produkčného a testovacieho prostredia bude nutné odlíšiť jednotlivé indexy logov. Služba Loggly túto funkcionalitu poskytuje v tarifu Standard.

- **Heroku:** 158,65 Kč mesačne za tarif Hobby obsahujúci 512 MB RAM.
- **AWS Simple Email Service:** Poslanie približne 37 400 emailov mesačne výjde na 77,05 Kč.
- **AWS S3 úložisko:** 62,76 Kč mesačne za 3 GB úložiska pri ktorom je mesačne súbor desaťkrát získaný z úložiska.
- **Loggly:** 1 780 Kč mesačne za tarif Standard obsahujúci možnosť sledovania troch rôznych indexov.

Všetky náklady na beh aplikácie v produkčnej fáze za predpokladu 660 aktívnych užívateľov za prvý mesiac činia 2 076,46 Kč.

Kapitola 9
Testovanie

Testovanie je jednou z fáz vývoja softvéru. Existuje mnoho typov testovania, a to napríklad automatizované testy, užívateľské testy, záťažové testy, procesné testy ale napríklad aj penetračné testy. Táto kapitola sa zaoberá a opisuje prvé dva typy testov a to automatizované testy a užívateľské testy.

9.1 Automatizované testy

Prvým typom testovania mojej aplikácie sú automatizované jednotkové testy, ktoré testujú jednotlivé jednotky kódu a to funkcie. Pre vytváranie testov som použil knižnice Mocha a Chai. V prvej podkapitole popíšem použité knižnice a následne popíšem ako sa testy pomocou týchto knižníc tvoria a aká funkcionalita bola týmito testami pokrytá.

9.1.1 Mocha a Chai

Mocha je testovací framework, podporujúci asynchrónne testy. Testy sa vykonávajú sériovo, čo nám zaručuje flexibilné a presné hlásenia.

Na druhej strane stojí porovnávacia knižnica Chai, ktorá môže byť spárovaná s ľubovolným testovacím frameworkom, avšak skoro stále ju uvidíme používanú s frameworkom Mocha. Poskytuje nám konštrukty ako napríklad Expect alebo Should.

9.1.2 Tvorba testov

Pred tvorbou samotných testov je potrebná inštalácia knižníc pomocou npm. Akonáhle máme knižnice nainštalované vytvoríme si skript, ktorý vykoná pred spustením testov pripojenie do testovacej databázy a následne bude sériovo spúšťať jednotlivé testy ako to môžeme vidieť na nasledujúcom obrázku 9.1.

```
describe( title: "All tests wrapper",  fn: function () {
  it( title: "should pass",  fn: function () {
    return db
      .connect() Promise<Mongoose>
      .then(function () {
        console.log("MongoDB is running");
        require("./units/users");
        require("./units/transactions");
        require("./units/events");
        require("./units/bars");
        require("./units/receipts");
      }) Promise<void>
      .catch((err) => {
        console.log(err);
      });
  });
});
```

Obrázok 9.1: Screenshot hlavného skriptu pre automatizované testovanie

Mocha umožňuje vydefinovať aké operácie sa majú vykonať pred samotným vykonaním testov, ale aj po vykonaní posledného testu. V mojom prípade sa jedná o vyčistenie testovacej databázy a následného naplnenia databázy testovacími dátami. Vždy po poslednom teste sa konkrétna databázová kolekcia vymaže. Nakoniec môžeme začať písať konkrétne testy s vedomím, že testovacie dáta sú pripravené a môžeme overiť pomocou knižnice Chai či je očakávaná hodnota rovnaká ako tá reálna. Na nasledujúcom obrázku 9.2 môžeme vidieť, že znovu vykonáme prečistenie testovacej databázy pred spustením testov a následne začíname vykonávať testy.

```
describe( title: "Transactions",  fn: () => {
  before( fn: (done : Done ) => {
    console.log("transaction tests");
    Promise.all( values: [
      TransactionModel.deleteAll(),
      BarModel.deleteAll(),
      TransactionModel.insertMany(data.initTransactionsData),
      BarModel.insertMany(data.initBarsData),
    ]).then(() => {
      UserModel.createAdminAccount(data.adminData).then((user) => {
        token = jwt.sign( payload: { userId: user._id }, config.secret,  option
          expiresIn: "1h",
        });
        done();
      });
    });
  });

  /*
   * Test the /POST addTransaction route valid path
   */
  describe( title: "/POST addTransaction",  fn: () => {
    it( title: "it should ADD the transaction",  fn: (done : Done ) => {
      chai
        .request(server) Agent
        .post( url: "/api/transactions/") SuperAgentRequest
        .send(data.addTransactionData) SuperAgentRequest
        .set("accesstoken", token) SuperAgentRequest
        .end( callback: (err, res : Response ) => {
          res.should.have.status( code: 201);
          done();
        });
    });
  });
                                                    WebStorm 20
```

Obrázok 9.2: Screenshot použitia automatizovaných testov pomocou knižníc Mocha a Chai

9.1.3 Pokrytie testami

nasledujúca štruktúra nám ukazuje, aké moduly boli pokryté jednotkovými testami. Celkovo je 14 jednotkových testov pokrývajúcich 5 modulov. Niektoré moduly je ťažké pokryť pretože obsahujú funkcionalitu s ukladaním obrázkov do SES a preto testovanie tejto a podobnej funkcionality bolo obsiahnuté v užívateľských testoch.

```
units
├── bars.js
├── events.js
├── receipts.js
├── transactions.js
└── users.js
```

9.2 Užívateľské testy

Druhým spôsobom overenia správneho fungovania mojej aplikácie sú užívateľské testy. Na toto testovanie je potrebné zapojenie skupinky testerov, ktorým je na začiatku predstavená samotná klientska aplikácia a následne sú oboznámení aj s testovacími scenármi. Po tomto úvode začínajú testovacie scenáre vykonávať podľa určených pokynov. Ak narazia na chybu, zaznačia ju do formulára. Na toto testovanie sme spolu s kolegom Martinom Nešněrom zavolali 5 spolužiakov, ktorí nám s testovaním pomáhali.

9.2.1 Testovacie scenáre

nasledujúca podkapitola len stručne vymenuje testovacie scenáre, ktorými bola aplikácia otestovaná. Všetky detaily ohľadne jednotlivých scenárov je možné nájsť v bakalárskej práci kolegu Martina Nešněry[29].

- **TC1 Registrácia nového užívateľa v mobilnej aplikácií**
- **TC2 Prihlásenie užívateľa v mobilnej aplikácií**
- **TC3 Prihlásenie užívateľa vo webovej aplikácií**
- **TC4 Zmena hesla užívateľa**
- **TC5 Vytvorenie účtu podniku**
- **TC6 Vytvorenie udalosti**
- **TC7 Schválenie účtenky**
- **TC8 Ukončenie udalosti**
- **TC9 Potvrdenie transakcie**
- **TC10 Zablokovanie užívateľa**

9.2.2 Výsledok užívateľského testovania

S výsledkom testovania sme boli nadmieru spokojní, pretože všetci testery prešli úspešne všetky testovacie scenáre bez pádu aplikácie alebo bez závažnejšieho problému. Čo sa týka fungovania serverovej aplikácie, boli objavené viaceré nedostatky, ktoré musia byť do budúcna odstránené. Jednalo sa hlavne

o chybu s načítaním obrázkov v profile podniku. Akonáhle nemal podnik nastavenú profilovú fotku a užívateľ chcel načítať profil tohoto baru, došlo k chybe, pretože hodnota profilovej fotky mala hodnotu undefined. Ďalej ide o chyby v niektorých validačných schémach obsahujúcich pravidlá pre dĺžku a obsah užívateľského mena, emailu a telefónneho čísla. Neskôr bola objavená chyba s vypočítavaním kreditu pre výhercu v udalosti, kde táto hodnota nebola správne zaokrúhlená.

Kapitola 10
Záver

Cieľom tejto bakalárskej práce bol návrh a implementácia serverovej aplikácie pre platformu na podporu gastronómie. Na začiatku došlo k analýze existujúcich riešení s podobným zámerom a porovnali sme ich výhody a nevýhody. Na základe analýzy bolo zistené, že výhodou našej platformy je nulová vstupná investícia a propagácia podnikov cez túto platformu. Následne došlo k analýze požiadaviek na aplikáciu a k definovaniu doménového modelu. V návrhovej časti bola zo začiatku zvolená vhodná architektúra aplikácie podľa potrieb, neskôr došlo k porovnaniu a výberu technológií vhodných na implementáciu aplikácie. V implementačnej časti došlo k popisu použitých knižníc a k popisu dôležitých modulov aplikácie. Aplikácia bola následne otestovaná pomocou automatizovaných testov a užívateľských testov.

Všetky funkčné požiadavky na aplikáciu boli splnené. Aplikácia je nasadená a beží na verejnej adrese, kde je vystavené REST rozhranie a spracováva požiadavky od klientských aplikácií.

Tvorba serverovej aplikácie v Node.js na zelenej lúke s použitím množstva integrácií s cloudovými službami bola pre mňa na začiatku výzvou ale nakoniec som sa jednotlivé služby naučil používať, porozumel a implementoval som bezpečný spôsob autorizácie a následne sa mi podarilo aplikáciu úspešne otestovať.

Zároveň boli splnené všetky ciele stanovené v úvode tejto bakalárskej práce, a teda podarilo sa navrhnúť a implementovať backend časť aplikácie na podporu gastronomických podnikov v období po pandémii Covid-19.

10.1 Ďalšie kroky

Napriek tomu, že aplikácia splnila všetky požiadavky a prešla testovaním, je tu priestor na vylepšenia. Jedná sa napríklad o skript na naplánovanie úloh pri štarte aplikácie, ktoré boli zrušené pri nasadení aplikácie a to z dôvodu, že použitý plánovač úloh nie je perzistentný. Ďalej sa jedná o skript, ktorý bude spúšťaný pravidelne, a jeho úlohou bude mazať dáta z databázy, ktoré už nie sú aktuálne a zbytočne zaberajú kapacitu úložiska. Jednou veľkou zmenou bude zmena stratégie nasadzovania aplikácie v produkčnej fáze, pred ktorou musí prebehnúť rozsiahla analýza a naberanie vedomostí o technológiach ako je Docker alebo Kubernetes.

Čo sa týka projektu ako takého, keďže sa jedná o prvú verziu projektu, nebolo zatiaľ určené či sa pokúsime spoločne s kolegom Martinom Nešněrom projekt spustiť v produkcií. Predchádzalo by tomu masívnejšie hľadanie spoluprác s podnikmi, začiatok marketingovej činnosti a taktiež by bolo nutné vyriešiť právne náležitosti. Avšak ak by sme sa k tomuto kroku podujali, myslím si, že projekt má čo užívateľom ale aj podnikom ponúknuť. V prípade užívateľov sa jedná o možnosť získania kreditu v ich obľúbenom podniku a ušetrenia tak prostriedkov na ďalší nákup, a v prípade podniku sa jedná o zvýšenie povedomia verejnosti o podniku, a s tým spojený nárast príjmov.

10.2 Ponaučenia

Vďaka tomu, že s niektorými použitými technológiami a implementačnými postupmi som doposiaľ nemal skúsenosti, vyskytli sa ponaučenia, ktoré budem schopný aplikovať do ďalších projektov. V nasledujúcom zozname sú uvedené ponaučenia z implementácie a návrhu serverovej aplikácie v Node.js.

- **Logovanie :** logovanie v konzole nie je vždy ideálne riešenie, a preto správna infraštruktúra logov by mala byť prioritnou úlohou pri návrhu systému.
- **Štruktúra kódu :** pri vyvíjaní aplikácie s veľkým počtom modulov je niekedy ťažké udržať správne konvencie písania a štruktúrovania kódu, a preto by som v ďalšom projekte použil nástroj na statickú analýzu kódu.
- **Použitie Typescriptu :** pre implementáciu novej aplikácie by som namiesto Javascriptu použil Typescript, ktorý napríklad zabezpečuje typovú kontrolu a zavádza rozhrania. Tieto prvky v Javascripte chýbajú a niekedy sa stáva, že parameter vo funkcii je iného typu ako som očakával.

Dodatok A

Literatúra

[1] BREJČÁK Peter. *Odměna za pití piva. Česká aplikace BeerSport na to využívá umělou inteligenci a používá ji 400 tisíc lidí* [online]. CzechCrunch s.r.o. , 2021 [cit. 2022-04-11]. Dostupné z: https://cc.cz/odmena-za-piti-piva-ceska-aplikace-beersport-na-to-vyuziva-umelou-inteligenci-a-pouziva-ji-400-tisic-lidi/

[2] *Wolt Delivery: Jedlo a iné* [online]. Google Play, 2022 [cit. 2022-04-11]. Dostupné z: https://play.google.com/store/apps/details?id=com.wolt.androidhl=skgl=US

[3] *Manage loyalty with ease* [online]. TRIFFT Loyalty Cloud, 2021 [cit. 2022-04-11]. Dostupné z: https://www.trifft.io

[4] TKACHENKO Igor. *Functional vs. non-functional requirements : Main differences and examples* [online]. The App Solutions Inc. , 2021 [cit. 2022-04-11]. Dostupné z: https://theappsolutions.com/blog/development/functional-vs-non-functional-requirements/

[5] *Domain Models and Object Oriented Analysis* [online]. Newfoundland and Labrador University [cit. 2022-04-17]. Dostupné z: http://www.cs.mun.ca/ harold/Courses/Old/CS3716.W12/Diary/Rod/domain-modeling/domain-modeling.html

[6] *All You Need to Know about State Diagrams* [online]. Visual Pardigm, 2022 [cit. 2022-04-17]. Dostupné z: https://www.visual-paradigm.com/guide/uml-unified-modeling-language/about-state-diagrams/

[7] HAQ el Siraj. *Introduction to Monolithic Architecture and Micro-Services Architecture* [online]. Medium, 2018 [cit. 2022-04-17]. Dostupné z: https://medium.com/koderlabs/introduction-to-monolithic-architecture-and-microservices-architecture-b211a5955c63

[8] *What is a multi layered software architecture?* [online]. Packt Publishing Ltd, 2018 [cit. 2022-04-17]. Dostupné z: https://hub.packtpub.com/what-is-multi-layered-software-architecture/

[9] DATS Oleg. *Node.js vs Java: What to Choose in 2022?* [online]. TechMagic, 2021 [cit. 2022-04-20]. Dostupné z: https://www.techmagic.co/blog/node-js-vs-java-what-to-choose/

[10] *What is PostgreSQL?* [online]. PostgreSQL, 2022 [cit. 2022-04-20]. Dostupné z: https://www.postgresql.org/about/

[11] *What is MySQL? Everything You Need to Know* [online]. Talend, 2022 [cit. 2022-04-22]. Dostupné z: https://www.talend.com/resources/what-is-mysql/

[12] *What Is MongoDB?* [online]. MongoDB Inc., 2022 [cit. 2022-04-23]. Dostupné z: https://www.mongodb.com/what-is-mongodb

[13] *Open Source NoSQL Database* [online]. The Apache Software Foundation , 2022 [cit. 2022-04-23]. Dostupné z: https://cassandra.apache.org/_/*index.html*

[14] *Firebase Authentication* [online]. Firebase, 2022 [cit. 2022-04-23]. Dostupné z: https://firebase.google.com/docs/auth

[15] *AWS vs Azure-Who is the big winner in the cloud war?* [online]. Iconic Inc., 2022 [cit. 2022-04-24]. Dostupné z: https://www.projectpro.io/article/aws-vs-azure-who-is-the-big-winner-in-the-cloud-war/401mcetoc$_1 fuo85qrsg$

[16] *Amazon Simple Email Service* [online]. Firebase, 2022 [cit. 2022-04-24]. Dostupné z: https://aws.amazon.com/ses/

[17] *What is REST?* [online]. Codeacademy, 2022 [cit. 2022-04-24]. Dostupné z: https://www.codecademy.com/article/what-is-rest

[18] *What is GraphQL?* [online]. RedHat, 2019 [cit. 2022-05-01]. Dostupné z: https://www.redhat.com/en/topics/api/what-is-graphql

[19] *SOAP (Simple Object Access Protocol)* [online]. TechTarget, 2019 [cit. 2022-05-01]. Dostupné z: https://www.techtarget.com/searchapparchitecture/definition/SOAP-Simple-Object-Access-Protocol

[20] *What Are WebSockets, and When Should You Use Them?* [online]. PubNub, 2022 [cit. 2022-05-01]. Dostupné z: https://www.pubnub.com/guides/what-are-websockets-and-when-should-you-use-them/

[21] *10 Heroku alternatives to host your app in 2022* [online]. AutoIdle, 2022 [cit. 2022-05-01]. Dostupné z: https://autoidle.com/blog/heroku-alternatives

[22] *nodemon* [online]. npm, 2021 [cit. 2022-05-01]. Dostupné z: https://www.npmjs.com/package/nodemon

[23] *socket.io* [online]. npm, 2022 [cit. 2022-05-01]. Dostupné z: https://www.npmjs.com/package/socket.io

[24] *joi* [online]. npm, 2022 [cit. 2022-05-01]. Dostupné z: https://www.npmjs.com/package/joi

[25] *express* [online]. npm, 2022 [cit. 2022-05-01]. Dostupné z: https://www.npmjs.com/package/express

[26] *mongoose* [online]. npm, 2022 [cit. 2022-05-09]. Dostupné z: https://www.npmjs.com/package/mongoose

[27] *What is OAuth 2.0?* [online]. Auth0, 2022 [cit. 2022-05-09]. Dostupné z: https://auth0.com/intro-to-iam/what-is-oauth-2/

[28] *Chai Assertion Library* [online]. Chai Assertion Library [cit. 2022-05-10]. Dostupné z: https://www.chaijs.com/api/

[29] NEŠNĚRA Martin. *Klientska strana souťežní platformy pro podporu gastronomie* [bakalárska práca, draft]. 2022.

[30] DUBAJ Andrzej. *Multilayer architecture*[online]. 2017 [cit. 2022-05-12]. Dostupné z: https://www.handsonprogramming.io/blog/2017/03/multilayer-architecture/

Dodatok B

Obsah priloženého komprimovaného súboru

- **images** ..... Obrázky uvedené v práci.
  - **diagrams** ..... Diagramy uvedené v práci.
  - **screenshots** ..... Screenshoty uvedené v práci.
- **drinkingBackend.zip** ..... Zdrojové kódy aplikácie.
- **thesisSource.zip**..... Zdrojové kódy textu vo formáte LaTeX.
- **thesis.pdf**..... Text práce v PDF formáte.
- **partnershipInterests.docx**... Dokument s dátami o záujme podnikov spolupracovať.

Dodatok C

Zoznam použitých skratiek

Skratka	Význam
MVC	Model View Controller
NFR	Non-Functional Requirement
FR	Functional Requirement
WAR	Web Application Resource
ZIP	Typ komprimovaného súboru
JSON	JavaScript Object Notation
SES	Simple Email Service
S3	Cloudové úložisko od AWS
REST	Representational State Transfer
SOAP	Simple Object Access Protocol
HTTP	Hypertext Transfer Protocol
XML	Extensible Markup Language
PaaS	Platform as a Service
npm	Node package manager
MIT	Massachusetts Institute of Technology
URI	Uniform Resource Identifier
GB	Gigabyte
RAM	Random Access Memory
TC	Test Case
NoSQL	Not Only SQL
SQL	Structured Query Language