ÓBUDAI EGYETEM
ÓBUDA UNIVERSITY

NEUMANN JÁNOS
INFORMATIKAI KAR

SZAKDOLGOZAT

OE-NIK	Hallgató neve:	**Petrik Ádám**
2023	Hallgató törzskönyvi száma:	**T/006898/FI12904/N**

Óbudai Egyetem
Neumann János Informatikai Kar
Kiberfizikai Rendszerek Intézet

SZAKDOLGOZAT FELADATLAP

Hallgató neve: **Petrik Ádám**
Törzskönyvi száma: T/006898/FI12904/N
Neptun kódja: [redacted]

A dolgozat címe:

Web- és mobilalkalmazás fejlesztés vendéglátóhelyek számára
Web and mobile application development for restaurants

Intézményi konzulens: Rusznák Attila
Külső konzulens:

Beadási határidő: 2022. december 15.

A záróvizsga tárgyai: Számítógép architektúrák
Big Data és üzleti intelligencia specializáció

A feladat

A szakdolgozat kapcsán a hallgató feladata egy olyan applikáció és webes alkalmazás kialakítása, dokumentálása és tervezése, amely nagymértékű segítséget biztosít a mai vendéglátóhelyek számára. Az applikációnak olyan funkciókkal kell rendelkeznie, hogy a vendégek képesek legyenek megnézni milyen helyek vannak a környéken, képesek legyenek megnézni a helynek az ital/étlapját és ezen információk ismeretében tudjanak rendelést leadni az adott helyen. A webalkalmazásnak olyan folyamatokban kell támogatást nyújtania a rendelések kezelése mellett, ami mindennapos a vendéglátóhelyek életében, mint például a készletnyilvántartás, és ezen felül egy adattárház, illetve egy BI rendszer. Továbbá a hallgató feladata, hogy felmérje a piacon már elérhető megoldásokat, valamint a probléma lehetséges megközelítési módjait.

A dolgozatnak tartalmaznia kell:

- a feladat pontos ismertetését,
- a webes alkalmazás elkészítését,
- a követelmények specifikációját,
- az applikáció elkészítését,
- a fejlesztőkörnyezet ismertetését,
- a tesztadatok létrehozását az adatbetöltési folyamatokban való felhasználásra,
- a többféle adatbázis elkészítését (Postgres,Mongo),
- az adattárház kiépítését,
- a vizualizáció megvalósítását,
- a választott megoldások indoklását,
- a tervezés és megvalósítás részletes bemutatását,
- a továbbfejlesztési lehetőségek számba vételét.

Ph.

ÓBUDAI EGYETEM * Neumann János Informatikai Kar. * 1. Budapest

.....
Dr. Fleiner Rita
intézetigazgató

A szakdolgozat elévülésének határideje: **2024. december 15.**
(OE TVSz 55.§ szerint)

A dolgozatot beadásra alkalmasnak tartom:

.....
külső konzulens

.....
intézményi konzulens

HALLGATÓI NYILATKOZAT

Alulírott hallgató kijelentem, hogy a szakdolgozat / diplomamunka saját munkám eredménye, a felhasznált szakirodalmat és eszközöket azonosíthatóan közöltem. Az elkészült szakdolgozatomban / diplomamunkámban található eredményeket az egyetem és a feladatot kiíró intézmény saját céljára térítés nélkül felhasználhatja.

Budapest, 2022. december 09.

.....

hallgató aláírása

KONZULTÁCIÓS NAPLÓ

Hallgató neve: Petrik Ádám Neptun kód: [redacted] Tagozat: Nappali

[redacted]

Szakdolgozat / Diplomamunka címe magyarul:

EAZY – webes és mobilos alkalmazás vendéglátóhelyek számára

Szakdolgozat / Diplomamunka címe angolul:

EAZY – web and mobile application for catering places

Intézményi konzulens: Külső konzulens:

Rusznák Attila

Kérjük, hogy az adatokat nyomtatott nagybetűkkel írja!

Alk.	Dátum	Tartalom	Aláírás
1.	2022.03.20	A szakdolgozat témájának kiválasztása, megbeszélése	
2.	2022.04.15	Az irodalomkutatással kapcsolatos kérdések megvitatása	
3.	2022.05.02	A formai követelmények pontosítása	
4.	2022.05.10	Beadás előtti ellenőrzések	

A Konzultációs naplót összesen 4 alkalommal, az egyes konzultációk alkalmával kell láttamoztatni bármelyik konzulenssel.

A hallgató a Szakdolgozat I. / Szakdolgozat II. (BSc) vagy Diplomamunka 1 / Diplo-mamunka 2 / Diplomamunka 3 / Diplomamunka 4 tantárgy követelményét teljesítette, beszámolóra / védésre bocsátható.

Budapest, 2022. 05. 13.

Rusznák Attila

.....

intézményi konzulens

KONZULTÁCIÓS NAPLÓ

Hallgató neve: Neptun kód: Tagozat:

Petrik Ádám ..... [redacted] ..... Nappali .....

Telefon: Levelezési cím (pl: lakcím):

[redacted]

Szakdolgozat / Diplomamunka[1] címe magyarul:

Web- és mobilalkalmazás fejlesztése vendéglátóhelyek számára .....

Szakdolgozat / Diplomamunka[2] címe angolul:

Web and mobile application development for restaurants.....

Intézményi konzulens: Külső konzulens:

Rusznák Attila..... .....

Kérjük, hogy az adatokat nyomtatott nagybetűkkel írja!

Alk.	Dátum	Tartalom	Aláírás
1.	2022. 09. 16.	A szakdolgozat formai követelményének megbeszélése	
2.	2022. 10. 14.	A dolgozat felépítése, a kötelező formai elemek megadása	
3.	2022. 10. 28.	Az egyes témák, részfeladatok felépítésének megbeszélése	
4.	2022. 11. 11.	A felmerülő kérdések, formázási nehézségek egyeztetése	

A Konzultációs naplót összesen 4 alkalommal, az egyes konzultációk alkalmával kell láttamoztatni bármelyik konzulenssel.
A hallgató a Szakdolgozat I. / Szakdolgozat II. (BSc) vagy Diplomamunka 1 / Diplomamunka 2 / Diplomamunka 3 / Diplomamunka 4[3] tantárgy követelményét teljesítette, beszámolóra / védésre[4] bocsátható.
Budapest, 2022. november 21.

.....
intézményi konzulens

[1] Megfelelő aláhúzandó!
[2] Megfelelő aláhúzandó!
[3] Megfelelő aláhúzandó!
[4] Megfelelő aláhúzandó!

Tartalomjegyzék

Bevezetés

A szakdolgozatom témája egy olyan alkalmazás és applikáció elkészítése, ami nem csak a vendéglátó, hanem a vásárlói oldalról is nagy mértékű segítséget nyújt.

A szakdolgozat ötlete legelőször akkor pattant ki a fejemből, mikor bementünk a barátaimmal Budapestre, de több problémába is futottunk. Az első ilyen probléma az volt, hogy pontosan hova is menjünk, aztán mikor nagy nehezen kitaláltuk, hogy hova is üljünk be és már a helyen voltunk egy ideje szomorúan vettük észre mikor az idő a csúcsforgalomhoz közeledett, hogy egyre többet álltunk a sorban, mint amennyit együtt tölthettünk volna az asztalnál, így egy olyan ötlettem támadt, hogy milyen jó lenne, ha megtudnám egy applikáció segítségével rendelni a kívánt ételt/italt és jelezne számomra az applikáció, hogy a rendelésem elkészült és csak átkell vennem a rendelésemet. De mikor ezen elmélkedtem, hogy ez az applikáció mennyire megkönnyítené az életemet eszembe jutott az a probléma, hogy lehet, hogy tudok rendelést leadni a helynek, de a helynek is kellene egy szoftverrel kezelni a rendeléseket. Ezen felül egy olyan ötletem is támadt, hogy jó lenne, ha a hely is látná mielőtt a rendelés beérkezik hozzájuk, hogy a megrendelt termékből van e még raktáron, hogy elkerüljük azokat az eseteket mikor a rendelés megérkezik, de mégse tudja a hely teljesíteni a rendelést. Röviden fogalmazva mind a 2 félnek segítséget szeretnék nyújtani a szoftverrel és az applikációval a vásárló oldalról magát a rendelés folyamatot és a hely keresésének a terhét venném le a hely oldaláról meg egy olyan szoftver írnék, ami tudja kezelni a rendeléseket és ezen felül még magát a készletet és minden más apró műveletet, ami egy vendéglátóhely működésesben előfordulhat.

1. Feladat részletes ismertetése

A szakdolgozat témája egy applikáció és egy szoftver elkészítése az applikáción a vásárló képes legyen válogatni a vendéglátóhelyek között egy térképen és képesnek kell lennie egy kereső mező segítségével keresni is bizonyos helyre ezen felül képesnek kell lennie rendelést leadni a kiválasztott helynek azután, hogy végig nézte a helynek az étel/ital lapját ezen kívül mikor a rendelése elkészül a vásárlónak akkor a telefon képes legyen visszajelzéssel jelezni, hogy elkészült a rendelése. Az applikációnak ezen felül rendelkeznie kell egy bejelentkező felülettel, ami bejelentkezés után megjeleníti az adatainkat, nyilván valóan, ha van bejelentkező felület akkor rendelkeznie kell egy regisztrációs lappal is, ami a regisztráció után elmenti az adatainkat, mint például név, életkor stb. A szoftvernek még képesnek kell lennie, hogy a bejövő rendeléseket kezelje, ezen felül képesnek kell lennie mutatni a még el nem készített rendeléseket és képesnek kell lennie vissza jelezni az applikációnak, hogy az adott rendelés elkészült. A szoftvernek nem csak a rendelések lebonyolítására szükséges funkciókkal kell csak rendelkeznie, hanem például egy olyan fülel is, ahol magát a készletet is tudja kezelni és ha egy adott termékből kevesebb lenne az elvártnál jeleznie is kell a felhasználónak, hogy elfog fogyni az adott készlet. A szoftvernek kell még egy fül, ami az aznapi rendeléseket is megjeleníti, hogy a nap végén meglehessen nézni milyen és hány rendelés érkezett be. A szoftvernek is kell lennie egy bejelentkező rendszerrel, ahova a pultos betud jelentkezni a műszak elején és a műszak végén kitudd jelentkezni a rendszerből és ezeket menti a program, amire véleményem szerint azért van szükség, hogy ha bármi probléma lenne, mint például hiányzik a kasszából valamennyi pénz akkor a tulaj visszatudja nézni akkor kivolt műszakban. A szoftvereken kívül egy adattárházat szeretnék kiépíteni, ami menti az összes rendelést, hogy a későbbiekben BI eszközökkel megtudjuk nézni, hogy mekkora fogyás volt, mint például egy adott termék csoportból (szeszes italok) és ezek tudatában a későbbiekben döntéseket tudjunk hozni mikor az adott készlet csoportokra rendelést intézünk.

2. Kliens technológiák

A kliens (angolul client) olyan számítógép, amely hozzáfér egy (távoli) szolgáltatáshoz, amelyet egy számítógép hálózathoz tartozó másik gép nyújt. A kifejezést először önálló programmal nem rendelkező végkészülékekre, illetve terminálokra alkalmazták, amelyek legfontosabb szerepe az volt, hogy a hálózaton keresztül kapcsolatba lépjenek az időosztással működő nagyszámítógépekkel és elérhetővé tegyék azok szolgáltatásait. [1]

Jellemzői:

- Kéréseket, lekérdezéseket küld a szervernek
- A választ a szervertől fogadja
- Egyszerre általában csak kisszámú szerverhez kapcsolódik
- Közvetlenül kommunikál a felhasználóval, általában egy GUI-n (Graphical User Interface = grafikus felhasználói felület) keresztül

A feladatomban 2 klienst tervezek az egyik, amivel a vendéglátóhely tud kommunikálnia szerverrel a szoftveren keresztül a másik pedig amivel a vásárló tud kommunikálni a szerverrel az applikáción keresztül.

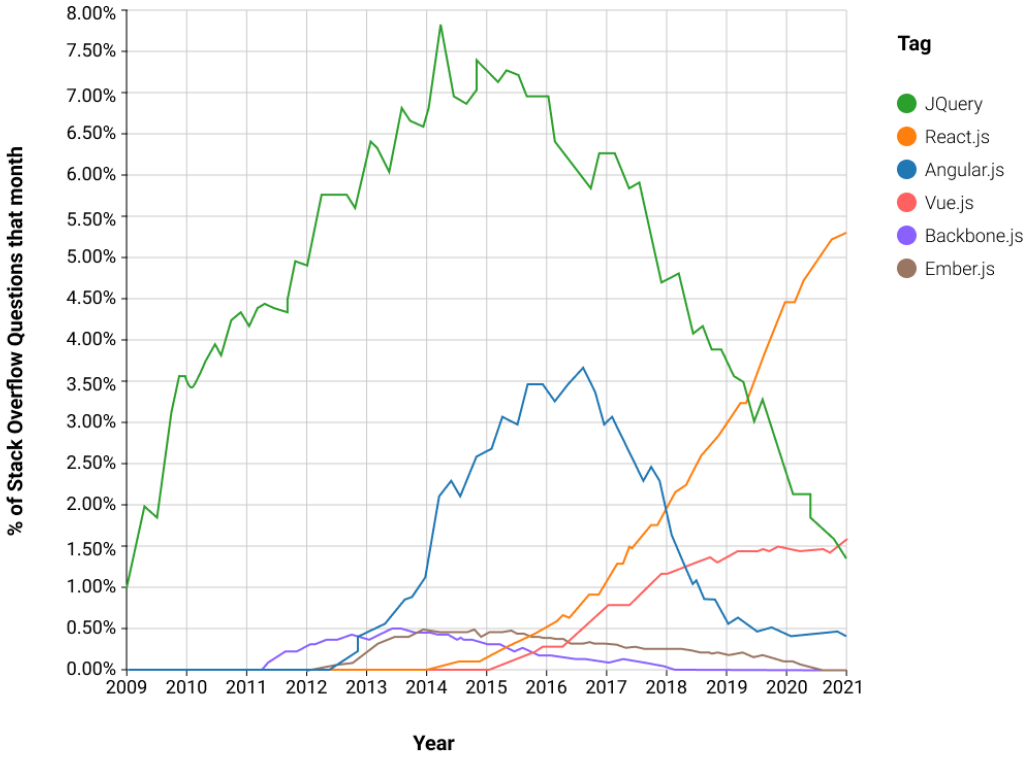
2.1 Webes kliens

A webes kliens Rest api-n keresztül kommunikál a szerverrel. Amint a szerver megkapta a kérést egy JSON objektummal fog válaszolni a kliens-nek. A Rest-es megoldás azért is tökéletes lesz a számomra mivel a Rest-et bármilyen típusú szerver (Java, PHP, NODE.js) könnyen tud kezelni és mivel nekem több szerverem lesz a kliens azonos megoldással tud kommunikálni velük. [2]

2.2 Applikáció kliens

Az applikációt is Rest api-n keresztül érkezik, mint a webes kliensnél azért választottam a Rest-et az applikációhoz is mert Native alkalmazást szeretnék írni, ami azonos technológiákat fog használni, mint maga a webes kliens.[3] A native alkalmazásokat azért jó használni mert egy köztes nyelvben írod meg az alkalmazást az én esetemben JavaScriptben és mégis képes elérni a telefon eszközeit, mint például a kamerát. A fejlesztés végén képes vagy a neked megfelelő platformra lefordítani, mint például Android vagy IOS.

Az én esetemben hosszas megfontolás után a React Native-ot választottam maga az applikáció megírásában és a webes kliensnél is a React-nél maradtam, ezt a kettő rendszert azért is nagyon jó együtt használni mert azonos technológiákat használ mind a 2 és könnyű áttérni az egyikről a másikra, ezen felül a React a legtámogatottabb Front end rendszer napjainkban.



1. ábra: Front end popularity

3. Szerver technológia

A szerver olyan számítógépet, illetve szoftvert jelent, ami más gépek számára szolgáltatja a szerveren tárolt adatokat, a szerver erőforrásait, illetve más szolgáltatásokat. A szervert és a hozzá kapcsolódó programokat kiszolgálónak is szokták nevezni. [1]

A szerver jellemzői:

- Passzív, a kliensektől várja a kéréseket
- A kéréseket feldolgozza, majd visszaküldi a választ
- Általában több klienst is kiszolgál egyszerre
- Általában a felhasználó nem áll közvetlen kapcsolatban a szerverrel

A feladatomban lévő szoftvernek és applikációnak 2 szerver biztosítja az adatokat és ezeket az adatokat, mint SQL és NoSQl adatbázisokban tárolóm el. Az utóbbi 2 adatbázis típusról picit bővebben is beszélni fogok a későbbiekben.

3.1 Back-end Szerverek

A Backendnek alapvetően az a dolga, hogy a kliens által kért adatokat szolgáltassa vagy feldolgozza azon kérdéseket. Napjainkban rengetek remek technológia van a Backend elkészítésére, mint a JAVA, PHP, JAVASCRIPT, GO, PYTHON stb. De az én választásom 2 olyan Backend szerverre esett amik napjainkban elégé népszerűek az egyik ilyen a Node.js keretrendszer ami JavaScriptet használ és a Spring keretrendszer ami meg Java-t használ, azért választottam 2 Backend rendszert is mert a Java nagyon szépen tud bánni az SQL-es adatbázisokkal a JavaScript pedig a NoSQl adatbázisokkal és mivel mind a 2 megtalálható a rendszeremben célszerű ezeket használni és mivel Rest api-n keresztül kommunikálnak a Frontenddel így probléma se adódik abból, hogy éppen melyik szerver is szolgálja ki az adott frontendet.

4. SQL adatbázis bemutatása

Először is, hogy tudjuk is, hogy mire jó picit beszélnék a történetéről.

Az SQL nyelv fejlesztése az 1970-es évek elején az IBM-nél kezdődött. Ezzel egy időben készültek el az első relációs adatbázisok (RDBMS). Az SQL kidolgozásánál elsődleges szempont volt az egyszerűség megőrzése azzal együtt, hogy rendkívül komplex lehetőséget nyújtson az adatok hozzáféréséhez. Az SQL és RDBMS rendszerek kéz a kézben fejlődtek az elmúlt 40 évben. Hatalmas adatmennyiséget kezelünk ezekkel a rendszerekkel, nagyon sokáig svájci bicska-szerűen mindenre megoldást kínáltak. Kialakultak az informatika szegmenseihez idomuló megoldások, így amikor a web elterjedt, vele együtt lettek népszerűek az egyszerűbb RDBMS rendszerek, mint a MySQL vagy a PostgreSQL. [4]

Az SQL nyelvi elemeket 4 részre, adatdefiníciós (Data Definition Language, DDL), adatkezelési (Data Manipulation Language, DML), lekérdező (QUERY (Language - QL)) és adatvezérlő (Data Control Language, DCL) részekre lehet bontani. [10]

4.1 ACID

Adatbázisok esetén az ACID az Atomicity (atomiság), Consistency (konzisztencia), Isolation (izoláció), és Durability (tartósság) rövidítése. Ezek az adatbázis-kezelő rendszer tranzakciófeldolgozó képességeinek alapelemei. Ezek nélkül az adatbázis integritása nem garantálható. [10]

4.2 Alapkövei

4.2.1 Atomicitás

Az atomicitás megköveteli, hogy több műveletet atomi (oszthatatlan) műveletként lehessen végrehajtani, azaz vagy az összes művelet sikeresen végrehajtódik, vagy egyik sem. [10]

4.2.2 Konzisztencia

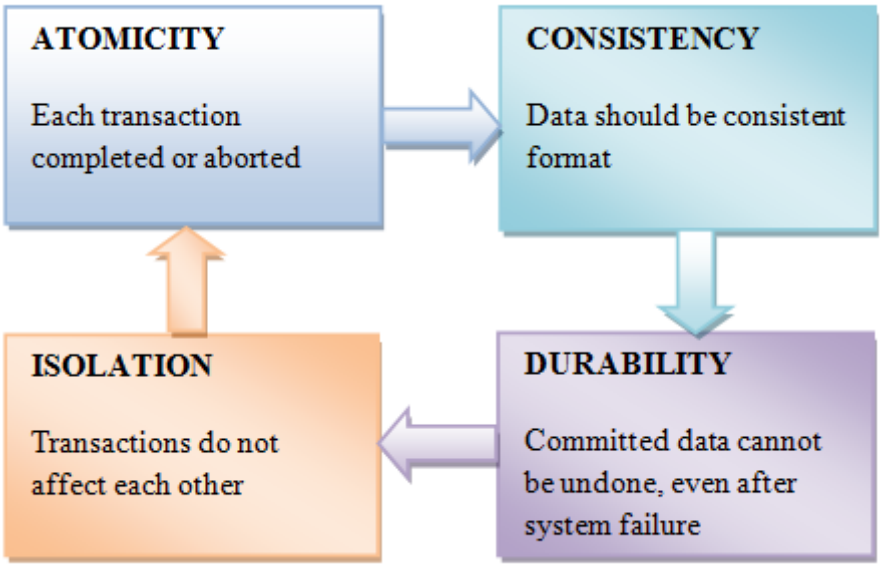
A konzisztencia biztosítja, hogy az adatok a tranzakció előtti érvényes állapotból ismét egy érvényes állapotba kerüljenek. Minden erre vonatkozó szabálynak (hivatkozási integritás, adatbázis triggerek stb.) érvényesülnie kell. [10]

4.2.3 Izoláció

A tranzakciók izolációja azt biztosítja, hogy az egy időben zajló tranzakciók olyan állapothoz vezetnek, mint amilyet sorban végrehajtott tranzakciók érnének el. Egy végrehajtás alatt álló tranzakció hatásai nem láthatóak a többi tranzakcióból. [10]

4.2.4 Tartósság

A végrehajtott tranzakciók változtatásait egy tartós adattárolón kell tárolni, hogy a szoftver vagy a hardver meghibásodása, áramszünet, vagy egyéb hiba esetén is megmaradjon. [10]

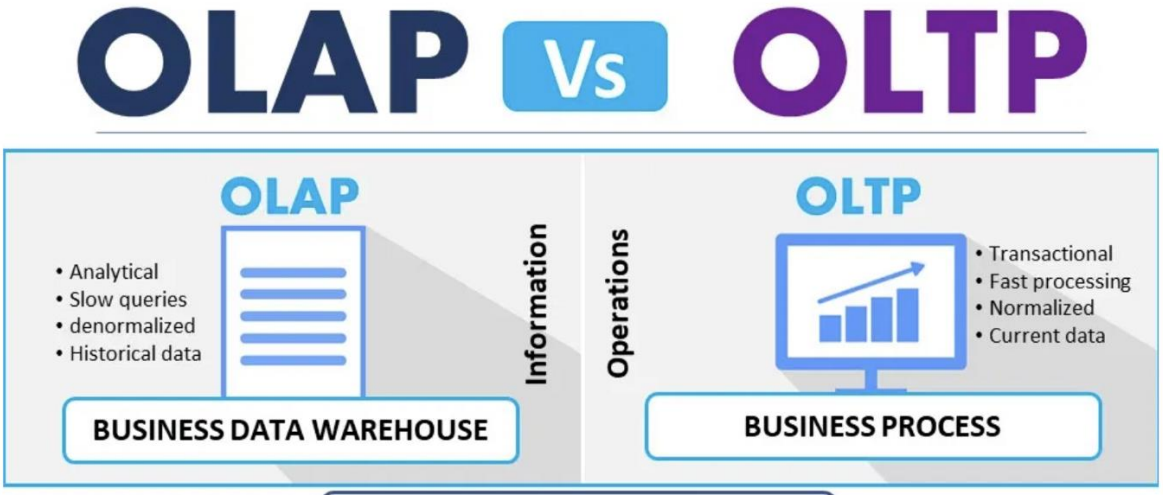


2. ábra: ACID

4.3 OLAP és OLTP

Ebben a szekcióban pár szót beszélnék az OLAP és az OLTP-ről mert nem csak simán Sql-es adatbázist szeretnék használni a szoftvernél, hanem adattárházat is kiszeretnék építeni a szakdolgozatban, így mind a kettő típusú adatbázis felépítésre szükségem lesz a szakdolgozat folyamán.

Az OLTP és az OLAP egyaránt az online feldolgozó rendszerek. Az OLTP egy tranzakciós feldolgozás, míg az OLAP egy analitikus feldolgozó rendszer. Az OLTP olyan rendszer, amely tranzakció-orientált alkalmazásokat irányít az interneten, például az ATM-en. Az OLAP egy olyan online rendszer, amely többdimenziós analitikai lekérdezéseket jelent, mint például a pénzügyi jelentések, az előrejelzések stb. Az OLTP és az OLAP közötti alapvető különbség az, hogy az OLTP egy online adatbázis-módosító rendszer, míg az OLAP egy online adatbázis-lekérdezési rendszer. [3]



3. ábra: OLAP VS OLTP

4.3.1 OLTP

Az OLTP egy online tranzakció-feldolgozó rendszer. Az OLTP rendszer fő célja az aktuális frissítés, beillesztés és törlés rögzítése tranzakció során. Az OLTP lekérdezések egyszerűbbek és rövidebbek, ezért kevesebb időt igényelnek a feldolgozásban, és kevesebb helyet is igényelnek.

Az OLTP adatbázis gyakran frissül. Előfordulhat, hogy az OLTP tranzakciója közepén sikertelen, ami az adatok integritását befolyásolhatja. Ezért különös gondot kell fordítania az adatok integritására. Az OLTP adatbázis normalizált táblázatokkal rendelkezik (3NF).

A legjobb példa az OLTP rendszerre egy ATM, amelyben rövid tranzakciókkal módosítjuk fiókunk állapotát. Az OLTP rendszer az OLAP adatforrásává válik. [3]

4.3.2 OLAP

Az OLAP egy online analitikai feldolgozó rendszer. Az OLAP adatbázis az OLTP által bevitt történelmi adatokat tárolja. Lehetővé teszi a felhasználó számára a többdimenziós adatok különböző összefoglalóinak megtekintését. Az OLAP használatával információkat gyűjthet egy nagy adatbázisból, és elemezheti a döntéshozatalhoz.

Az OLAP azt is lehetővé teszi, hogy a felhasználó komplex lekérdezéseket hajtson végre a többdimenziós adatok kinyeréséhez. Az OLTP-ben még akkor is, ha a tranzakció nem sikerül középen, az nem fogja károsítani az adatok integritását, mivel a felhasználó az OLAP-rendszert használja az adatok elemzéséhez egy nagy adatbázisból. Egyszerűen a felhasználó újra lekérdezheti a lekérdezést, és kivonhatja az adatokat elemzés céljából.

Az OLAP tranzakció hosszú, ezért viszonylag több időt vesz igénybe a feldolgozáshoz, és nagy teret igényel. Az OLAP tranzakciók ritkábbak az OLTP-hez képest. Még az OLAP adatbázisban található táblázatok sem normalizálhatók. Az OLAP példája a pénzügyi jelentés, a költségvetés, a marketing menedzsment, az értékesítési jelentés stb. Megtekintése. [3]

4.3.3 Összehasonlító táblázat

Az összehasonlítás alapja	**OLTP**	**OLAP**
Alapvető	Ez egy online tranzakciós rendszer, és kezeli az adatbázis módosítását.	Ez egy online adatgyűjtő és adatelemző rendszer.
Fókusz	Beszúrás, frissítés, információk törlése az adatbázisból.	Az adatok elemzése az elemzéshez, amely segít a döntéshozatalban.

Adat	Az OLTP és annak tranzakciói az eredeti adatforrás.	A különböző OLTP-adatbázisok az OLAP adatforrásává válnak.
Tranzakció	Az OLTP rövid tranzakciókkal rendelkezik.	Az OLAP hosszú tranzakciókkal rendelkezik.
Idő	Egy tranzakció feldolgozási ideje viszonylag kisebb az OLTP-ben.	Egy tranzakció feldolgozási ideje viszonylag nagyobb az OLAP-ban.
Lekérdezések	Egyszerűbb lekérdezések.	Komplex lekérdezések.
Normalizálás	Az OLTP adatbázis táblázatai normalizálva vannak (3NF).	Az OLAP adatbázisban található táblázatok nem normalizálódnak.
Becsületesség	Az OLTP adatbázisnak meg kell őriznie az adatok integritásának korlátozását.	Az OLAP adatbázis nem módosul gyakran. Ezért az adatok integritását nem érinti.

1. táblázat: OLAP vs OLTP

5. NoSQL adatbázis bemutatása

A NoSQL-adatbázisokat egymással felcserélhető módon a „nemrelációs”, a „NoSQL DB” és a „nem SQL” elnevezéssel is hivatkozzák, kihangsúlyozva azt a tényt, hogy gyorsan változó, strukturálatlan adatok nagy tömegének kezelésére képesek, a sorokat és oszlopokat kezelő relációs (SQL-) adatbázisoktól eltérő módokon. [5]

NoSQL-technológiák már az 1960-as években is léteztek különböző neveken, a népszerűségük mégis most van emelkedőben, amikor az adatok eloszlása megváltozik, a fejlesztőknek pedig alkalmazkodniuk kell a felhőben, a mobileszközökön, a közösségi médiában keletkező, és a big data jellegű adatok roppant tömegéhez és óriási változatosságához. [5]

A weben megjelenő alkalmazások új kihívások elé állították az RDBMS rendszereket: hatalmas adatmennyiségből a másodperc tört része alatt kell eredményeket produkálniuk. Sokáig nem mutatkoztak a gyengeségeik. A kilencvenes évek végén az internet robbanásszerű elterjedésével azonban néhány cégen belül hatalmas adattömegek gyűltek össze. Ekkor ütköztek falba az ezeket a rendszereket építő fejlesztők, ami arra kényszerítette őket, hogy új típusú adattárolási és adatfeldolgozási eljárásokat fejlesszenek. Először az internetes keresők (pl. Google, Yahoo) kezdtek kutatásba az új lehetőségek iránt, később a közösségi hálózatok elterjedésével az ezeket fejlesztő cégek is hasonló problémák okán indították el az RDBMS rendszerek cseréjének folyamatát. A kétezres évek közepére a folyamat az új típusú rendszerek publikálásával felgyorsult. A Google nyilvánosság elé tárta több belső szoftverének specifikációját, ezután pedig beindultak azok a projektek, melyek ezeket szerették volna lemásolni.

2009-re ezek a fejlesztések beértek, és hónapról hónapra jelentették be nagy cégek az adataik migrálását az új tárolókra. Általánosságban elmondható, hogy a fejlesztések ebben az évben érték el azt a stabilitást, amikor kritikus rendszereket lehetett rájuk építeni. Az egyre szaporodó cégek ezen projektek körül továbbgerjesztették a fejlesztési folyamatot. Érdemes megjegyezni, hogy az Apache Alapítvány nagy szerepet vállalt a szoftverek fejlesztési hátterének biztosítására. Több terméket is (Hadoop, Cassandra, Thrift stb.) kiemelt projektté nyilvánítottak, mellyel jelezték és egyúttal biztosították, hogy a későbbiekben sem fog leállni a fejlesztésük. [5]

5.1 Cap Tétel

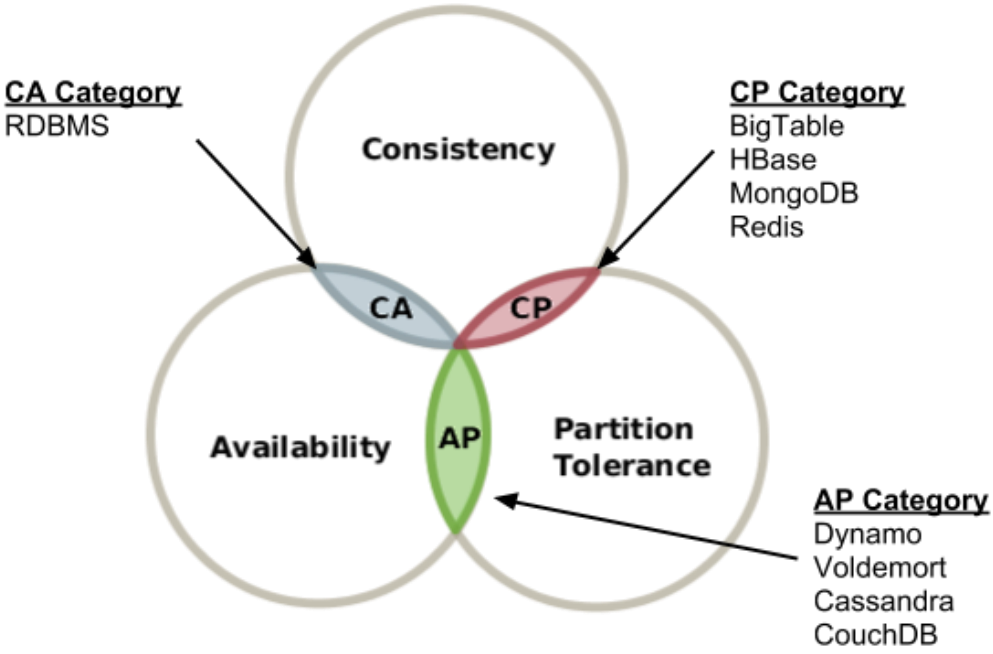
5.1.1 Eric Brewer CAP tétele

Térjünk át egy fontos elméleti pontjára az adatbázisoknak. A tételt először Eric Brewer fogalmazta, miután több skálázhatósági problémával találkozott különböző rendszereken. [9]

A tétel kimondja, hogy egy elosztott rendszer nem képes egyszerre teljesíteni a

- konzisztencia,
- hozzáférhetőség (availability) és a
- partíció tolerancia

által támasztott követelményeket.[4]



4. ábra: CAP tétel

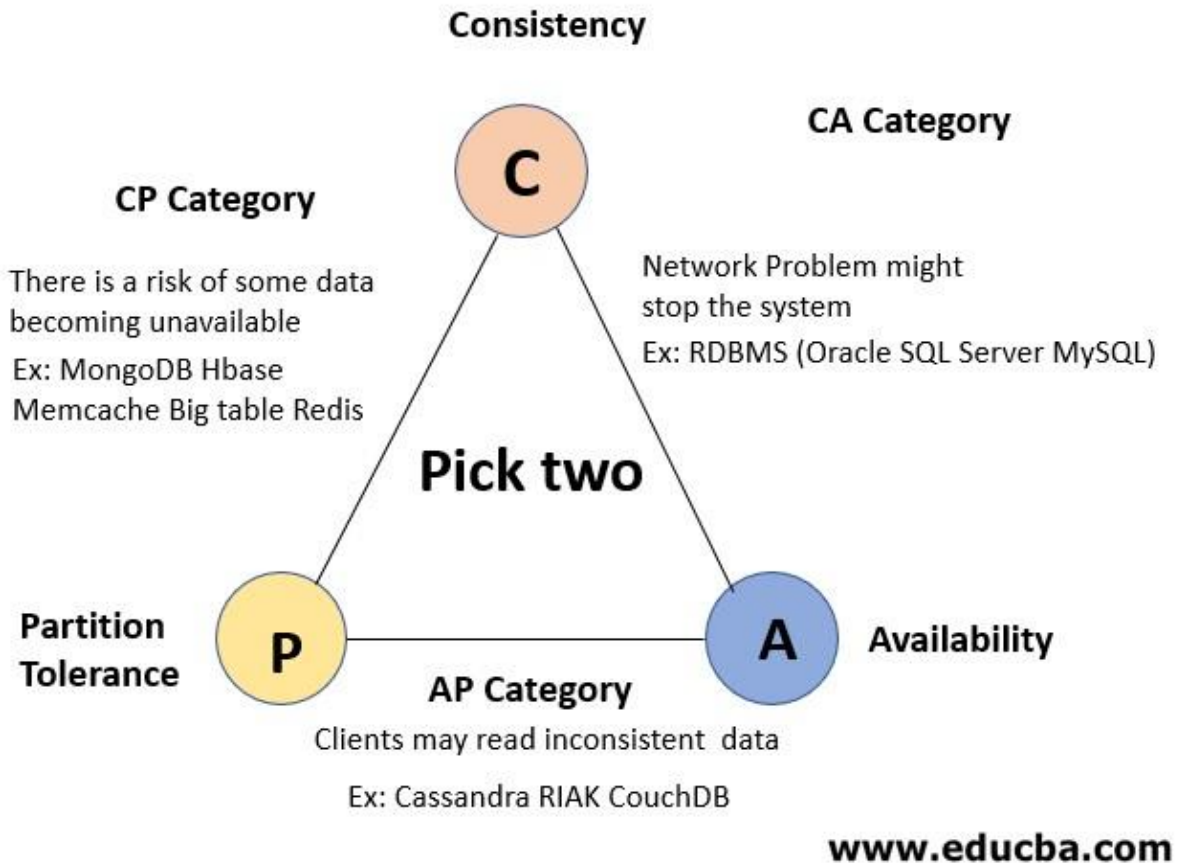
A konzisztencia fogalma az elosztott tárolók, mint rendszerek esetén azt jelenti, hogy a tároló egy változást egyik pillanatról a másikra hajt végre. Nem történhet olyan eset, hogy a rendszer egyes elemei különböző válaszokat szolgáltatnak a klienseknek. Példaként, adott egy 2 csomópontból (A, B) álló adatbázis hálózat, amelynek minden pontja elérhető a kliens számára. A kliensek bármely csomóponton keresztül módosíthatják és olvashatják az adatbázist. Ha A csomópontban változás történik egy adatsoron, annak a változás pillanatában meg kell jelennie a B csomóponton is. Tehát az adatbázis állapotváltozásai csak egyszerre történhetnek a csomópontokon, ezért a csomópontoknak szinkron működésűnek kell lenniük. A szinkronizációban több helyen is előfordulhatnak problémák, ezeket az adatbázis rendszerek más-más stratégiával védik ki. Csak akkor nevezhető egy adatbázis konzisztens rendszernek, ha bármely csomópontjából bármely időpontban ugyanazon információt kapja vissza a kliens. A konzisztencia különösen a kritikus rendszereknél fontos. Ezek például a bankok, repülőterek és minden olyan rendszer, ahol hibákat okozhat az inkonzisztens adat megjelenése. A NOSQL adatbázisoknál gyengébb elvárások jelentek meg a konzisztencia felé. Az ezeket az adatbázisokat használó rendszereknél gyakran nem okoz problémát, ha egy adatsor nem a legfrissebb változatát kapja vissza a kliens.

A hozzáférhetőség vagy magas rendelkezésre állás kérdése a tétel második sarokpontja. A rendelkezésre állás százalékban kifejezhető érték, amely megmutatja, hogy az adott rendszer X időtartamból mekkora Y időtartamban tudott válaszokat szolgáltatni, azaz volt elérhető. A legsebezhetőbb az egy csomópontot tartalmazó rendszer, mert az adatok 100%-a eltűnik a csomópont kiesése esetén. Több csomópont esetén a hozzáférhetőség mérése bonyolultabbá válik. Ezekben az esetekben elképzelhető, hogy a rendszer nagy része elérhető, de a benne található egyes csomópontok kiesése miatt a teljes adattartalmat még sem tudja elérni a kliens. [9]

A partíció tolerancia akkor merül fel egy rendszernél, amikor az azt alkotó csomópontok közötti hálózatban hiba keletkezik, vagy megszűnik a kapcsolat. Ekkor az adatbázis két különálló hálózatra szakad. A rendszer ekkor választás elé kerül: folytathatja a működését, vagy leáll. A két hálózat ilyenkor nem tud egymás működéséről, teljesen függetlenül változnak a kapcsolat megszakadása után. Amikor a hiba megszűnik, és a két hálózat újra egyesül, a rendszernek képesnek kell lennie az adatokat szinkronba hozni, hogy megszakítás nélkül folytatható legyen az adatbázis működése. [9]

A tételből következően egy tároló csak kettő tulajdonságot valósíthat meg a háromból. Ez alapján különböztetjük meg a tárolók 3 fő csoportját:

- **CA** (Consistency + Availability): A konzisztenciát és a hozzáférhetőséget tartják szem előtt ezek a rendszerek, a hálózat szétesése esetén működésképtelenné válnak. Eszerint működő rendszerek: MySQL, PostgreSQL, Vertica
- **AP** (Availability + Partition tolerance): Ezek a rendszerek a lehetőségekhez képest mindig elérhetőek, és jól működnek a hálózat szétesése esetén is, cserébe feladják a konzisztens működést. Ilyenek például: Cassandra, SimpleDB, CouchDB, Riak, Tokio Cabinet
- **CP** (Consistency + Partition tolarence): Konzisztens működés várható el ezektől a rendszerektől, és a hálózat elválása esetén is működnek, de cserébe romlik a hozzáférhetőség. A következő rendszereket lehet ide sorolni: Bigtable, HBase, MongoDB, Redis, Scalaris



5. ábra: CAP

5.2 Tárolók csoportosítása

Jelenleg 4 főbb csoportját lehet elkülöníteni a tárolóknak. Ezek főleg az adatstruktúra kezelésében és a CAP tételből eredően más-más kitűzött célban térnek el egymástól. Ez a felosztás azonban már egyre felületesebb, havonta egy-egy új kategória jelenik meg. [9]

5.2.1 Kulcs-érték tárolók

A kulcs/érték tárolók a legegyszerűbb tárolók a négy típusból. Ennél az adatmodellnél az adott információt tartalmazó adatrekeszt egy kulcs segítségével lehet elérni. A kulcsot a felhasználó maga választhatja meg, esetleg a tárolóra is bízhatja annak generálását.

A tipikus kulcs/érték tárolók string kulcsokkal és string értékekkel dolgoznak, melyeket byte tömbökké alakítanak. Az értékek általában binárisan biztonságosan (binary safe) tárolódnak, így lehetőség nyílik sorosított (szerializált) objektumok vagy bináris fájlok tárolására is.

Ezek a tárolók gyakran az adatok nagy részét – vagy egészét – a memóriában tartják, ezzel gyorsítják a kiszolgálást. A cache rendszerektől (pl. Memcached) azonban ezeket az különbözteti meg, hogy az adatokat valamilyen nem illékony tárolón is lementik annak érdekében, hogy egy esetleges nem várt esemény (pl. áramszünet) után is vissza tudják állítani azokat. [9]

A jelentősebb tárolók ebben a típusban: Amazon SimpleDB, Microsoft Azure Table Storage, Redis, Tokio Cabinet, Scalien Keyspace (magyar fejlesztés), Berkley DB, MemcacheDB. [9]

5.2.2 Dokumentumtárolók

Erre a típusra is jellemző, hogy az adatokat valamilyen hash alapján tárolja az adatbázis, de magának a letárolt adatnak van szerkezete, azaz az adatbázis képes az adatstruktúra alapján kereséseket és gyűjtéseket végezni. Tehát nem lehet csak egyszerűen egy továbbfejlesztett kulcs/érték tárolóként gondolni ezekre a rendszerekre. A fejlesztésük mozgatórugója az, hogy a fejlesztők gyakran az adatbázisokban dokumentumokat tárolnak, melyeket sokszor csak egy kulccsal (elsődleges kulcs) érnek el. Sőt a válaszidőt gyorsítandó eltérnek az adatbázis normalizálástól, és redundanciákat vezetnek be az adattárolási struktúrában. Ezeket a rendszereket valószínűleg sokkal jobban ki tudja szolgálni egy dokumentum tároló.

A dokumentum tárolók legnépszerűbb adatstruktúra leíró nyelve a JSON, ami egy ember számára értelmezhető, a számítógépeknek pedig könnyen feldolgozható struktúrát biztosít.

A jelentősebb dokumentum tárolók: CouchDB, MongoDB [9]

5.2.3 Oszloptárolók

A Google 2006-ban publikálta a legtöbb szolgáltatásához használt Bigtable adattároló specifikációját. Ebben kellő részletességgel leírták, hogy milyen módon valósították meg a hatalmasra nőtt adatbázisainak tárolását. Több fejlesztés is megindult, hogy a specifikációban leírt tárolóhoz hasonló alkalmazásokat dolgozzanak ki. Ezek közül számos nyílt forráskódú, elérhető mindenki számára.

Az RDBMS rendszerek egy táblában a sorokat egymás után tárolják. Ha csak egy mezőre van szüksége a kliens alkalmazásnak, először a sort keresik ki, majd a sorban a megfelelő oszlopot.

Ettől eltérően az oszlopokat szétbontva is lehet szervezni a tároló rendszert. Ez az alapja az oszloptároló rendszereknek.

Gyakori igény az adatok különböző verzióinak megőrzése. Ezt általában ezek a rendszerek egy új dimenzió bevezetésével, a verzióval valósítják meg.

Példa implementációk: Hadoop, Cassandra. [9]

5.2.4 Gráftárolók

A kétezres években megjelenő „web 2” új kihívások elé állította azokat a fejlesztőknek, akik szociális hálókkal vagy webkettes portálokkal foglalkoztak. Hatalmas és mély hálózatokat, gráfokat kell egy-egy portálnak kezelnie, és azokból kimutatásokat készítenie. Elég, ha a közösségi portálok egyik alap funkciójára, a közös ismerősök megtalálására gondolunk. Erre az RDBMS adatbázisok igen gyenge megoldást kínálnak, mivel sokszor többszörös JOIN-okkal kell egy-egy információt kinyerni. Természetesnek tűnik tehát, hogy valamilyen más módszerrel tárolják le ezeket az adatokat. Erre a gondolatra épülve fejlődtek ki a gráf tárolók, melyek a gráfok feldolgozása során felmerülő nehézségeket szeretnék megkönnyíteni.

A közösségi hálók után a második felhasználási területe az ilyen típusú tárolóknak a szemantikus web. A W3C az RDF ajánlással kíván egy olyan nyelvet a felhasználók kezébe adni, amivel az egyszerű adatokból strukturált információvá alakítható a web. Az adatobjektumok közötti hivatkozások tárolásához az egyik legjobb lehetőség a gráf tárolók alkalmazása. [9]

Jelentősebb tárolók a kategóriában: Neo4J, InfoGrid

5.3 Adatbázis technológia kiválasztása

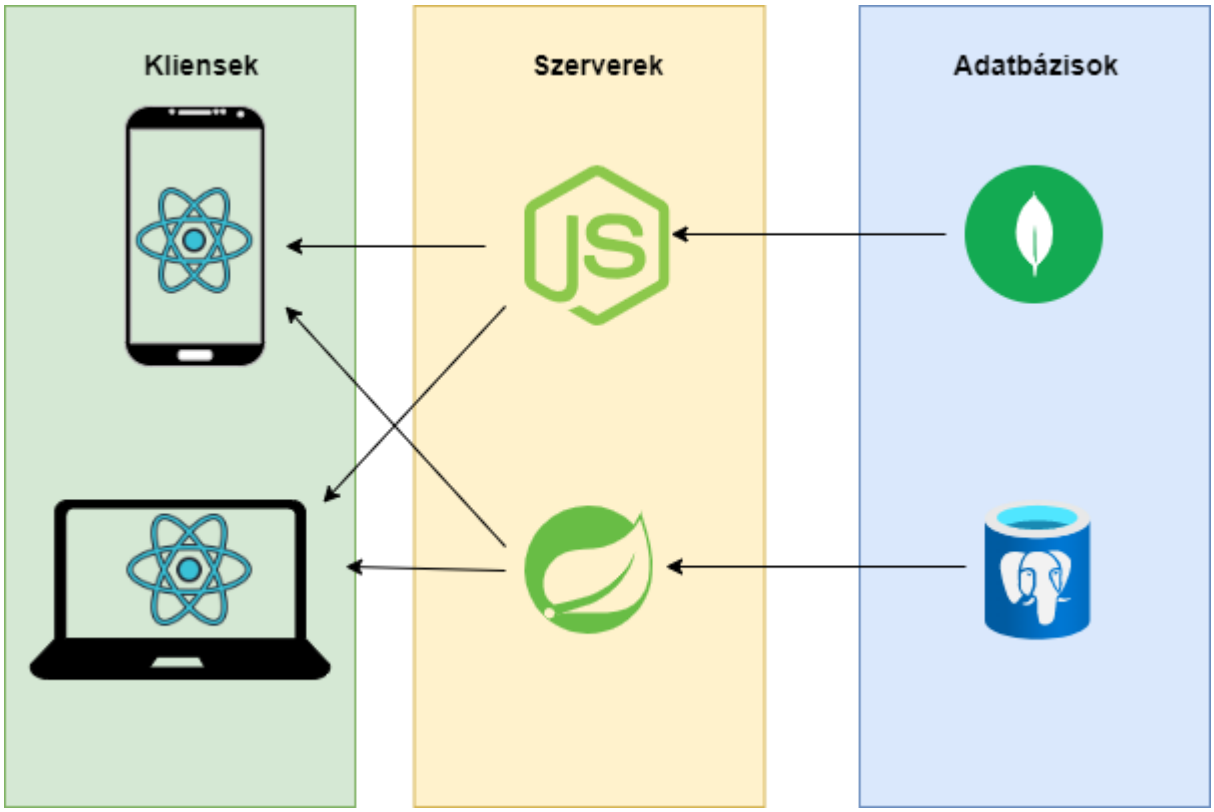
Az előbb felsorolt adatbázisok közül az SQL oldalon, mint az OLAP és OLTP résznél a PostgresSQL-t választom, mert a Java-val nagyon szépen tud működni és manapság egy nagyon elterjedt adatbázis a NoSql oldalt meg a Mongo-ra esett a választásom, mert lényegében JSON-okat akarok tárolni a rendeléseknél és ez a legmegfelelőbb adatbázis szerintem ehhez.

6. Követelmény specifikáció

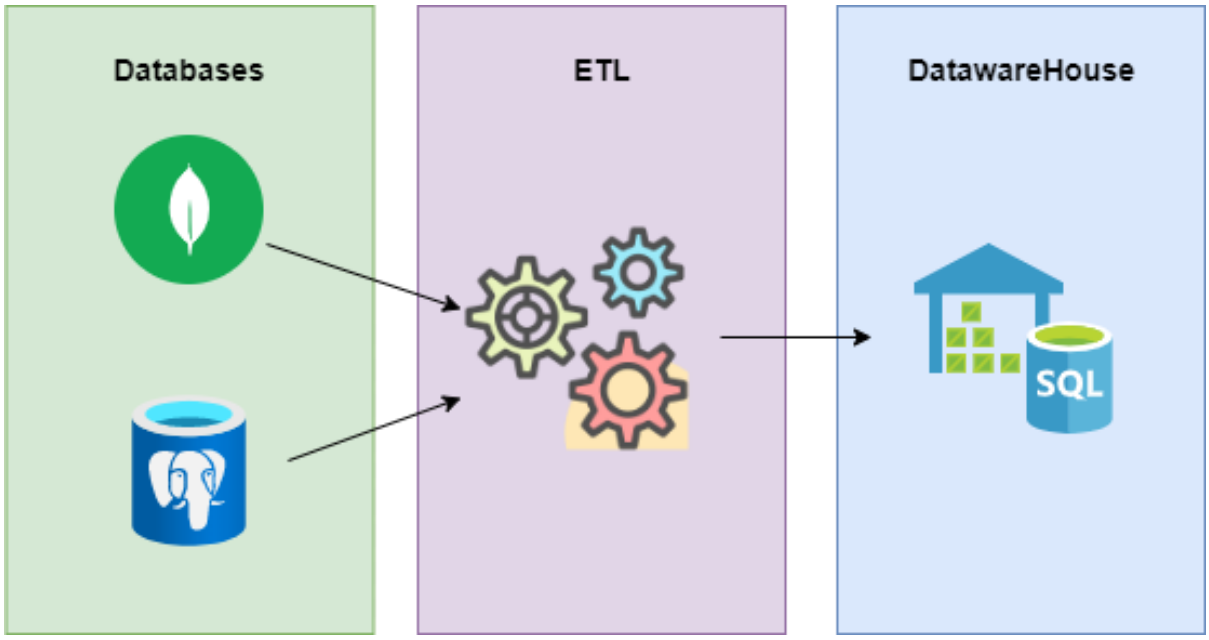
6.1 Általános követelmény specifikáció

A szakdolgozatomat lényegében 8 részre bontanám.

- Mongo adatbázis
- Postgres adatbázis
- Node.js backend
- Spring backend
- Applikáció
- Webes felület
- ETL pipeline
- Adattárház



6. ábra: Front end Back end terv



7. ábra: Back end to DatawareHouse

6.1.2 Mongo adatbázis

Az alkalmazásnak egy dokumentum tároló adatbázist is használnia kell erre a MongoDB-t fogom használni, mert a megrendeléseket 1-1 „dokumentumnak tekintem” és mivel a Mongo nagyon rugalmasan tudja kezelni az adott dokumentum sémáját, így tökéletes számomra, mert egy rendelésnél változhatnak a rendelt termékek száma. Ezen felül remekül skálázható és nagyon könnyű adathozzáférést biztosít. A szakdolgozat keretein belül az adatbázis feladata a rendelések tárolása és a kiszolgálása a kliensek felé.

6.1.3 Postgres adatbázis

Egy relációs adatbázisra is szükségem lesz a fejlesztés folyamán és én erre a postgresSQL-t választottam mert a Java backenddel nagyon jól kommunikál és véleményem szerint most a piacon az egyik legjobb adatbázis, amit lehet használni. A szakdolgozat keretein belül ezt az adatbázis feladata többek között a készlet tárolása, szerkesztése és az adatok kiszolgálása a kliensek felé.

6.1.4 Node.js backend

A Node.js backend szerver feladata a rendelések kezelése és az applikációtól fogadni a rendeléseket és menteni a NoSQL adatbázisba ezen felül, hogy a webes szoftvernek biztosítsa az rendeléseket ezen felül, hogy az összes szükséges CRUD és NONCRUD műveletet képes legyen kezelni. Ezen felül képesnek kell lennie, mint Get mint POST kérések kiszolgálására is.

6.1.5 Spring backend

A Spring backend szerver feladata a vendéglátóhelyeken lévő készlet kezelése az autentikáció megoldása mind az applikáció, mint a szoftveres irányban authoráció megoldása, hogy csak bejelentkezés után lehessen elérni adott site-okat ezen felül, hogy az összes szükséges CRUD és NONCRUD műveletet képes legyen kezelni. Ezen felül képesnek kell lennie, mint Get mint POST kérések kiszolgálására is.

6.1.6 Alkalmazás

Az Alkalmazás feladata az adatok lekérése mind a kettő szerverről REST-en keresztül és ezeknek a megjelenítése reszponzívan. Ezen kívül az alkalmazás feladata, hogy az aktuális pultos belépését biztosítsa. Ezt React-tel fogom megoldani, mert nagyon egyszerű a szintaktikája, képes vagyok mind a HTML, CSS, JS-t egy helyen kezelni, rengetek küldő könyvtárt támogat, így nagyon széles palettából tudok választani és ezeken felül nagyon nagy a támogatottsága olyan cégek által, mint a FaceBook, Netflix, AirBnb stb.

6.1.7 Applikáció

Az Applikáció feladata, hogy Rest-en keresztül képes legyen a szerverekkel kommunikálni. A felhasználónak biztosítani a kell egy beléptető felületet térképet, ahol láthatja, hol vannak a helyek, ahova betud ülni jelezve, hogy mely helyek vannak nyitva. Mind Android mind IOS eszközökre optimalizát megjelenítést. Az applikációhoz a React Native-ot választottam mert JavaScript-ben kell benne írni a kódot és a későbbiekben fogja lefordítani a megfelelő platformra a fordító. Erre a Framework-re is ugyan azok igazak, mint az előbb említett nyelvre mivel ugyan az ez, mint a React csak ezt Applikációk készítésére találtak ki.

6.1.8 ETL

Az ETL eszköznek annyi dolga lesz, hogy a 2 adatbázisból átjuttatja az adatokat az adattárházba.

6.1.9 Adattárház

Az adattárház feladata lesz, hogy a 2 adatbázisból megkapott adatokat elmentse hisztorikusan, hogy a későbbiekben képesek legyünk erre BI eszközök segítségével, minden féle információt kinyerni. Az adattárházra is Postgres adatbázist fogok használni, ezt OLAP berendezkedéssel nem úgy, mint a fent említett Postgres adatbázist, amit OLTP berendezkedéssel fog készülni

6.2 Funkcionális követelmény specifikáció

Az én esetemben regisztrációhoz van kötve minden funkció, mivel az applikációnál fontos, hogy ki rendelt a szoftvernél még azért fontos, hogy a pultosók betudjanak lépni mert webes felület és Api-kon fog kommunikálni a szerverekkel, így az adatok elérése szempontjából fontos, hogy csak olyan érje el azokat, akik bevannak lépve és másfelöl azért is fontos, mert a szoftver logol, hogy mikor ki lépet be és így nyilván lehet tartani mikor ki dolgozott a helyen.

6.3 Regisztrációhoz kötött funkciók

A regisztrációhoz kötött funkciók csak bejelentkezés vagy regisztráció után elérhető, mint az Applikáció, mint a Szoftver esetén.

Az applikáción ezek a funkciók válnak elérhetővé a felhasználó belépése után:

- Fiókunk kezelése
- Helyek böngészése a térképen
- Helyek keresése kereső sávban
- Adott helyen lévő ételek/italok böngészése
- Ételek/Italok rendelésbe helyezése
- Rendelés ellenőrzése

- Rendelés leadása

A szoftveren ezek a funkciók válnak elérhetővé a felhasználó belépése után:

- Bejövő rendelések kezelése
- Rendelések státusza készé alakítása
- Napi rendelések megtekintése
- Napi bevétel/fogyás megtekintése
- Készlet kezelése
    - felvétel
    - szerkesztés
    - törlés

7. Rendszerterv

A szakdolgozatomban az elkészítendő alkalmazás és applikáció, egy olyan szoftvercsoport, amely képes mind a vendéglátóhelyek mind a vendégek részére, ezeket a szoftverek 2 típusú backend fog kiszolgálni, amely mind NoSQl és SQL-es adatbázisra épülnek.

Az alkalmazás 8 részből áll.

- Mongo adatbázis
- Postgres adatbázis
- Node.js backend
- Spring backend
- Applikáció
- Webes felület
- ETL pipeline
- Adattárház

7.1 Mongo adatbázis

Az egyik adatbázis megvalósítása egy dokumentumtár típusú NoSQL adatbázis. Az adatbázis nem lesz teljesen strukturált, de a feldolgozott adatoknak meghatározott struktúrát határozok meg.

7.2 Postgress adatbázis

A másik adatbázis egy SQL típusú adatbázis. Az adatbázis egy előre elvárt struktúrát fog elvárni, amelyet majd Constraintek segítségével, fog betartatni.

7.3 ETL pipeline

Az ETL eszköznek nem lesz más dolga mint, hogy a 2 adatbázisból kinyert adatokat közös formátumra hozza és ezt betöltse az adattárházba.

7.4 Adattárház

Az adattárháznak csak annyi dolga van, hogy hisztorikusan tárolja a 2 adatbázisból kinyert adatokat.

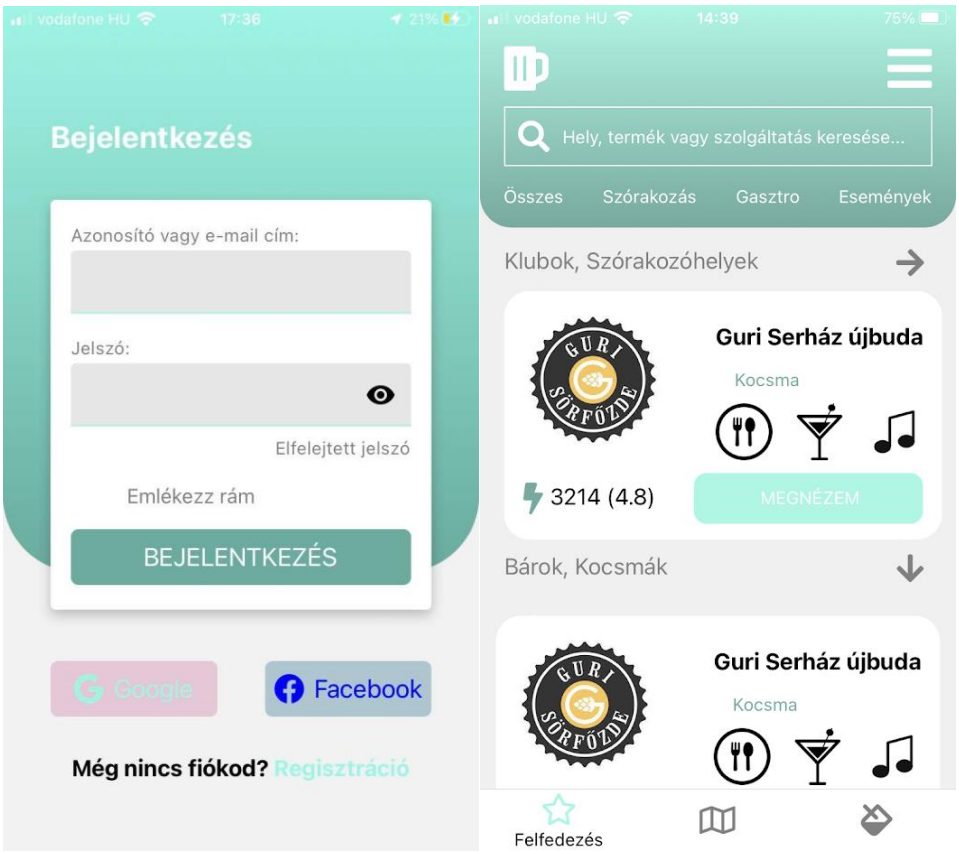
7.5 A tranzakciókat tároló dokumentum

A tranzakciót tároló dokumentum tárolja a vendég nevét a vendég email címét a helyet és hogy mit rendeltünk, ami egy Termék típusú tömb lesz ezen, kívül a rendelés dátumát és a státuszát, ahol nyomon tudjuk követni, hogy a rendelés elkészült e már vagy nem.

8. ábra: JSON strúktura

7.6 Applikáció

Az applikációt React Native-ban fogom megvalósítani. Az applikáció Rest api-n keresztül fog kommunikálni a szerverrel. A fent említett React Native technológia JavaScriptben íródott, így minden eszközt tudok használni a fejlesztés folyamatában, amit weben is képes vagyok használni, sőt még a native elemeket is képes maga a framework megszólítani mint például a kamerát, navigációt stb.



9. ábra: Applikáció Wireframe

7.7 Alkalmazás

Az alkalmazást React-ben szeretném megvalósítani. Az alkalmazás is Rest api-n keresztül fog kommunikálni a szerverrel. A fent említett React technológiával fogom megvalósítani, mert kimondottan nagy szabadságot ad, hogy milyen eszközöket szeretnél használni a fejlesztés folyamatában és tökéletesen lehet vele megoldani, hogy adott komponenseket többször is feltudjál használni, így jóval csökken a kód szám és sokkal átláthatóbb lesz, így az egész kódod és mivel lightweight a React sokkal gyorsabban fordul, mint a konkurens frameworkok mint például az Angular.

10. ábra: Alkalmazás Wireframe

8. Fejlesztés

A szoftver fejlesztési folyamata a legvégéig párhuzamosan haladt, mivel a Front-endeket teszt adatokkal teszteltem és fejlesztettem és a Back-endek a végpontjait pedig Rest kérésekkel teszteltem, hogy megbizonyosodjak a működésről és csupán a végén kötöttem rá a felületeket a Back-endekre, hogy adatot biztosítson nekik. Ezzel a módszerrel azt értem el, hogy nem volt függés és esetleges elakadás esetén is képes voltam haladni.

8.1. Front-endek megvalósítása

Első lépésként magát a webes felületet valósítottam meg, oly módon, hogy megcsináltam a mappa struktúrát, hogy ellegyenek szeparálva maguk az oldalak az újra felhasználható komponensektől. Amint elkészült maga a mappa struktúra minden szükséges könyvtárat beimportáltam a projektbe, mint például a navigációért felelős csomagot. Amint előkészült minden szükséges alapbeállítás neki álltam a tényleges fejlesztésnek. Kialakítottam az oldalak navigációját és elkészítettem a navigációs bárt.

```
<div className={hover ? "Navbar__ContainerBig" : "Navbar__Container"}>
            <span className="Navbar__LinkContainer"><FontAwesomeIcon icon={faBeerMugEmpty}
                                                    className={hover ? 'Navbar__logo Navbar__logohover' : 'Navbar__logo'}
                                                    /><span
                 className={hover ? 'Navbar__MenuTextNone Navbar__MenuText' : 'Navbar__MenuTextNone'}>BUP</span></span>
    <FontAwesomeIcon icon={!hover ? faArrowRightLong : faArrowLeftLong} onClick={() => sethover(!hover)}
                     className={hover ? "Navbar__arrow Navbar__arrowPadding" : "Navbar__arrow"}/>

    <NavLink to="/orders" style={({isActive : boolean }) => isActive ? activeStyle : notactivestyle}><span
        className="Navbar__LinkContainer"><FontAwesomeIcon icon={faCartShopping}
                                                    className={checker( url: "/orders") && hover
                                                        ? 'Navbar__Icons Navbar__ActiveIconPadding'
                                                        : !checker( url: "/orders") &&  hover ? 'Navbar__Icons Navbar__IconPadding'
                                                            : 'Navbar__Icons'}/><span
        className={hover ? 'Navbar__MenuTextNone Navbar__MenuText' : 'Navbar__MenuTextNone'}>Orders</span></span></NavLink>
    <NavLink to="/stock" style={({isActive : boolean }) => isActive ? activeStyle : notactivestyle}><span
        className="Navbar__LinkContainer"><FontAwesomeIcon icon={faBeer}
                                                    className={checker( url: "/stock") && hover
                                                        ? 'Navbar__Icons Navbar__ActiveIconPadding'
                                                        : !checker( url: "/stock") &&  hover ? 'Navbar__Icons Navbar__IconPadding'
                                                            : 'Navbar__Icons'}/><span
        className={hover ? 'Navbar__MenuTextNone Navbar__MenuText' : 'Navbar__MenuTextNone'}>Stock</span></span></NavLink>
    {sessionStorage.getItem( key: "position") === "1" && (
        <NavLink to="/charts" style={({isActive : boolean }) => isActive ? activeStyle : notactivestyle}><span
            className="Navbar__LinkContainer"><FontAwesomeIcon icon={faChartSimple}
                                                    className={checker( url: "/charts") && hover
                                                        ? 'Navbar__Icons Navbar__ActiveIconPadding'
                                                        : !checker( url: "/charts") && hover ? 'Navbar__Icons Navbar__IconPadding'
                                                            : 'Navbar__Icons'}/><span
            className={hover ? 'Navbar__MenuTextNone Navbar__MenuText' : 'Navbar__MenuTextNone'}>Charts</span></span></NavLink>
    )}
```

11. ábra: Navigációs bar

A navigációs bár egy viszonylag hosszabb és bonyolultabb kód volt, mivel nem csak a navigációt kellett megoldanom, hanem magát az animációt is, hogy ne csúszanak el a formázások a 2 animációs állapot között és maga az aktív navigáció jelölésének a stílusa a navigációs báron.

Ezután az applikációra vetettem az összes figyelmemet, mivel nagyobb falatnak találtam magát az applikáció elkészítését.

Az applikációnál is hasonlóképpen jártam el, mint a webes felületnél elkészítettem a mappa struktúrát, a könyvtárak importálását és magát a navigációt. Az applikációnál először a regisztráció és bejelentkezés felületet készítettem el a fejlesztés korai fázisában még beégetett felhasználóval lehetett belépni, ami később adatbázisos authentikációval lett leváltva. A főmenük közül a térképet tartalmazó nézet készült el először, ahol a legnagyobb fejtörést a Zoom szint meghatározása volt, mivel a térképen lévő képek nem voltak reszponzívak, így kicsinyítésnél összecsúsztak a képek. Az előbb említett problémát egy internet-en talált képlettel oldottam meg, amely kettő bemeneti adatsegítségével megtudta határozni számban, hogy mennyire is nagyíthattunk bele a térképbe. (Math.log2(360 * ((„Képernyő mérette” / 256) / „longitudeDelta”)) + 1). A Térkép után elkészült az a nézet, ahol tudunk böngészni a helyek között és különféle kategóriák szerint tudjuk szűrni a találatokat is. Eleinte egy tömbben tárolt adatokból generálta ki a helyeket, amiket megjelenített, később ezt leváltotta az adatbázisban tárolt helyek adatai. Az adatok egy saját készítésű komponensként generálodnak ki és ezt a React Native beépített komponense generálja ki görgethető listaként.

```
<FlatList data={DATA}
          renderItem={(item : ListRenderItemInfo<any> ) ⇒
              <ExploreBox
                  image={item.item.image}
                  title={item.item.title}
                  description={item.item.description}
                  subImages={[food,drink,music]}
                  rate={item.item.rate}
                  rateNumber={item.item.rateNumber}
                  horizontal={false}
                  routeName={"PubScreen"}
                  id={item.item.id}
              />
          }
          keyExtractor={item ⇒ item.id}
          showsHorizontalScrollIndicator={false}
          ListHeaderComponent={<FlatListHeader Headertext={"Klubok, Szórakozóhelyek"}
                                               Bottontext={"Bárok, Kocsmák"}
                                                DATA={DATA}/>}
          contentContainerStyle={{paddingBottom: 100}}
/>
```

12. ábra: Helyek generálása

Az adott kód részletben látható, hogy a FlatList komponenst használtam arra, hogy görgethető listaként generáltassam ki a saját komponensemet. A data paraméterben várja az adathalmazt. Ennek az adathalmaznak a részeit elérem a renderItem paraméterben, ahova beadtam a saját komponensemet, hogy azt generálja ki minden iterációban az átadott adatokkal. A keyExtractor csak egy id-t generál minden iterációban létrehozott komponensnek a többi attribútum pedig csak a megjelenésért felelős, mint például, hogy legyen-e az adott görgethető listámnak fejléce. Ahogy megfigyelhető a fejléc is egy sajátkomponens, ami egy görgethető lista, de az nem vertikálisan görgethető, mint a kódban látható, hanem az horizontális.

Amint elkészült a helyek listázására szolgáló nézet aktuális lett a helyeknek a saját nézetük elkészítése, ahol láthatók a termékek és ahol kosárba lehet rakni a termékeket.

```
const [load,setLoad] = useState( initialState: false)
const [list,setList] = useState<Array<prodInt>>( initialState: [])
const arr: Array<prodInt> = [];
const getProducts = async ()=> {
    await axios.get( url: "http://192.168.1.74:8080/api/getProductByManufacturer/"+id.toString()).then(res => {
        for (let i = 0; i < res.data.length; i++) {
            const data:prodInt = {
                id: res.data[i].id,
                title: res.data[i].name,
                description: res.data[i].description,
                price:res.data[i].brutto_price,
                size: 500
            }
            arr.push(data);
            setList(arr);
            setLoad( value: true);
            console.log(list)
        }
    }).catch(err => console.log(err))
}
```

13. ábra: Termékek lekérése

A kódrészletben az adott vendéglátóhely termékeinek lehívását láthatjuk és egy termék típusú tömb feltöltését. Amit külön meg kell jegyeznem, hogy TypeScript típusos nyelv a JavaScript-tel ellentétben, ezért is szükséges minden tömbnek, változónak típust létrehozni. Ami még megfigyelhető, hogy konkrét ip címre küldöm a Rest kérést és nem localhost-ra. Ez azért is van, mert a React Native csak konkrét ip címekre képes kérést küldeni, így megkell adni annak a gépnek a fizikális ip címét, amiről futtatjuk magát a Back-end-et.

A kosár olyan logika mentén készült el, ha már egy vendéglátóhely termékei a kosárban vannak nem lehet más helynek a termékeit berakni, de ha mégis ragaszkodunk hozzá akkor egy felugró ablak értesít róla, hogy az előző hely termékei kifognak kerülni a kosárból és az újonnan választott hely termékei fognak a helyükre kerülni.

```
const addBasket = (prodName:string) => {
    if(id != manufacturerId ) {
        setVisible( value: true);
        basket.length = 0;
    }
    let product= list.find( prod => prod.title.valueOf() == prodName);
    setManufacturerId(id);
    if (basket.some(prod => prod.Name.valueOf() == prodName)) {
        basket.find(prod => prod.Name.valueOf() == prodName)!.Count += 1;
    }
    else {
        basket.push({
            // @ts-ignore
            product_id:product.prodid,
            // @ts-ignore
            Name:product.title,
            // @ts-ignore
            Price:product.price,
            Count:1
        });
    }
}
```

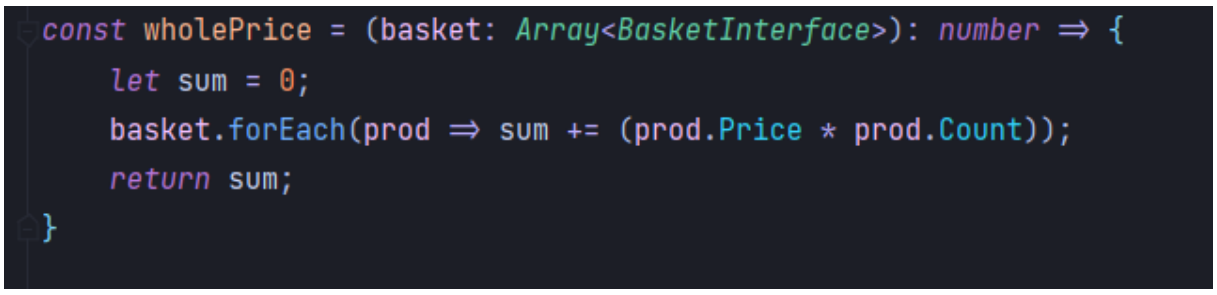
14. ábra: Kosár

A kosárhoz adásnak a kódját figyelhetjük meg ennél a résznél. Először megnézzük, hogy volt-e helyváltoztatás, ezután a kosárhoz pluszba mentett hely id-t összehasonlítjuk az aktuális helynek az id-ával, ha különbözik akkor ürítjük a kosarat és egy üzenetet küldünk, hogy töröltük a kosár tartalmát. Később a kosárba rakni kívánt terméket kikeressük a hely összes termékéből, mivel a terméket alapjáraton az összesből választottuk, így nem kell megnézni, hogy tartalmazza-e. Amint elkértük magát a terméket beállítjuk a helység id-ját és megnézzük, hogy van-e már ilyen termék a kosárban, ha van növeljük a darabszámát, ha viszont nincsen akkor hozzáadjuk a kosárhoz.

Amint ezek elkészültek elkészült a kosárnak a nézete, ahol megtekinthető, hogy mit is szeretnénk rendelni, ha rendben találjuk ez az oldal átnavigál minket az utolsó nézetre.

15. ábra: Kosár nézet kód

A kosár nézet úgy van megvalósítva, hogy ha üres a kosarunk akkor egy üzenetet láthatunk csak, hogy nincs termék a kosárban viszont, ha van létre hozza nekünk a táblázatos formát és kigenerálja nekünk a kosárban lévő termékeket táblázatos formában. A generálás előtt még egyszer megnézi, hogy biztos csak azokat a termékeket generálom-e ki, amik darabszáma nem 0. Erre azért van szükség, mert a kosár egy globális React változó és ritkán a kosárnak a nézete hamarabb ki generálódik, mint ahogy kiszedné a kosárból a terméket, aminek már 0 a darabszáma a kosárban. Az utolsó nézeten beállíthatjuk, hogy hány százalék borravalót szeretnénk adni, vagy éppen, hogy pontosan mekkora összeget. Amint ezeket kitöltöttük a rendelés gombra kattintva elküldésre kerül az adatbázisba a rendelés, ami a későbbiekben megfog jelenni a webes felületen.



16. ábra: Ár képzés

Ez a kód részlet a kosártartalmát összesíti, amit a későbbiekben fogunk megjeleníteni a felületen.

```
<View style={{paddingTop: 20, backgroundColor: "#fff"}}>
    <View style={style.sum}>
        <Text style={{fontSize: 18, color: "grey"}}>Borravaló összege</Text>
        <Text style={{fontSize: 18, color: "#5EFC8D"}}>{(wholePrice(basket) / 100) * tipPercent} Ft
            ({tipPercent}%)</Text>
    </View>
    <View style={style.sum}>
        <Text style={{fontSize: 18, color: "grey"}}>Fogyasztás összege </Text>
        <Text style={{fontSize: 18, color: "#5EFC8D"}}>{wholePrice(basket)} Ft </Text>
    </View>
    <View style={[style.sum,{paddingBottom:15}]}>
        <Text style={{fontSize: 18, color: "grey"}}>A teljes összeg</Text>
        <Text style={{
            fontSize: 18,
            color: "#5EFC8D"
        }}>{((wholePrice(basket) / 100) * tipPercent) + wholePrice(basket)} Ft </Text>
    </View>
```

17. ábra: Összesítő felület

Itt láthatjuk, hogy hogyan íratja ki a végén a kosár értékét, a borravalót és összesítve a kettőt. A wholePrice függvény visszatér a kosár tartalmával és ezt osszuk majd 100-zal és felszorozzuk a kívánt borravalóval, hogy megkapjuk mennyi a borravaló az adott rendelésünkben.

Az applikáció befejeztével a webes felülettel folytattam, mivel a hozzá tartozó back-end már olyan szinten elkészült, hogy pár lényeges funkciót, mint például az authentikációt már rátudtam építeni. A webes felületen az első nézet maga volt a bejelentkezési felület. A bejelentkezési felülettel párhuzamosan nem készült regisztrációs felület, mivel egy ott dolgozó személy megkapja a belépéséhez szükséges jelszót, felhasználónevet, így ilyen tekintetben szükségtelen a regisztráció.

18. ábra: Login kód

A kód részletben azt figyelhetjük meg, hogy elküldjük a Back-endnek az email-t és a jelszót és a Back-end pedig visszatér egy true-val vagy egy false-sal, attól függően, hogy van-e ilyen User az adatbázisban, ha van akkor még a jogkörével is visszatér, hogy milyen pozíciót lát el az adott cégnél. A sessionStorage-et azért használom, hogy oldal újra töltésnél is belegyek jelentkezve a felületre. A MySwal pedig egy változó, amit a Swal osztályból példányosítok. A Swal a SweetAlert nevezetű könyvtárból jön, ez egy olyan könyvtár, ami stílusos és könnyen szerkeszthető pop-up ablakokat biztosít számunkra. A bejelentkezési felület utána rendelések kezelésére szolgáló felület készült el, amit drag and drop-os megoldással valósítottam meg. Mikor belépünk a rendszerben az összes rendelést először a Todo oszlopban látjuk és ezeket vagyunk képesek mozgatni, oly módon, hogy az adott rendelés már készülődő fázisban van vagy éppen már el is készült. Az itt felsorakozó rendeléseknél egy timer-t is megfigyelhetünk, ami számolja, hogy az adott rendelésre már mióta is várnak.

```
let ord: Array<orderInterface> = [];
const getOrders = async () => {
    await axios.get( url: "http://localhost:4000/getOrders").then(res => {
            for (let i = 0; i < res.data.length; i++) {
                let rentArray: Array<rentsInterface> = [];
                for (let j = 0; j < res.data[i].Rents.length; j++) {
                    const rent: rentsInterface = {
                        name: res.data[i].Rents[j].Name,
                        count: res.data[i].Rents[j].Count
                    }
                    rentArray.push(rent)
                }
                const order: orderInterface = {
                    id: res.data[i]._id,
                    rents: rentArray
                }
                ord.push(order)
            }
            setLoad( value: true);
        }
    ).catch(err => console.log(err))
}
```

19. ábra: Rendelések betöltése

Ezen a kód részleten magát a rendelések betöltését figyelhetjük meg, ami bonyolultabb volt ebben, mint a többi betöltésnél, hogy nem csak a rendeléseket, hanem a rendelések tartalmát is bekellett tölteni és ezt egy dupla ciklussal tudtam megoldani. Ezen kívül egy load változott is bevezettem, hogy csak a feltöltés után legyen betöltődve maga az oldal.

```
interface rentsInterface {
    name: string,
    count: number
}

interface orderInterface {
    id: string,
    rents: Array<rentsInterface>
}
```

20. ábra: Rendelések betöltése típus

Ezen a képen a fent látható dupla ciklusos betöltésnek a típus megadását láthatjuk, ami furának tűnhet, hogy interface-ket hozzuk létre, hogy típust adjunk meg, viszont a TypeScript-ben interface-ként kell definiálva a típusokat. A rendelések kezelésére szolgáló nézett után a készlet nyilvántartására szolgáló nézetett készítettem el, ennek a nézetnek a sajátossága, hogy ha sima pultos jogkörrel rendelkező profillal lépünk be akkor csak megtekinteni tudjuk a készletet, de ha admin jogkörrel rendelkező profillal lépünk be, akkor képesek vagyunk a termékeket törölni, módosítani vagy épen újat hozzá adni. A legvégén meg egy kimutatásokat megjelenítő nézet volt, ami az adattárházból kiszedett adatokból vizualizálja nekünk például a havi fogyást termékekre lebontja vagy éppen a napi aktuális bevételt.

8.2. Bank-endek megvalósítása

Első lépésként egy docker fájlt hoztam létre az alkalmazásban, aminek a lefuttatása után elindult nekem a kettő adatbázisom. Második lépésként az adatbázist hoztam létre, ami tartalmazza a termékeket és minden fontos információt a vendéglátóhely számára. Az adatbázist nem a szokványos SQL kódokkal hoztam létre, hanem egy külön könyvtárat használtam név szerint a liquibase-t. Ennek a könyvtárnak a sajátossága, hogy ha egy alkalmazás rávan csatlakoztatva egy adatbázisra a mi esetünkben a Postgress adatbázisra akkor a liquibase a szoftverben megírt tábla leírást futtatásnál létrehozza az adatbázisban. [6] Ez a tábla leírás xml formátumú, de képesek vagyunk SQL kódot is írni a megfelelő tag közzé. Ennek a könyvtárnak az előnye a verzió kezelés mivel látjuk mikor milyen változtatást eszközöltünk az adatbázisban.

```
</changeSet>
<changeSet author="adam.petrik" id="1661886399260-3">
    <createTable tableName="employee">
        <column autoIncrement="true" name="employee_id" type="BIGINT">
            <constraints nullable="false" primaryKey="true" primaryKeyName="employee_pkey"/>
        </column>
        <column name="first_name" type="VARCHAR(100)"/>
        <column name="hire_date" type="TIMESTAMP WITHOUT TIME ZONE"/>
        <column name="last_name" type="VARCHAR(100)"/>
        <column name="mail" type="VARCHAR(50)"/>
        <column name="phone" type="VARCHAR(11)"/>
        <column name="manufacturer_id" type="BIGINT"/>
        <column name="position_id" type="BIGINT"/>
        <column name="password" type="VARCHAR(50)"/>
    </createTable>
</changeSet>
```

21. ábra: liquibase xml

Ezen a képen egy changeSet-et láthatunk, ami a liquibase-es xml fájlban van. Egy changeSet egy darab módosítást reprezentál az adatbázisban. Egy changeSet-nek 2 attribútuma van az, hogy ki készítette és mi ennek az azonosítója. A changeSet-en belül láthatjuk magát a táblának a létrehozását és hogy milyen oszlopai lesznek a táblának milyen típussal. Ami lényeges, hogy ha van bármi constraint-je az oszlopnak akkor a Tag-ek közzé kell írni az adott megszorítást. Amint elkészültek a táblák és a táblák közötti kapcsolatok elkészítettem a tábláknak a modelljeit ezeket a modelleket automatikusan a Spring összeköti az adatbázisban megfeleltethető táblával és csupán annyira van szüksége az összekötéshez, hogy a modellnek és az adatbázisban lévő táblának azonos nevei legyenek és a modell adattagjai is megfeleltethetők legyenek az adott tábla attribútumainak. Amint a modellek elkészültek repository-kat hoztam létre az összes modellnek ehhez a CrudRepository könyvtárat használtam. A CrudRepository könyvtárnak van egy olyan sajátossága, hogy ha megadunk neki egy modell-t és az adott modellnek az id-ját akkor abból képes az összes CRUD műveletet legeneráltatni számunkra plusz megadja a lehetőséget arra is, hogy saját műveleteket generáltassunk magunknak, mint például egy felhasználónév és egy jelszó alapján visszaadja az adott rekordót az adatbázisból. A repository-k után a servicek készültek el, amik összekötötték az controllereket a repository-kkal és az üzleti logikát tartalmazták, mint például az adattárház feltöltését.

```java
@Override
public Set<Product> getProducts(Long id) {

    Set<Product> products = new HashSet<>();

    Optional<Manufacturer> manufacturer = manufacturerRepository.findById(id);
    manufacturer.ifPresent(value -> productRepository.getProductsByManufacturer(value).iterator().forEachRemaining(products::add));
    return products;
}
```

22. ábra: Service

A kód részletben egy egyszerűbb Service látható, ami elkéri vissza egy vendéglátóhely azonosítóját és visszaadja az összes terméket, ami kapható a vendéglátóhelyen. Ami lényeges a kód részletben azaz Optional generikus adattároló komponens a Java-ban, ennek a sajátossága, hogy megtudjuk nézni egy függvényével, hogy van e benne érték és ha igen eltudjuk kérni és tudunk vele dolgozni. A legvégén pedig a controllerek készültek el, amik biztosították a végpontokat a front-end-ek felé.

```java
@RestController
@RequiredArgsConstructor
public class AuthController {

    private final EmployeeRepository employeeRepository;

    @PostMapping(value = "/login",produces = {MediaType.APPLICATION_JSON_VALUE})
    @CrossOrigin(origins = "http://localhost:3000")
    public String auth(@RequestBody Login body) {

        Optional<Employee> emp = employeeRepository.findByMailAndAndPassword(body.mail,body.password);
        Employee employee;
        if (emp.isPresent()){
            employee = emp.get();
            return true+","+employee.getPosition().getId();
        }
        else {
            return false+","+0;
        }

    }
}
```

23. ábra: Controller

A fent látható kód részletben egy Controller-t láthatunk, ami fogadja a front-endről kapott email-t és jelszót ezután pedig a megnézi, hogy van e hozzátartozó User, ha van visszatér egy true-val és a pozíciójával, ha pedig nincs akkor false-sal tér vissza. Amiket fontos megemlíteni szerintem az a @RequiredArgConstructor, ami példányosít minden osztályt/interfacet amik még nem voltak példányosít és a @CrossOrigin-t, ami meghatározza, hogy honnan is fogad az adott végpont kéréseket.

A fent leírt Spring back-end-nél az adattárház kiépítése és az adattárházhoz tartozó lekérdezések elkészítése volt.

```java
@Component
public class CustomQuerys {

    @Autowired
    private JdbcTemplate jdbcTemplate;

     private  final String GET_PRODUCT_AND_SUM = "select name,sum(price)from order_fact ord inner join product_dim pd" +
         "    on ord.product_dim_id = pd.product_dim_id group by name";

    public List<ProductNameAndSum> productNamesAndSum(){
        List<Map<String, Object>> list = jdbcTemplate.queryForList(GET_PRODUCT_AND_SUM);
        var productandSum = new ArrayList<ProductNameAndSum>();
        for (int i = 0; i < list.size(); i++) {
           productandSum.add(new ProductNameAndSum(list.get(i).get("name").toString(),
               Integer.valueOf(String.valueOf(list.get(i).get("sum")))));
        }
        return productandSum;
    }
}
```

24. ábra: Adattárház lekérdezés

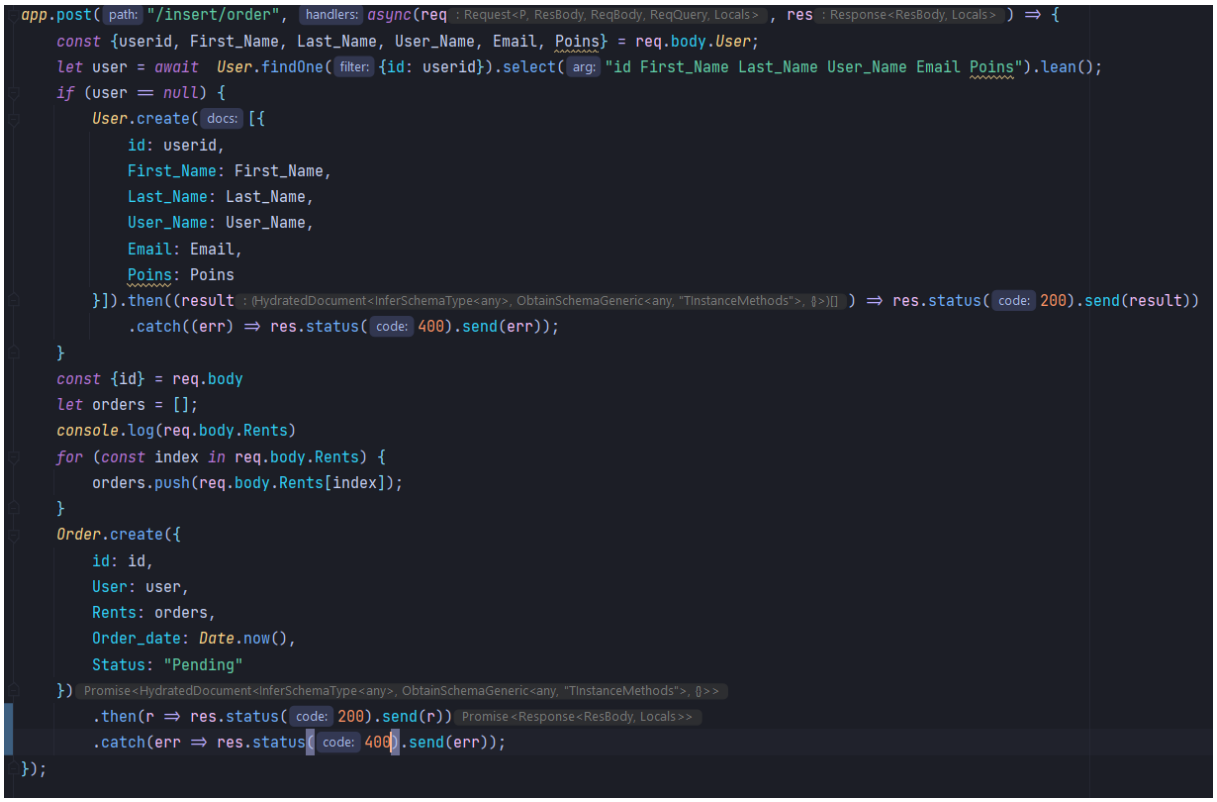
Az itt látható kódrészletben láthatjuk az adattárházhoz tartozó lekérdezést. Itt a jdbcTemplattel lekérdezést intézek közvetlenül az adattárházhoz és a megkapott értékeket egy saját készítésű osztály típusú listát feltöltők és térek vissza vele.

Az adattárház feltöltése azért okozott több fejtőrést is nekem, mert egy megfelelő sorrendiségben kellett feltölteni a táblákat, hogy ne fusson hibára a hozzá tartozó lekérdezések, meg azért voltak bonyolultabbak, mint a többi lekérdezés, mert itt nem lehetett repository-kat használni, hanem jdbcTemplattel közvetlenül az adatbázishoz kellett kéréseket intézni. A jdbcTemplate egy beépített könyvtár magában a Spring-ben segítségével képesek vagyunk sql lekérdezéseket intézni az adatbázis felé.

25. ábra: Dátum dimenzió generálás

Az itt látható kód részletben magát a dátum dimenzió generálását láthatjuk, ami fontos itt, hogy mindig a mai naptól kezdi el generálni az értékeket percenként és tölti fel magát a dimenzió táblát. Ez a metódus körülbelül egy hónap-ot tölt fel, így minden egyes betöltésnél megnézi, hogy az aktuális idő több-e, mint ami az adatbázisban van, ha igen akkor újra futtatja a metódust viszont már a saját idejével. Azért lett ez így megvalósítva, mert így sose lesz olyan, hogy nem lesz kapcsolható dátum a csillagsémában.

A vendéglátóhelyek számára szolgáló back-end befejeztével az applikációt kiszolgáló back-enddel folytattam ennek a back-endnek lényegesen kisebb feladatköre van, mint az előzőnek, mivel csak a rendeléseket kellett mozgatnia Mongo adatbázis és a végpontok között és a felhasználó adatait kellett tárolnia és ezek ismeretével a bejelentkezést lebonyolítania. Ennek a back-endnek a fejlesztését először az adatbázist összekötöttem a back enddel, ehhez a Mongoose nevezetű könyvtárat használtam. A Mongoose egy olyan könyvtár, aminek a segítségével képesek vagyunk csatlakozni egy Mongo adatbázishoz és tudunk lekérdezéseket is írni a könyvtár segítségével. Amint elkészült az adatbázis kapcsolat létrehoztam a schema-kat a Mongoose segítségével és megírtam a hozzájuk tartozó végpontokat. Először a regisztrációért és bejelentkezésért felelős végpontok készültek el azután azok a végpontok, amik fogadták az applikációból jövő rendeléseket és elmentették azokat az adatbázisba és ezt vissza is adták a webes felület felé.



26. ábra: Rendelés kezelés

A fent látható kód részlet tárolja el az applikációkból bejövő rendeléseket. Először megnézem, hogy van az adatbázisban olyan User aki rendelt, ha nem akkor létrehozom. Ezt biztonsági okból csinálom, hogy ha bármi ok miatt nem lenne elmentve a User az adatbázisba. Az utána látható kódrészletben, pedig egy tömbbe belerakom a rendeléseket és létrehozom magát a rendelést, úgy, hogy a státusza Pending lesz hiszen még csak most adta le a rendelés dátuma pedig az aktuális dátum lesz.

8.3. Összekötés

Utolsó lépésként, amint elkészült mind a Front- end mind a Back-end rész kitöröltem az összes teszt adatot tartalmazó tömböt a Front-end-ekről és rákötöttem őket a back-endek végpontjaira. Eleinte voltak kisebb fent akadások, mert nem mindig feltétlen olyan típusú adatokat kapott a Back-end, amit feltudott dolgozni, de ezt segéd osztályok segítségével viszonylag hamar orvosolva lett.

9. Tesztelés

A tesztelés elengedhetetlen része a fejlesztési folyamatnak. Ilyenkor nem csak a programozó győződik meg a specifikációban leírtak megvalósításának sikeréről, hanem a felhasználók is próbára tehetik a programot, hogy az az elvárásaiknak megfelelően működik-e, illetve a megrendelő szemszöge is fontos, hogy a program valóban a megrendelésben leírtakkal megegyező.

A tesztelést két részre bontottam. Első körben én teszteltem a szoftver különböző részeit, hogy minden része az elvárt funkcionalitást produkálja és mikor meggyőzöttem róla, hogy az elvárt módon működik a szoftver jött a második rész, mikor ismerősöket kértem meg, hogy laikusként teszteljék magát a szoftvert. Ha a tesztelésnél hiba üzenet kellettkezet, vagy csak szimplán nem töltődött be a kívánt tartalom a felületre, akkor rögtön elkezdtem a böngészőben nézni a console-t, hogy történt e hiba a tartalom betöltésében. Viszont a hiba maga a backend-en történt akkor debuggolással kutattam fel és javítottam a hibát. Ehhez a lehetséges összes esetet végig próbáltam, ezen esetek listája és kimenetelei a tesztelési táblázatban láthatók. A teszteléshez a saját számítógépemet használtam, amelyen a teljes fejlesztés történt.

Számítógép	**Személyre szabott desktop**
Processzor	Intel(R) Core(TM) i5-7500 CPU @ 3.40GHz
Memória	16 GB

2. táblázat: PC Hardware

A tesztelés szoftver környezete:

Operációs rendszer	**Microsoft Windows 10 Enterprise**
Keretrendszer	Java Spring 8.2, Node.js, Expo
Fejlesztőkörnyezetek	Web Storm, Intellij Idea Ultimate
Programozási nyelvek	Java, JavaScript, TypeScript
Alkalmazások típusai	React, React Native
Adatbázisok	Mongo, Postgress

3. táblázat: Fejlesztési környezet

Az alábbi követelmények, amelyek szerepelnek a specifikációban, tesztelésre kerültek, a teszteseteket és sikerességüket.

Bejelentkezés az Applikációba	Sikeres
Helyek betöltése a képernyőre	Sikeres
Kiválasztott helyről termékek böngészése	Sikeres
Termékek kosárba rakása és kiszedése	Sikeres
Kosár tartalmának összesítése	Sikeres
Rendelés leadása	Sikeres
Rendelés tárolása az adatbázisban	Sikeres
Bejelentkezés a webes alkalmazásba	Sikeres
Leadott rendelések megjelenítése	Sikeres
Rendelések kezelése	Sikeres
Termékek megjelenítése	Sikeres
Termékek módosítása	Sikeres
Termékek törlése	Sikeres
Termékek hozzáadása	Sikeres
Adattárház betöltése	Sikeres
Diagrammok megjelenítése	Sikeres

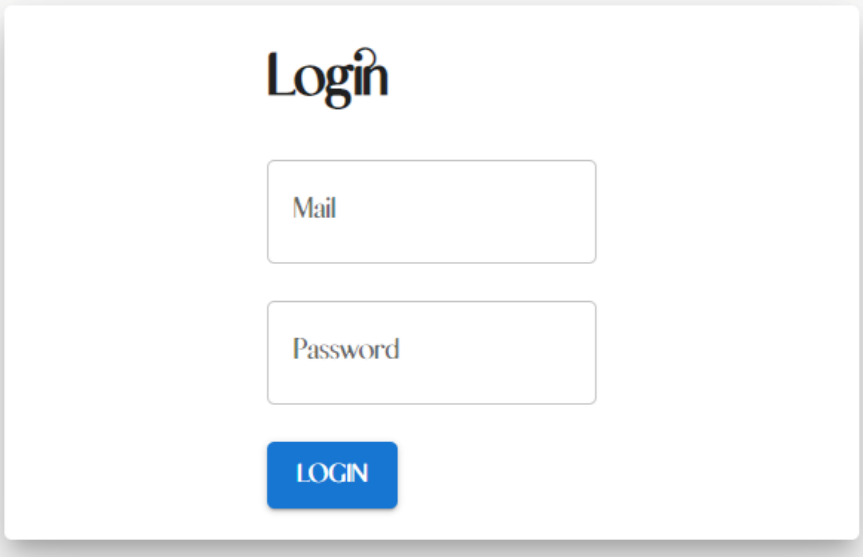
4. táblázat: Tesztesetek

Ezeket a tesztesetekkel próbáltam lefedni, azonban a használat során, főként az első időszakban felmerülhetnek újabb, eddig nem ismert hibák, emiatt is nagyon fontos az utógondozás. A felmerült új hibákat javítani kell, illetve azokat a megoldásokat, amik mégsem nyújtanak 100%-os megoldást, fejleszteni.

10. Felhasználói felületek

10.1. Webes felület

A felhasználói felületek bemutatását a vendéglátóhely oldaláról kezdeném a későbbiekben térnék ki az applikáció felületeinek bemutatására.



27. ábra: Vendéglátóhely login

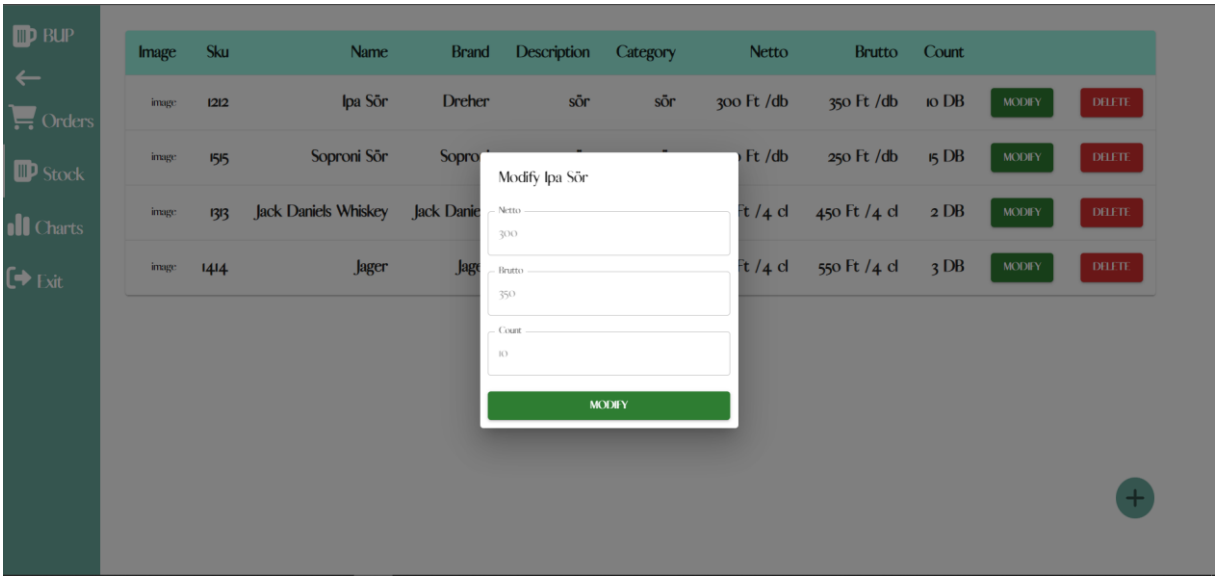
Ezen a felületen egy egyszerű bejelentkezési ablakot láthatunk, aminek a kitöltésével képesek vagyunk belépni a főoldalra. Az itt látható komponenseket, mint a gomb és mint a beviteli mezők egy UI könyvtárból importáltam be, aminek a neve Material UI. A Material UI a legelterjedtebb UI könyvtár a React-es alkalmazásoknál, több nagyobb cég is használja ezt a könyvtárat, mint például a Netflix vagy éppen a Spotify. [7]

28. ábra: Rendelések kezelése

Ezen a felületen láthatjuk magát a rendelések kezelésére szolgáló felületet és bal oldalt meg a navigációs bar-t aminek a segítségével képesek vagyunk navigálni az adott nézetek között. A rendeléseket, oly módon vagyunk képesek kezelni, hogy egy színes kis hasáb 1 db rendelést takar a háttérben és ennek a hasábnak a mozgatásával a különböző állapot oszlopok között tudjuk az adott rendelés státuszát. Ezen felül megfigyelhető a hasábok bal alsó sarkában a rendelés azonosítója és az, hogy hány perce kaptuk meg az adott rendelést.

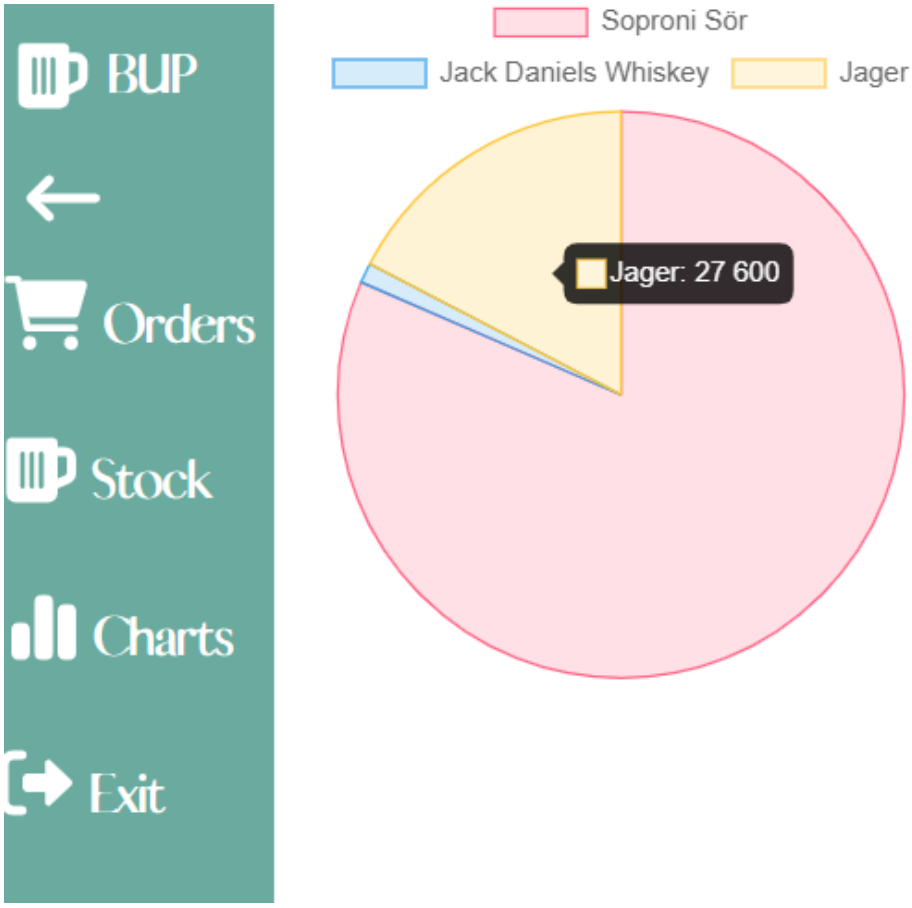
29. ábra: Készlet nyilvántartás

Ezen a felületen a készlet nyilvántartást láthatjuk a táblázat minden sora 1 db terméket reprezentál, ahol felvannak sorolva a termék bizonyos tulajdonságai a végén meg 2 darab gombot láthatunk, amiknek a segítségével tudjuk törölni vagy módosítani azt adott terméket. Ezen felül még azt vehetjük észre a képen, hogy az oldalsó navigációs bar már nem csak a oldalak logóit, hanem a nevüket is tartalmazza, viszont, ha nekünk jobban tetszik, hogy csak a logók látszódnak akkor a balra mutató nyílra kattintva visszahúzódik a fent látható képen látható formába.



30. ábra: Termék módosítása

Ezen a felületen azt figyelhetjük meg, hogy ha rányomunk egy termék módosítás gombjára, akkor felugrik egy ilyen kisebb ablak, ahova már a program beilleszti az alap információkat és ezzen információk módosításával tudjuk a termék értékeit megváltoztatni. A felugró ablak és a rajta látható gombok és beviteli mezők, mind a Material UI-ból jöttek, így egy ilyen összetettebb komponenst is viszonylag rövid idő alatt összelehet állítani.



31. ábra: Vizualizáció

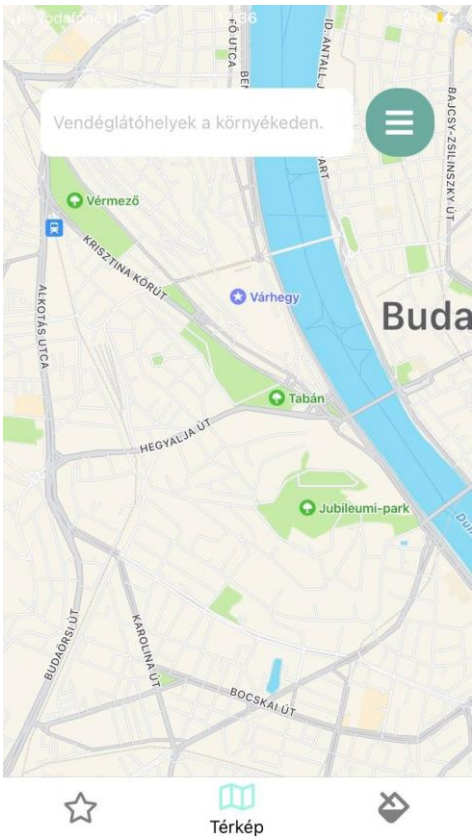
Ezen a felületen egy vizualizációt láthatunk, amit csak az admin jogkörrel rendelkező személyek érnek el. A vizualizációt az adattárházból kinyert adatokból készítettem. Ezen a diagramon azt figyelhetjük meg, hogy az adott hónapban bejövő bevétel, hogy oszlik el az adott termékek között. Ezt a diagramott a React egyik diagram készítő könyvtárával készítettem és azért is választottam ezt Power BI helyett, mert közel olyan szinten tudom az adatokat vizualizálni, mint a Power BI-jal viszont ezt beletudom ágyazni az adott rendszerbe. Végezetül, amit fontos kiemelni a végén, hogy csak az adott vendéglátóhelyhez kapcsolódó termék, rendelés és vizualizációs adat töltődik be és jelenik meg a felületen.

10.2. Applikáció felület

Az előző pontban bemutattam a webes felületet és most kitérnék magára az applikációra.

32. ábra: Applikáció login

Ezen a felületen látható maga a bejelentkezés. Az itt látható komponenseket is egy UI könyvtárból lettek beimportálva, aminek a neve React Native Paper. Azért a React Native Paper-t választottam és nem más UI könyvtárat, mert napra kész maga a könyvtár a felmerülő hibákat heti szinten javítják és ennek a könyvtárnak a legnagyobb a komponens palettája ennek meg az az elsődleges előnye, hogy egységes lesz a Design-ja az összes komponensnek, mert minden UI könyvtárnak meg van a sajátossága, ahogy a komponenseket formázza és ha több UI könyvtárból vagy kénytelen lefedni a szükséges komponenseidet, akkor lesznek bizonyos eltérések adott komponens formázások között.

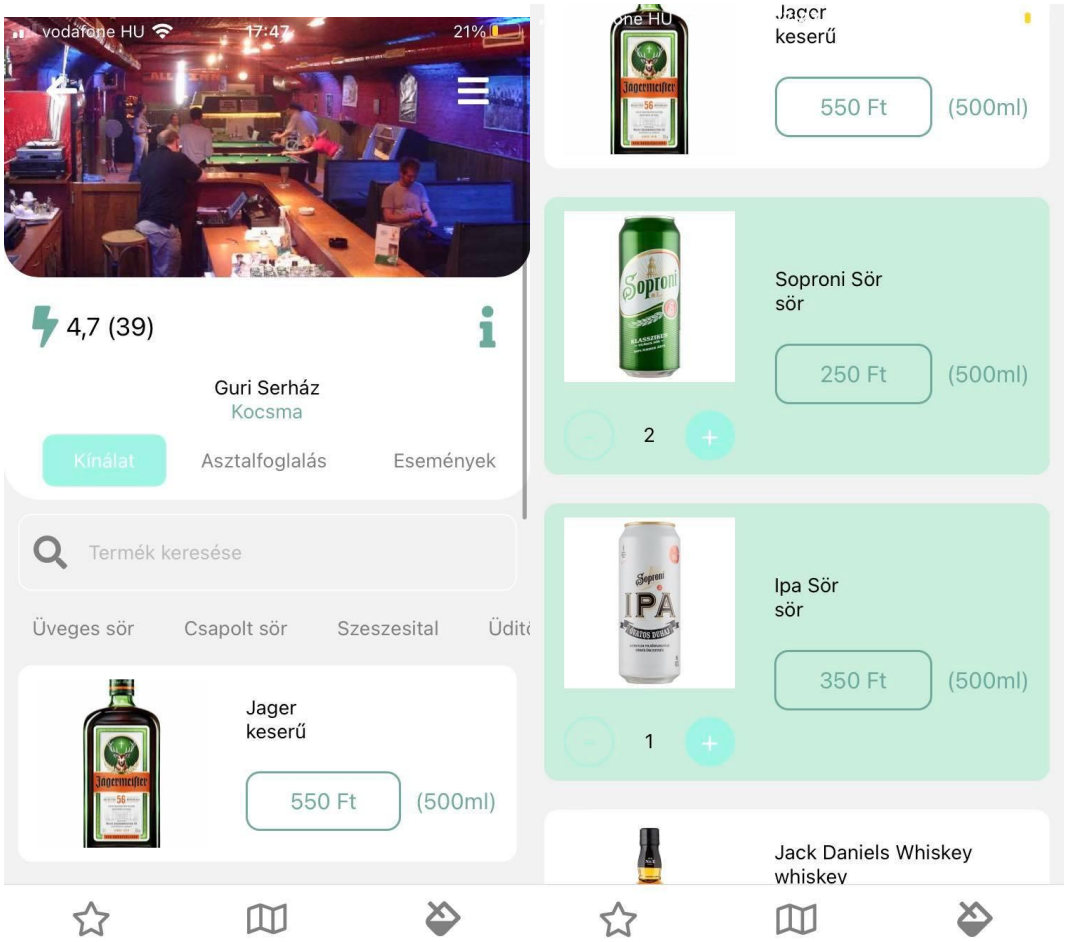


33. ábra: Applikáció térkép

Ezen a felületen maga a térkép komponens látható, ennek a sajátossága az, hogy a térkép könyvtár a telefon térkép komponensét használja, így eltér a megjelenése IOS-en és Androidon. A könyvtár neve React Native Maps és ez a legelterjedtebb térkép könyvtár napjainkban, mivel nem kell API kulcs ahhoz, hogy rendesen működjön, és nagyon széles a palettája a funkcióknak, amit használni tudunk körülbelül minden olyan funkciót tartalmazz, mint a Google Maps. Ezen kívül és kereső sávot is megfigyelhetünk magán a térkép komponensen, ahol rálehet keresni adott helyekre is, amik megjelenek számunkra a térképen.

34. ábra: Applikáció felfedezés

Ezen a felületen láthatjuk a felfedezést. A felületen megjelenik többféle szűrő is az első ilyen szűrő mikor konkrétan helyre vagy termékre keressünk a második szűrő pedig az első alatt helyezkedik el és itt kategóriánkként lehet szűrni magát a vendéglátóhelyeket. Minden hely egy darab kártyának feleltethető meg. Egy kártyán látható, hogy az adott hely milyen szolgáltatásokat biztosít és hogy hányan pontozták a helyet és ezeknek a pontoknak mennyi is az átlaga. A megnézem gombra kattintva vagyunk képesek átnavigálni az adott hely oldalára.

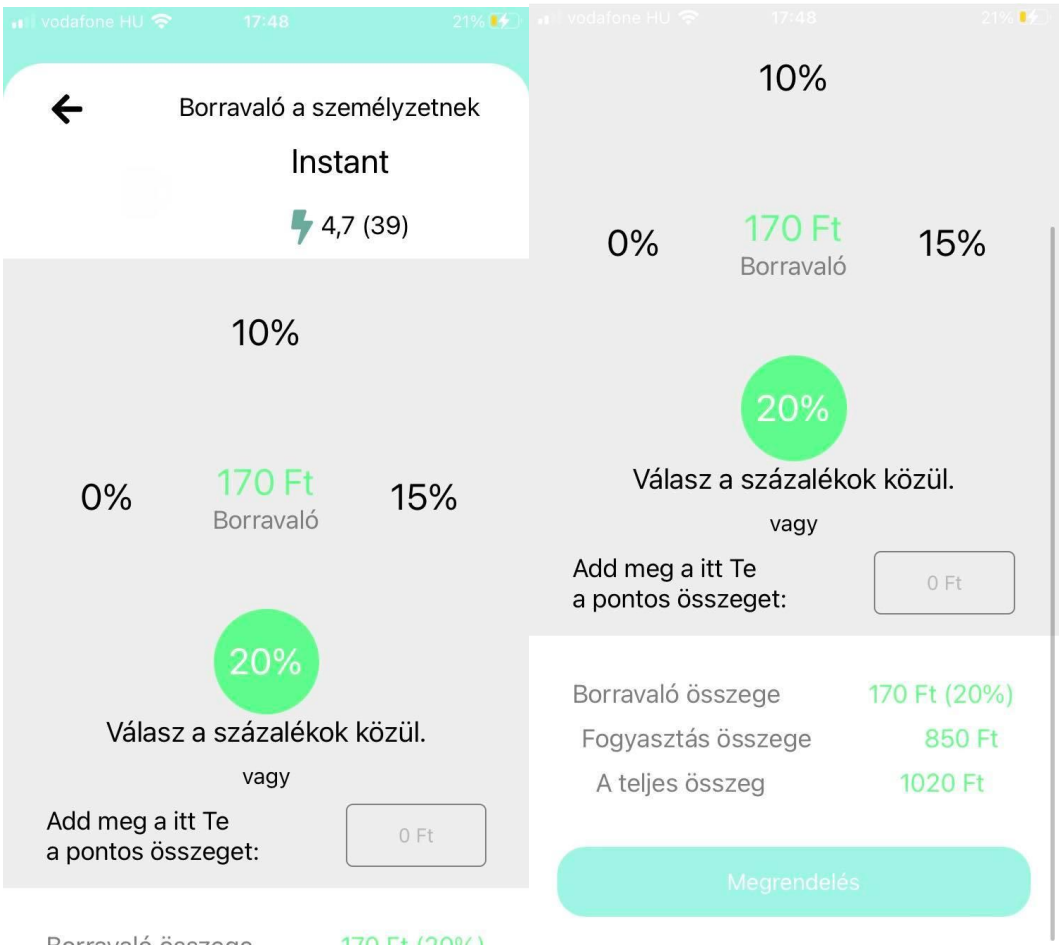


35. ábra: Vendéglátóhely

Ezen a felületen látható maga a vendéglátóhely, ahol képesek vagyunk leadni a rendeléseinket. Ez a felület is rendelkezik szűrővel, hogy képesek legyünk szűrni a termékeket. A termékeket úgy tudjuk belehelyezni a kosarunkba, hogy rányomunk egy termék kártyára akkor az adott termék aktív lesz ezt mutatja a zöld háttér is a kártyán. Amint aktív lett a kártya megjelenik egy plusz és egy mínusz gomb és egy számláló, hogy mennyi termék van éppen a kosarunkban. Ha egy adott terméket nem szeretnénk megrendelni akkor elég, ha levisszük a darabszámát 0-ra és akkor elmúlik a termék aktív státusza és kikerül a kosarunkból.

36. ábra: Kosár

A tálca felület összesíti nekünk össze a rendelésünket és ha a kosarunkban minden termék benne van, amit megrendelni kívánnunk akkor a Tovább a rendeléshez gombra kattintva megkezdhetjük a rendelés menetét. A felületen a táblázatot a React Native Paper-ből importáltam be a táblázat szerkezete és használati módja megfeleltethető a webes környezetben megtalálható táblázatokkal, így igen könnyen és gördülékenyen lehet akár összetettebb táblázatokat is alkotni és amit külön is kiemelnék, hogy táblázat komponens csak a React Native Paper-ben találhatunk, így reflektálnék a fent említett állításomra, hogy ennek a könyvtárnak a legnagyobb a komponens könyvtára.



37. ábra: Borravaló és rendelés

Ez az applikációnak az utolsó felülette itt már csak a borravalóról tudunk dönteni, hogy szándékozunk-e adni és ha igen akkor hány százalékot kívánunk adni, viszont, ha nem százalékban gondolkozunk akkor képesek vagyunk pontos összeget is megadni egy beviteli mezősegítségével a legvégén pedig kiírja számunkra, hogy mennyi volt maga a fogyasztás összege, a borravaló összege a végén meg összeadja számunkra, hogy mennyi is lesz a rendelésünk végösszege. A megrendelés gombra kattintva a rendelést leadjuk magának a vendéglátóhelynek.

11. Továbbfejlesztés

Természetesen, mint az összes informatikai rendszert, ezt az elkészült programot is lehetne és ajánlott is továbbfejleszteni, új modulokat, funkciókat integrálni. Első sorban a vendéglátóhely számára készült szoftver részt lehetne több olyan funkcióval kiegészíteni, ami az admin jogkörrel rendelkező személyeknek nyújtana nagy segítséget az egyik ilyen lenne a Munkatársak beosztásának böngészése ez a funkció véleményem szerint azért is lenne kimondottan hasznos, mert láthatóvá válna ki mikor dolgozott és így nyomon lehetne követni, ha esetleges eltérések lennének a készletben nyilvántartott termékek között. A másik hasznos tovább fejlesztési funkció a Beszállítások megtekintése, mert ha valamelyik termék már fogyatkozásban lenne, akkor egyszerűen meglehetne nézni mikor is jön új szállítmány az adott termékből. A másik kettő tovább fejlesztési ötlettem már nem funkció szintű az egyik ilyen ötlettem, hogy egy ML rendszert építünk rá az adattárházra, hogy képesek legyünk prediction-őket is mondani a múltbeli adatokból, nem csak historikusan ábrázolni azokat. Az utolsó tovább fejlesztési ötlettem, hogy magát a back-end-eket és az adatbázisokat átültetjük egy felhőbe és onnan fogva onnan fogja kiszolgálni a klienseket, ennek az előnye a teljesítmény növekedése, mivel nagyobb az erőforrás és maga a rendelkezésre állás. A rendelkezésre állás azért, mert ha például áramszünet lenne ott, ahol futtattom magát az alkalmazást kiszolgáló back-end-eket akkor elérhetetlenné válnának az applikáció bizonyos funkciói, viszont, ha felhőben futna akkor fel se merülhet ilyes fajta probléma. Az applikáció oldalról is akadnak tovább fejlesztési lehetőségek, mint például az asztalfoglalás beépítése, hogy ne csak rendelni tudjunk egy vendéglátóhelyről, hanem képesek is legyünk asztalt foglalni a helyen. A másik tovább fejlesztési ötletem a navigáció beépítése, hogy ne csak mutassa meg, hogy hol van a vendéglátóhely, hanem navigáljon is oda. Az utolsó tovább fejlesztési elképzelésem az applikáció oldalán, hogy ne csak a felhasználó profilját mentse el, hanem a korábbi rendeléseit is, hogy mikor a felhasználó belép az applikációba képes legyen az app ajánlásokat tenni, hogy adott helyen mi volt az utolsó rendelése és hogy szeretné-e újra rendeli azt.

12. Összefoglalás

A szakdolgozatom egy felmerülő probléma megoldására készült. Ez a probléma pedig nemmás mint, hogy a piacon nincs egy egységes, modern, jól átlátható és könnyen kezelhető szoftvercsomag, ami képes kielégíteni azt az igényt, hogy kényelmesen az asztaltól tudok rendelni és maga a pultos pedig egyszerűen tudja a leadott rendeléseket kezelni. A fejlesztés végére elértem a célomat, ugyanis képes voltam minden kitűzött célomat megvalósítani és minden elém guruló nehézséget megoldani. Mivel a konténerizálás mellett döntöttem, így maga az alkalmazás mobilis és erőforrás hatékony. A technológiák, amikben a szakdolgozatom íródott népszerűek és elterjedtek, így a későbbi tovább fejlesztések folyamán nem lesz probléma hasznos anyagot találni. A tesztelés folyamán a felhasználók elégedettek voltak a megvalósítással, de felmerültek újabb funkciók, melyek a továbbfejlesztési fázisban fognak majd szerepet kapni. A fejlesztés befejeztével rengeteg új dolgot tanultam, mint programozás, mint technikai megvalósítás tekintetében. Annak ellenére, hogy a fejlesztésnél használt programozási nyelvekkel és fejlesztői környezetekkel nem csak a szakdolgozat előtt találkoztam, hanem már több éve használtam ezeket hobbi szinten. Azonban a szakdolgozatomban az adattárháznál vagy éppen a sok különböző rendszer összekötésénél jobban belekellett mélyedni az adott programozási nyelvbe, hogy minden felmerülő problémát, úgy tudjak orvosolni, hogy a későbbiekben képes legyek karbantartani az adott kód részletet.

13. Summary

My thesis was prepared to solve an emerging problem. And this problem is nothing more than the fact that there is no uniform, modern, transparent and easy-to-use software package on the market that can satisfy the need to conveniently order from the table and the counter staff can easily manage the orders placed. At the end of the development, I reached my goal, because I was able to realize all my goals and solve all the difficulties that came before me. Since I decided in favor of containerization, the application itself is mobile and resource efficient. The technologies in which my thesis was written are popular and widespread, so it will not be a problem to find useful material during further development. During the testing, the users were satisfied with the implementation, but new functions emerged, which will play a role in the further development phase. After completing the development, I learned a lot of new things, both in terms of programming and technical implementation. Despite the fact that I did not only encounter the programming languages and development environments used in the development before the thesis, I have been using them as a hobby for several years. However, in my thesis, I had to delve deeper into the given programming language than the data warehouse or the connection of many different systems, so that I could remedy any problems that arose, so that I would be able to maintain the given code detail later on.

14. Ábrajegyzék

15. Táblázatjegyzék

16. Irodalomjegyzék

[1] „Wikipédia,” [Online]. Available: https://hu.wikipedia.org/wiki/Kliens-szerver_architekt%C3%BAra. [Hozzáférés dátuma: 07 05 2022]. Utoljára megtekintve: 2022.11.10

[2] [Online]. Available: https://www.redhat.com/en/topics/api/what-is-a-rest-api. Utoljára megtekintve: 2022.11.23

[3] [Online]. Available: https://hu.gadget-info.com/difference-between-oltp. Utoljára megtekintve: 2022.11.12

[4] [Online]. Available: https://hu.wikipedia.org/wiki/SQL. Utoljára megtekintve: 2022.11.15

[5] [Online]. Available: https://azure.microsoft.com/hu-hu/overview/nosql-database. Utoljára megtekintve: 2022.11.26

[6] [Online]. Available: https://www.liquibase.org/. Utoljára megtekintve: 2022.11.20

[7] [Online]. Available: https://mui.com/. Utoljára megtekintve: 2022.11.21

[8] [Online]. Available: https://www.techopedia.com/definition/27568/native-mobile-app, . Utoljára megtekintve: 2022.11.10

[9] [Online]. Available: http://weblabor.hu/cikkek/nosql-bevezeto. Utoljára megtekintve: 2022.11.10

[10] [Online]. Available: https://hu.wikipedia.org/wiki/ACID. Utoljára megtekintve: 2022.11.10