Miskolci Egyetem

Gépészmérnöki és Informatikai Kar

Szakdolgozat

Bolti készletnyilvántartó rendszer

Készítette:

Sárosi Bence

Üzemmérnök informatikus BProf

Témavezető:

Szücs Miklós

Mesteroktató

Miskolc,2024

Tartalom jegyzék

Köszönetnyilvánítás

Szeretnék köszönetet mondani témavezetőmnek, Szűcs Miklós tanár úrnak, aki az útmutatásaival, szakmai tanácsaival segített és bátorított a nehezen induló szakdolgozatom elkészítéséhez. Köszönöm a Java alapismeretek tantárgyhoz készített anyagait, ami a kollégiumban való tanuláshoz segítségemre volt.

Szeretném megköszönni Dr. Baksáné Dr. Varga Erika tanárnő kitüntető figyelmét és szigorúságát OOP tantárgyból, amivel ráébresztett a tudás megszerzésének hasznosságára és a kerülőutak elhagyására.

Köszönöm Dr. Baksa Attila tanár úr Unix alapismeretek tantárgyból megtartott elméleti óráit és az azokra kis csoportunknak külön példákat hozó gyakorlati foglalkozásait.

Mindig emlékezni fogok Ficsor Lajos gyakorlatvezető tanáromra C programozás és Java I. programozás tantárgyakból, aki már nem érhette meg ezen mű elkészültét.

Végül de nem utolsó sorban szeretném megköszönni a Miskolci Egyetem Gépész és Informatikai Kar valamennyi oktatójának és dolgozójának a lelkiismeretes munkáját, amivel hozzájárult szakmai fejlődésemhez.

Köszönöm középiskolai tanáraimnak a továbbtanulás irányába tett erőfeszítéseit: Kiszely Anna Andrea, Vágó Tamás.

Köszönöm nagyszüleimnek azokat a boldog nyaralásokat, szüleimnek és öcsémnek, hogy szeretetükkel és tanácsaikkal támogatták tanulásomat.

Külön köszönet édesapámnak és Sárosi Ádám nagybátyámnak, akik külső konzulensként építő megjegyzésekkel segítették munkámat.

Szeretném megköszönni régi középiskolai barátaimnak, akik velem voltak ezen az úton és segítettek engem. Köszönöm a közös tanulás óráit, amelyek során nem csak a tananyagot, de egymást is jobban megismertük. Köszönöm a hülyeségeket, amiket a kollégiumban és az egyetemen csináltunk, ezek adták a mindennapokhoz szükséges könnyedséget és derűt. Tudom, hogy ezek a segítségek nagyban hozzájárultak ahhoz, hogy ma itt állhatok. Köszönöm, hogy segítettetek abban, hogy kitartó legyek és hogy soha ne adjam fel.

Köszönöm, hogy velem tartottatok, és remélem, hogy a jövőben is számíthatok rátok, ahogy ti is számíthattok rám.

1. Bevezetés

1.1 Előzmények

A szakdolgozatom témája a Bolti Készletnyilvántartó Rendszer, amely egy terméknyilvántartó és számlázó alkalmazás, kifejezetten egy családi kisvállalkozás számára. Döntésemet az indokolja, hogy a napjainkban működő vállalkozásoknál fennmaradásuk érdekében létfontosságú a logisztika bevezetése és folyamatos fejlesztése, amely egy dinamikusan fejlődő tudományág.

Raktárkezelő szoftverek nélkül ma már lehetetlen nyomon követni és ellenőrizni a raktári tevékenységeket, valamint kezelni a készleteket és a leltározást. Még egy kis webáruháznak is hamarosan többre lesz szüksége, mint néhány Excel-táblázatra, hogy nyomon kövesse az összes nyilvántartást, ami a hibák esélyével, ami csökkentheti a hatékonyságot, és végső soron veszteségeket eredményezhet. Mik tehát az alternatívák? Mikor van szükségünk raktárkezelő szoftverre, és mit várhatunk tőle? Itt vesszük szemügyre a piacon elérhető legjobb szoftverlehetőségek közül néhányat. A raktárkezelő szoftver elengedhetetlen a leltározási folyamatok automatizálásához, mivel segít a raktári információk gyors rögzítésében és pontos tárolásában. Ez a szoftver lehetővé teszi a készletmozgások nyomon követését és valamint az egyedi terméktörténetek rögzítését. Emellett a szoftver megkönnyíti a készletgazdálkodást és -elemzést az emberi hibák csökkentése érdekében. Fő célja a valós idejű készlet lekérdezés biztosítása. A "raktárkezelő szoftver" kifejezést gyakran felváltva használják a "készletgazdálkodási szoftverrel", mivel a kettő között általában sok átfedés van. A félreértések elkerülése érdekében tisztázzuk a köztük lévő fő különbséget. Egy raktárkezelő program elsősorban a raktári tevékenységekkel foglalkozik, például a termékek mozgatásával, tárolásával és adatainak kezelésével, míg a készletgazdálkodási szoftverek nagyrészt azzal foglalkoznak, hogy mi történik a termékkel, miután az elhagyja a raktárat. A készletgazdálkodási program a készlet jellemzőire, valamint a virtuális környezetben történő mozgására összpontosít. Ezt a kettős célt gyakran kombinálják, és egy gyakorlati példa is megmutatja a különbséget: egy készletgazdálkodási program jellemzően nem követi nyomon a raktáron belüli összes tevékenységet, míg egy készletgazdálkodási rendszer nem mindig tartalmaz számlázást.

1.2 Felhasználói környezet

Egy raktárkezelő programnak számos fontos funkciót kell támogatnia annak érdekében, hogy hatékonyan kezelje a raktárakat és segítse a vállalkozásokat az áruk kezelésében. azok a konkrét lépések, amiket egy raktárkezelő programnak tudnia illik:

- A termékszállítmány kézhezvétele után az első teendő, hogy elpakolja a termékeket, és nyomon követi, hogy hová kerültek.
- Képesnek kell lenned a vásárlói megrendelések kezelésére, a termékek kiválasztására és a megrendelések teljesítésére.
- Nyomon kell követnie az áruforgalmat, a szükségletektől függő részletességgel.
- Mindenképpen képesnek kell lennie arra, hogy tájékoztatást adjon a készletezéssel, az átadással, a kiválasztással, a kiszállítással és az esetleges visszáruval kapcsolatban. Egyes programok még arra is képesek, hogy a raktári gépek tevékenységét nyomon kövessék.
- A rendeléscsomagolással összefüggésben látható, hogy mely termékeket csoportosították és melyik címzettnek küldték el.
- A raktárkezelő programok képesek leltárakat és különféle egyéb jelentéseket generálni és összeállítani.

Az alábbiakban felsorolok néhány olyan funkciót, amit egy jó raktárkezelő programnak érdemes támogatnia:

1. **Készletnyilvántartás:**
   - Pontos és valós idejű készletinformációk nyomon követése.
   - Részletes termékadatok, például terméknév, leírás, kategória, beszerzési és eladási árak.
2. **Rendelések kezelése:**
   - Beérkező és kimenő rendelések nyomon követése.
   - Automatikus rendeléskészítés és raktárellenőrzés.
3. **Készletmozgások kezelése:**
   - Készletmozgások és tranzakciók rögzítése (pl. átrakás, átvétel, selejtezés).
   - Nyomon követés, hogy ki és mikor hajtott végre készletmozgást.

4. **Raktári elrendezés és térkép:**
    - Virtuális térkép a raktárban lévő termékek elrendezésének nyomon követésére.
    - Rendszeres raktári ellenőrzési folyamatok támogatása.
5. **Készletfigyelmeztetések:**
    - Automatikus értesítések alacsony készletszintekről vagy túl sok készletről.
    - Készletmozgásokat és változásokat követő értesítések.
6. **Raktármenedzsmenti riportok:**
    - Részletes riportok a készletmozgásokról, készletszintekről, eladásokról stb.
    - Elemzések a készlettel kapcsolatos teljesítményről.
7. **Integrációk más rendszerekkel:**
    - Integrációk más üzleti szoftverekkel, például pénzügyi szoftverek, értékesítési platformok.
8. **Raktárkezelési rendszerek és kódolások támogatása:**
    - Különböző raktárkezelési módszerek (pl. FIFO, LIFO) támogatása.
    - Vonalkódok és RFID technológia támogatása.
9. **Felhasználói jogosultságok:**
    - Különböző felhasználói szintek és jogosultságok beállítása.
    - Raktári tevékenységek naplózása és nyomon követése.
10. **Mobilalkalmazás:**
    - Hozzáférés és raktárkezelési funkciók elérhetősége mobilkészülékekről is.

Az "egyik legjobb" raktárkezelő program kiválasztása számos tényezőtől függ, beleértve a vállalkozás méretét, az igényeket és a költségvetést. Néhány népszerű raktárkezelő szoftver közé tartozik például az Odoo, a Zoho Inventory, a Fishbowl, a QuickBooks Commerce és a WMS (Warehouse Management System) rendszerek, mint például az SAP EWM (Extended Warehouse Management) vagy a Manhattan Associates WMOS (Warehouse Management for Open Systems). Minden esetben ajánlott alaposan elemezni a vállalkozás igényeit, és a rendszer kiválasztásánál figyelembe venni a szükségleteket és az elérhető lehetőségeket.

Itt az ideje, hogy a konkrétumokra koncentráljunk. A hazai piacon jelenlévő legjobb programok közül néhányat ajánlok, figyelembe véve azok kivételes tulajdonságait és költségeit. Tekintettel arra, hogy a vállalkozások és az e- kiskereskedők szükségletei nem azonosak, nem létezik minden webáruház számára egy véglegesen legjobb szoftver, ezért előnyös az ajánlatok és a tulajdonságok óvatos elemzése:

- **Kulcs-Soft**: A Kulcs-Soft felhőalapú szoftvercsomagokat kínál vállalati ügyfeleknek. Ezek a csomagok irodai, bérszámfejtési és számlázási szoftvereket, valamint moduláris és bővíthető szoftvereket tartalmaznak, amelyek a leltárkészletet és a különböző vállalatirányítási funkciókat ötvözik. A számlázáshoz, raktárkezeléshez és házipénztárhoz tartozó egységes felület a platform egyik legnagyobb előnye. A Key-Soft 30 éves megbízható, bizonyított tapasztalattal rendelkezik, és kifejezetten az online boltok igényeihez igazított szolgáltatásokat kínál [1].

1. ábra (https://www.kulcs-soft.hu/)

- **OVIP**: Az OVIP átfogó, adaptálható és bővíthető felhőalapú szolgáltatásokat nyújt, amelyek a vállalatirányítás egészét lefedik. Ezen túlmenően rendelkeznek egy kifejezetten raktárkezelésre kifejlesztett modullal, amely kifejezetten a hagyományos bolti értékesítésre és a webalapú áruházakra egyaránt szabott. A termékeket gyártó és a bérbeadó vállalatok számára személyre szabott szoftverük kifejezetten ajánlott [1].

2. ábra (https://www.ovip.hu/)

- **Naturasoft:** A Naturasoft raktárkezelő programját úgy tervezték, hogy felhasználóbarát és a kezdő webáruházak számára is elérhető legyen. Ez egy praktikus, átlátható és egyszerű megoldás, amely leltárkövető és számlázó szoftvert is tartalmaz. A Naturasoft a legfontosabb funkciók kiemelésével a felhasználóbarátságra helyezi a hangsúlyt, miközben nem köt kompromisszumot a teljesség terén. Emellett telefonos támogatási rendszert is kínálnak az ügyfelek számára [1].

3. ábra (https://www.naturasoft.hu/raktarkezelo_program.php)

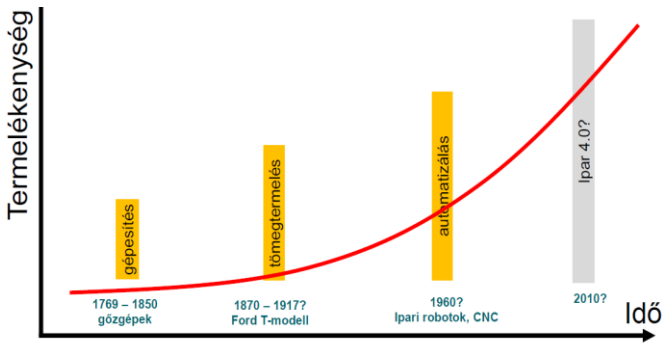
- **Logicort:** A Logicort naprakész, automatizált nyomon követési rendszert biztosít a raktárak számára, amely átfogó képet ad a raktári tevékenységről. Ez magában foglal egy interaktív térképet, amely pontosan megmutatja, hogy melyik targonca mozog a raktárban. Ez a program a raktárkezelésre van szabva, és a vállalatirányítási szoftvercsomagokhoz képest nagyobb részletességgel és speciális funkciókkal rendelkezik. A termékek azonosítását EAN- kódok vagy RFID -technológia támogatja [1].

4. ábra (https://www.logicort.hu/)

1.3. Célok, követelmények

Szakdolgozatom célja egy speciális adatbázis-kezelő szoftver kifejlesztése volt, gyártó és kisvállalatok számára, grafikus felhasználói felülettel. Ez a szoftver lehetővé teszi számukra, hogy raktározásukat és logisztikájukat intuitív és egyszerű módon, előzetes informatikai tapasztalat nélkül kezeljék.

Ezen alkalmazással a kisvállalkozás irányításához kaphatunk adatokat, így kihasználják a negyedik ipari forradalom fejlődését. Az ipari forradalmat 4 részre tudjuk osztani.



5. ábra ipar 4.0

Digitális technológia segítségével okosabban, kevesebb alkalmazottal dolgozhatunk, miközben a logisztikát, a raktározást és a valós idejű nyomonkövetés PC-ről vagy mobileszközökről használhatunk. Egy számítógépes program működésének megértése nagy kihívást jelenthet, ezért kiemelt fontosságúnak tartottuk, hogy a program intuitív, felhasználóbarát felülettel rendelkezzen. Ez lehetővé teszi, hogy még a számítógépes programozásban korlátozott tapasztalattal rendelkezők is nehézség nélkül használhassák a programot. A digitális átállás drasztikusan felgyorsítja és egyszerűsíti a munkafolyamatokat, és a digitális műveletek révén csökkenő papírfogyasztás a környezetre is jótékony hatással van.

Az egyetemi tanulmányaim során olyan digitális kompetenciákat szereztem, amelyek a szoftverek fejlesztéséhez szükségesek. Ezek a képességek bizonyos kritériumoknak kell megfeleljenek ahhoz, hogy a szoftver működőképesnek minősüljön. Ezek az alapvető követelmények a következők:

- Grafikus megjelenési felület,
- Könnyen értelmezhető, felhasználóbarát megjelenés,

- A felhasználói felületek ma már gyakran tartalmaznak gombokat, listákat és menüket ahelyett, hogy parancssort kellene használniuk.
- Bejelentkezési rendszer bevezetése az adatbázishoz való hozzáférés biztosítására felhasználónév és jelszó megadásával.,
- Az adatbázis biztonsága érdekében egy felhasználónévvel és jelszóval ellátott bejelentkezési rendszert kell bevezetni,
- A hozzáférési rendszernek képesnek kell lennie a helyi és távoli csatlakozásra.,
- Az információk beírásának, szerkesztésének és eltávolításának lehetősége közvetlenül az adattáblában.,
- Egy gomb, amely lehetővé teszi a felhasználók számára az adatok törlését közvetlenül a táblázatból.,
- Az adatbázis védelme érdekében minden törlés előtt visszaigazolást kell kérni,
- Az adatbeviteli ablakot világos és felhasználóbarát megjelenítéssel tervezték,
- Annak biztosítása, hogy az adatbeviteli ablakban megadott értékek összhangban legyenek a táblázatok megkötéseivel és követelményeivel
- A felhasználó által megadott értékek alapján az adatbeviteli ablak elvégzi a kiadott parancsok szemantikai ellenőrzését,
- Bruttó, nettó és diszkontált árak kiszámítása az önköltségből matematikai képletek segítségével szoftvereken és adatbázisokban,
- A számított értékek biztonságának biztosítása,
- Lehetőség van az " akciós " tételek bemutatására és arra, hogy csak ezeket a tételeket lehessen bemutatni,
- Naplóvédelem, amely utólag nem módosítható vagy törölhető.
- Lehetőség van a megjelenített adatok egyidejűleg több érték alapján történő szűrésére,
- Keresés az adatbázisban több mező értékének egyidejű vizsgálatával,
- A táblázatokat futás közben tetszés szerint átrendezheti,
- A felhasználó által bevitt adatok ellenőrzése mind az adatbázisban, mind az alkalmazásban.,
- Hibás adatbevitel esetén a hibás mezők jelölve lesznek, és a helyes szintaxis segít a javításban.,
- A programból, adatbázisból és adatkapcsolatból eredő hibák elhárítása

2. Követelmények

Amikor egy általános szolgáltatást vagy állapotot egy konkrét matematikai függvényre kell átalakítani, különböző típusú követelmények használhatók. Ezeknek a követelményeknek két elsődleges célja van:

- szolgálhatnak egy nyílt pályázati felhívás alapjául, lehetővé téve a különböző megközelítések mérlegelését;
- egy részletes szerződés alapjául is szolgálhatnak. Ezeket a követelményeket mindkét esetben "követelmények"-nek nevezik.

„Ha egy megrendelő egy nagy szoftver fejlesztésére pályázatot ír ki, az igényeket kellően absztrakt módon kell megfogalmaznia úgy, hogy azok a megoldást előre ne határozzák meg. Úgy kell a követelményeket kiírni, hogy több pályázó is versenyezhessen, lehetőleg több alternatív megoldási módot kínálva a megrendelő igényeinek kielégítésére. A pályázat győztesének ezután a megrendelő számára egy, a rendszert részleteiben leíró tervezetet kell készítenie, amelyből a megrendelő megértheti és ellenőrizheti a rendszer tevékenységét. Mindkét említett dokumentumot a rendszer követelmény-dokumentumának nevezzük.”

(**Davis**)

2.1 Funkcionális követelmények

A funkcionális követelmények meghatározásával biztosíthatjuk, hogy a rendszer a felhasználó minden igényét kielégítse. Biztosítanunk kell, hogy ezek a követelmények két feltételnek megfeleljenek: a teljesség elvének, amely megköveteli, hogy a felhasználó által kért összes funkció és szolgáltatás meg legyen határozva, és hogy a követelmények ne legyenek egymással ellentétesek.

1. Regisztráció:
    - A rendszernek meg kell könnyítenie egy új ügyfél, alkalmazott vagy vezető beiratkozását.
    - A regisztrációs ablaknak lehetővé kell tennie a rendszerbe való visszatérést.

2. Bejelentkezés:
    - A rendszernek lehetővé kell tennie az ügyfelek, alkalmazottak, vezetők és adminisztrátorok számára, hogy a megfelelő felhasználónévvel, jelszóval és szerepkörrel jelentkezzenek be.
    - Az új felhasználóknak képesnek kell lenniük a rendszerben való regisztrációra.

3. Magán adatok:
    - A felhasználónak képesnek kell lennie arra, hogy a rendszeren keresztül hozzáférjen a személyes adataihoz, és megnézze azokat.
    - A felhasználónak képesnek kell lennie arra, hogy a rendszeren keresztül módosítsa személyes adatait, például e-mail címét vagy jelszavát.
    - A felhasználónak meg kell adni a lehetőséget, hogy visszatérjen a fő menübe, vagy kilépjen az alkalmazásból.
4. Központ:
    - Az adminisztrációs irodának képesnek kell lennie arra, hogy az adatbázist beépítse a rendszerbe.
    - Az adminisztrátornak képesnek kell lennie arra, hogy a rendszeren belül változtatásokat hajtson végre az iroda adatain.

2.2 Nem funkcionális követelmények

Meghatározásra kerülnek a rendszerkövetelmények és működési feltételek, például a megbízhatóság, a válaszidő és a tárolási követelmények. A korlátozások közé tartozhatnak az I/O eszköz képességei, adatformátumok stb. A nem funkcionális követelmények gyakran kritikusabbak, mint a funkcionális követelmények; ha nem teljesülnek, a rendszer használhatatlan, míg az első esetben csak bizonyos funkciók nem működhetnek. A fejlesztési követelmények is meghatározhatók, beleértve egy adott CASE eszköz, programozási nyelv vagy fejlesztési módszer használatát. Ezek nem a műveletek és szolgáltatások sajátosságaihoz kapcsolódnak, hanem inkább a szolgáltatásokkal kapcsolatos különböző feltételeket írják le

1. Biztonság:
    - A felhasználók beleértve az ügyfelek, alkalmazottak, vezetők és adminisztrátorok minden jelszava titkosítva van, és egy hash-algoritmus segítségével tárolódik, így egyszerű szövegben olvashatatlan.
    - Bizonyos szerepkörök nem rendelkeznek olyan jogosultságokkal, amelyekkel más szerepkörök képességeit használhatják.
2. Teljesítmény:
    - A reakcióidő nagy részét a kapcsolati csatorna határozza meg, vagyis az alkalmazás és az adatbázis közötti kapcsolat sebessége.
    - A rendszernek bármely parancsra 15 másodpercen belül vagy annál rövidebb idő alatt kell válaszolnia.
3. Elérhetőség:
    - Az asztali alkalmazás elérhetősége internetkapcsolat meglététől nem függ.
4. Erőforrás:
    - Az alkalmazás nem igényel sok erőforrást, mivel nem használ bonyolult grafikai elemeket.
5. Kezelés:
    - Ez a szoftver felhasználóbarát, az egyes szerepkörökhöz egyszerű és világos menükkel, amelyek bármelyik almenüből elérhetők.

3. Alkalmazott technológiák

A fejlesztési folyamat kezdetén fontos volt a megfelelő környezet és technológiák gondos mérlegelése és kiválasztása. Ez magában foglalta a szoftver futtatására használt operációs rendszert, a szoftver elkészítéséhez használt nyelvet, a szoftver futtatásához szükséges keretrendszert, a fejlesztőkörnyezetet és az adatbázis-kezelő rendszert. Mivel Magyarországon évek óta a Microsoft Windows az uralkodó operációs rendszer, és mind szakmai, mind tudományos munkám során ezt használtam, a választásom egyszerű volt.

3.1 Java programozás nyelv

A Java egy sokoldalú, objektumorientált programozási nyelv, amelyet a Sun Microsystems fejlesztett ki az 1990 -es évek elején. A Sun Microsystems-t 2009-ben felvásárolta az Oracle. A Java alkalmazásokat jellemzően bytecode-ba konvertálják, de lehetőség van natív gépi kód létrehozására közvetlenül a Java forráskódból is. A bájtkódot a Java virtuális gép hajtja végre, amely vagy értelmezi a bájtkódot, vagy natív gépi kódot generál belőle, és azt futtatja az operációs rendszeren. Létezik olyan hardver is, amely közvetlenül képes a Java bájtkód futtatására, ez az úgynevezett Java processzor.

A Java szintaxisát nagyrészt a C és a C++ nyelvtől örökölte, de objektummodellje egyszerűbb, mint a C ++-é. A hasonló név és szintaxis ellenére a JavaScript nem olyan szoros rokonságban áll a Javával, mint azt sokan gondolják. Kezdetben a nyelvet Oaknak hívták, a Java megalkotójának, James Goslingnak az irodája előtt nőtt tölgyfáról elnevezve. Később azonban kiderült, hogy már létezett egy ilyen nevű nyelv, így a Java nevet fogadták el. A " Java " szó az Oracle védjegye, így engedély nélkül nem használható mások által kifejlesztett termékek megjelölésére, még olyan összetételben sem, mint például "Java-like", mivel ez sérti a védjegy tulajdonosának jogait.

3.2 IntelliJ IDEA

IntelliJ IDEA az egyik leginnovatívabb és legnépszerűbb integrált fejlesztői környezet (IDE), amelyet a JetBrains fejlesztett ki. Az IDEA különlegessége, hogy nem csupán Java nyelvhez kínál teljes körű támogatást, hanem más programozási nyelvekhez is, mint például Kotlin, Scala, Groovy és még sok más. Az IntelliJ IDEA egy olyan eszközkészletet biztosít, amely lehetővé teszi a fejlesztők számára, hogy hatékonyan dolgozzanak, minimalizálva a hibalehetőségeket és maximalizálva a termelékenységet.

Az IDEA fő jellemzői közé tartozik az intelligens kódolás segédprogramok széles skálája. Ezek az eszközök automatikusan javítják a kódot, segítenek a kódszerkesztésben, és gyorsan áttekintést nyújtanak az elérhető osztályokról és metódusokról. Emellett könnyen importálhatók a szükséges osztályok és csomagok, így a fejlesztők gyorsan és hatékonyan írhatnak kódot [3].

Az IntelliJ IDEA erős refaktorálási eszközöket kínál, amelyek lehetővé teszik a kód átszervezését és újra strukturálását anélkül, hogy megváltoztatnák annak működését. Ez segíti a kód tisztábbá tételét, könnyebben érthetővé tételét és könnyebb karbantarthatóságát [3].

Az IDE könnyű integrációt biztosít más fejlesztői eszközökkel és keretrendszerekkel, mint például a Git verziókezelő rendszer és a Maven vagy Gradle build eszközök. Ez lehetővé teszi a fejlesztők számára, hogy könnyen együttműködjenek másokkal és hatékonyan kezeljék a projektjeiket [3].

Az IntelliJ IDEA erős debuggolási környezettel is rendelkezik, amely lehetővé teszi a fejlesztők számára, hogy hatékonyan hibakeresést végezzenek és javítsák a kódhibákat. A könnyen követhető végrehajtási lépések, a változók állapotának megfigyelése és az interaktív hibakeresési lehetőségek mind hozzájárulnak a fejlesztői folyamat hatékonyságához és minőségéhez [3].

A JetBrains által kifejlesztett IntelliJ IDEA egy nagyra értékelt Java -fejlesztő platform, amely átfogó támogatást nyújt különböző programozási nyelvekhez, keretrendszerekhez és technológiákhoz. Intelligens szerkesztője, kódelemzője és fejlett refaktorálási képességei a fejlesztők hatékony eszközévé teszik. A forráskód indexelésével az IntelliJ IDEA képes gyors és intelligens javaslatokat adni a kódfejlesztéshez. Olyan funkciókkal is segíti a fejlesztők

munkáját, mint az azonnali és intelligens kódkiegészítés, a valós idejű kódelemzés és a robusztus refaktorálás. Ezek az eszközök, valamint az integrált verziókezelő rendszerek, a széles körű nyelvi támogatás és a keretrendszerek mind zökkenőmentesen integrálódnak a szoftvercsomagba, így nincs szükség további bővítményekre [3].

Kétféle termék változata ismert:

- A Community Edition a szoftver nyílt forráskódú változata, amely további bővítmények nélkül támogatja a Java, Kotlin, Groovy és Scala nyelveket. Támogatja az Android- fejlesztést is, mivel maga az Android Studio az IntelliJ IDEA módosított változata. A Community Edition tartalmazza a Maven, Gradle, Git, SVN és Mercurial verziókezelő rendszerek támogatását, és ingyenesen használható kereskedelmi fejlesztésekhez [3].

- Az Ultimate Edition egy havi vagy éves előfizetéses verzió, amely egy év után örökös licencet biztosít az előző megvásárolt verzióra. További funkciókat és támogatást kínál a vállalati és webes fejlesztéshez (Java EE, Spring, Grails és más keretrendszerek), valamint a kódminőség és a teljesítmény javítását szolgáló eszközöket. Emellett integrálódik a DataGrip adatbázis-eszközökkel és SQL-támogatással. Emellett az Ultimate Edition alapértelmezett JavaScript- és TypeScript- támogatást is tartalmaz [3].

Az IntelliJ IDEA egy átfogó, erőteljes és könnyen használható fejlesztői környezet, amely lehetővé teszi a fejlesztők számára, hogy hatékonyan dolgozzanak és kiváló minőségű szoftvereket hozzanak létre.

3.3 Scenebuilder

SceneBuilder egy grafikus felhasználói felület (GUI) tervezőeszköz, amelyet a JavaFX alkalmazásokhoz használnak. A JavaFX egy olyan platform, amely lehetővé teszi a gazdag, interaktív asztali és webes alkalmazások készítését Java nyelven. A SceneBuilder segítségével a fejlesztők intuitív módon tervezhetik meg az alkalmazások felhasználói felületét, anélkül, hogy kódot kellene írniuk.

A SceneBuilder egyedi előnye, hogy lehetővé teszi a felhasználók számára, hogy vizuálisan elrendezzék az alkalmazás elemeit, például gombokat, szövegdobozokat, képeket és táblázatokat, majd azokat összekapcsolják az alkalmazásban lévő funkciókkal. Emellett támogatja az FXML (FXML Markup Language) használatát, ami egy XML-alapú nyelv JavaFX alkalmazások felhasználói felületeinek leírására [2].

A SceneBuilder könnyen használható és intuitív felhasználói felülettel rendelkezik, amely lehetővé teszi a gyors és hatékony GUI-tervezést. A fejlesztők egyszerűen húzhatnak és ejthetnek elemeket a tervezőfelületre, majd testre szabhatják azokat a tulajdonságok panelen keresztül. Ez a vizuális szerkesztési folyamat segíti a fejlesztőket abban, hogy gyorsan létrehozzanak professzionális megjelenésű alkalmazásokat, minimalizálva a tervezési időt és a hibalehetőségeket [2].

Amikor a tervezés befejeződik, a SceneBuilder lehetőséget biztosít az alkalmazás felhasználói felületének exportálására FXML fájlba, amelyet aztán a JavaFX alkalmazásokban lehet használni. Ez a fájl tartalmazza az összes információt az alkalmazás grafikus felhasználói felületéről, így a fejlesztők könnyen beilleszthetik azt az alkalmazás forráskódjába és módosíthatják azt igény szerint [2].

A SceneBuilder egy hatékony eszköz a JavaFX alkalmazások felhasználói felületének tervezésére és létrehozására. A könnyen használható felhasználói felület és a vizuális szerkesztési lehetőségek segítenek a fejlesztőknek a gyors és hatékony GUI-fejlesztésben, csökkentve ezzel az alkalmazások fejlesztési időtávját és minőségét [2].

.

3.4 MYSQL

MySQL az egyik legismertebb és legelterjedtebb relációs adatbáziskezelő rendszer (RDBMS), amely a világ legkülönfélébb alkalmazásaiban használatos. A MySQL kiváló választás minden olyan szervezet vagy fejlesztő számára, akiknek hatékony adattárolásra, gyors adatelérésre és megbízható adatkezelésre van szükségük. Az alábbiakban bemutatjuk a MySQL-t, a használatát, a funkcióit és annak történetét, hogy hogyan vált az egyik legfontosabb technológiává az adatvezérelt alkalmazások világában [4].

Először is, a MySQL széles körű funkcionalitása lehetővé teszi az adatok hatékony tárolását és kezelését.

Másodszor, a MySQL könnyen integrálható más szoftverekkel és programozási nyelvekkel, például Pythonnal vagy PHP-vel. Ez lehetővé teszi számunkra, hogy az adatokat könnyedén betároljunk vagy kiolvassunk az adatbázisból, valamint lekérdezéseket és műveleteket végezzünk az adatokon a szakdolgozatban használt programozási környezetben [4].

Emellett a MySQL rendelkezik különféle eszközökkel és funkciókkal, amelyek segítenek az adatok elemzésében és vizualizálásában. Például lekérdezéseket és aggregációkat használhat a különböző adatok összehasonlítására vagy összefüggéseik elemzésére. Ezenkívül grafikus eszközök is elérhetők a MySQL-hez, amelyek segítenek az adatok grafikus ábrázolásában és

- **Webfejlesztés**: A MySQL a legnépszerűbb adatbáziskezelő rendszer a webfejlesztésben. Számos webalkalmazás, például WordPress, Drupal és Joomla, MySQL-t használ az adatok tárolására.
- **Üzleti alkalmazások**: Nagyvállalatok és kisvállalkozások egyaránt használják a MySQL-t üzleti alkalmazásokban, például az ügyfélkezelő rendszerekben, az értékesítési adatbázisokban és a pénzügyi rendszerekben.
- **Mobilalkalmazások**: A MySQL kiválóan alkalmas a mobilalkalmazásokhoz való adattárolásra és szinkronizálásra. Számos mobilalkalmazás backendjében használják az adatok tárolására és kezelésére.
- **Adatelemzés**: A MySQL lehetővé teszi az adatelemzési munkafolyamatokat is. Számos cég használja az adatok tárolására és elemzésére az üzleti intelligencia és adatelemzési célokra.

A MySQL számos funkciót kínál, amelyek lehetővé teszik az adatok hatékony kezelését és kihasználását:

- **Tranzakciók támogatása**: A MySQL támogatja a tranzakciókat, amelyek biztosítják az adatbázis konzisztenciáját és integritását.
- **Indexek és kulcsok**: Az indexek és kulcsok segítenek a gyors adatelérésben és a lekérdezések hatékonyságának növelésében.
- **Különféle tárolók és motorok**: A MySQL különféle tárolókat és motorokat kínál, amelyek lehetővé teszik az adatok hatékony tárolását és kezelését az adott alkalmazás igényeinek megfelelően.
- **Számos adattípus támogatása**: A MySQL számos adattípust támogat, beleértve a vizualizációjában.

Végül, a MySQL skálázható és megbízható megoldást kínál, amely alkalmas akár kisebb, akár nagyobb méretű adathalmazok kezelésére.

4. Tervezés

4.1 Use-case diagram

A használati modell a rendszer funkcióinak meghatározására szolgál a felhasználó számára. A rendszert mind a felhasználó, mind az ügyfél szemszögéből reprezentálja, és három fő elemből áll: szereplőkből, használati esetekből és ezek kapcsolataiból [5].

- Aktor: az a személy, aki magát a rendszert használja.
- Használati eset: a rendszer egy adott funkciója. Leírásának teljesnek és konzisztensnek kell lennie, tehát az összes funkciót tartalmazza, illetve egyik eset se álljon ellentmondásban egy másikkal.
- Kapcsolat: egy aktor és egy használati eset között fennálló viszony

Használati esetek

1. **Bejelentkezés**: a helyes felhasználónév és jelszó páros megadása után a rendszer érvényesíti a bejelentkezést, és átirányítja a felhasználót a menübe.
2. **Számla tétel készítés**: ügyfelek számára elérhető mely leírást add bizonylatként
3. **Termék készletezés**: alkalmazottak számára pedig az elérhető és nem elérhető termékek listázása, illetve abban a keresésében.
4. **Alkalmazottak kezelése**: listázni tudják az összes alkalmazottat, illetve megváltoztatni a hozzájuk tartozó adatokat vagy akár törölni tudjuk
5. **Termék adminisztráció**: listázni tudjuk az összes terméket, illetve hozzáadni, illetve vagy módosítani vagy akár törölni.
6. **Bizonylat készítés:** Ki tudjuk választani, hogy terméket veszünk be vagy terméket adunk el és azoknak a mennyiséget és árát meg tudjuk határozni
7. **Statisztika készítés**: különböző statisztikákat ábrázoló diagrammokat tanulmányozhat.
8. **Kijelentkezés***:* miután a felhasználó bejelentkezett az alkalmazásba, lehetősége van a program bármelyik részében a kijelentkezésre.

6. ábra Use-case diagram

4.2 ER (Entity-Relationship) modell

Az entitás -kapcsolat modell leegyszerűsíti az alkalmazás adatbázis -struktúrájának ábrázolását egy diagramon keresztül. Három fő összetevőből áll:

- **egyed**: Az egyed lehet gyenge, ami azt jelenti, hogy önmagában nem azonosítható, és csak egy másik egyénnel való kapcsolatán keresztül ismerhető fel, vagy normális, ami azt jelenti, hogy önmagában azonosítható.
- **tulajdonság**: Egyszerű (egyetlen értékkel rendelkezik), elsődleges kulcs (az azonosítás érdekében az elsődleges kulcsokat aláhúzással, az idegen kulcsokat pedig szaggatott vonallal jelölik), összetett (egynél több részből állhat), többszörös (több értéket is tartalmazhat), számított (az érték nem közvetlenül adott, hanem származtatható).
- **kapcsolat**: 1:1 (egy- az-egyhez kapcsolat, egy entitás pontosan egy másik entitáshoz kapcsolódik), 1: N (egy- többhöz kapcsolat, egy entitás több más entitáshoz is kapcsolódhat), N:M (sok - többhöz kapcsolat, több entitás több más entitáshoz is kapcsolódhat).

A mellékelt ábrán látható, hogy a modell tíz egyedet tartalmaz: Az egyedek között fentálló kapcsolatok: bizonylat tétel, bizonylat fej, bizonylat kód, számla fej, ügyfel, mennyiség egység, számla tétel, forgalmi adó, termék, készlet:

- **termék-készlet** (egy - több): A "termékek-készlet" kapcsolat általában azt jelenti, hogy van egy vagy több termék, amelyeket egy készletben tartanak.
- **termék-számla tétel** (egy - több): A "termék-számla tétel" kapcsolat alapján egy termék többször is előfordulhat különböző számlákon, vagy akár ugyanazon a számlán több különböző tételként. Egy vállalat számla kezelési rendszerében minden számlának több tétele lehet, és minden számla tétel egy adott terméket jelöl. A "termék-számla tétel" kapcsolat alapján egy termék többször is előfordulhat különböző számlákon, vagy akár ugyanazon a számlán több különböző tételként.
- **termék-forgalmi adó** (több - egy): A "termék-forgalmi adó" kapcsolat azt jelenti, hogy több termék is hozzárendelhető ugyanahhoz a forgalmi adóhoz. Egy adott országban lehet, hogy van egy általános forgalmi adókulcs, amelyet számos különböző termékre alkalmaznak az összes értékesítési tranzakcióban. Például egy 20%-os áfa, amely minden árut érint. Ebben az esetben minden termékhez hozzá lehet rendelni ezt a 20%-os áfakulcsot.

- **termék-bizonylat tétel** (több - egy): A "termék-bizonylat tétel" kapcsolat azt jelenti, hogy több termék is hozzárendelhető ugyanahhoz a bizonylattételhez. Egy számla esetében lehet, hogy több különböző termék szerepel, és mindegyik termékhez tartozik egy-egy tétel a számlán. Tehát egy számla lehet, hogy több termékhez is kapcsolódik, és minden egyes termékhez tartozik egy tétel a számlán.

- **bizonylat tétel-bizonylat fej** (több – egy): A "bizonylat tétel-bizonylat fej" kapcsolat azt jelenti, hogy több bizonylattétel tartozik egy bizonylatfejhez. Egy számlázó rendszerben egy számlafej tartalmazza az összes általános információt a számláról (például számlaszám, dátum, ügyfél stb.), míg a számlatételek részletezik a tényleges tételösszegeket (például terméknevek, mennyiség, árak stb.). Tehát több számlatétel kapcsolódik egyetlen számlafejhez.

- **számla tétel-számlafej** (egy – több): A "számla tétel-számlafej" kapcsolat azt jelenti, hogy egy számlatételhez egyetlen számlafej tartozik. Egy számlázó rendszerben minden tétel, például termék vagy szolgáltatás, amelyet egy ügyfél megvásárolt, kapcsolódik egy számlafejhez, amely tartalmazza az összes általános információt a tranzakcióról (például számlaszám, dátum, ügyfél stb.).

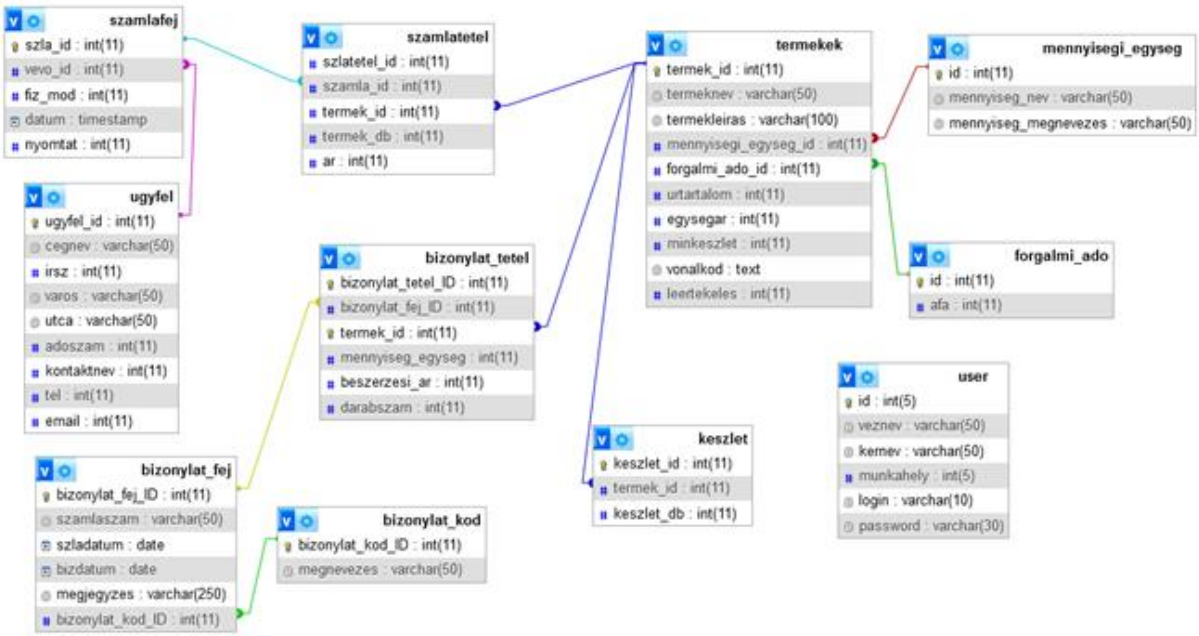
- **számlafej- ügyfél** (több – egy): A "számlafej-ügyfél (több – egy)" kapcsolat azt jelenti, hogy több számlafej is tartozhat egyetlen ügyfélhez. Ez a kapcsolat általában a számlázási és pénzügyi rendszerekben fordul elő, ahol egy ügyfélnek több számlát is kibocsátanak. Egy raktárnak több ügyfele van, és minden ügyfélhez különféle számlák tartozhatnak. Egy adott ügyfél számos terméket vagy szolgáltatást vásárolhat a vállalattól különböző időpontokban, és minden vásárláshoz külön számlát állíthatnak ki. Ez a több az egyhez kapcsolat, ahol a több számlafej hivatkozik egyetlen ügyfélre.
- **termék- mennyiség egység** (több – egy): A "termék - mennyiség egység (több – egy)" kapcsolat azt jelenti, hogy több termék ugyanahhoz a mennyiségi egységhez tartozhat. Tegyük fel, hogy egy élelmiszerboltban különböző termékek vannak, például gyümölcsök, zöldségek és tejtermékek, amelyeket különböző mennyiségi egységekben mérnek vagy adják el. Például a gyümölcsöket és zöldségeket kilogrammban mérik, míg a tejtermékeket literben mérik. Minden termékhez tehát ugyanazok a mennyiségi egységek tartoznak.

- **bizonylat fej-bizonylat kód** (egy – több): A "bizonylat fej-bizonylat kód (egy – több)" kapcsolat azt jelenti, hogy egy bizonylatfejhez több különböző bizonylatkód tartozhat. Egy raktár rendszerben egy bizonylatfej lehet például egy számla vagy egy másik dokumentum, amely összefoglaló információkat tartalmaz (például számlaszám, ügyfél, dátum). Ehhez a fejhez több különböző bizonylatkód tartozhat, amelyek az adott bizonylat különböző részleteit vagy jellemzőit írják le. Ezek a kódok lehetnek olyanok, mint például a tranzakció típusát, a számla státuszát vagy más specifikus jellemzőket meghatározó kódok.

Néhány, fontosnak tartott információ, amit közölni szeretnék a szakdolgozatomban a 1. számú mellékletben illusztráltam.

4.3 Relációs modell

Az entitás-kapcsolat (ER) modell alapján egy relációs modellt is kidolgoztam. Ez a modell egy olyan adatbázisból áll, amely több rekordot tartalmaz, amelyek mindegyike egy-egy személyt képvisel, a rekordokon belüli mezőkben tárolt megfelelő attribútumokkal. A rekordok közötti kapcsolatokat az elsődleges és idegen kulcspárok határozzák meg. Az ER -modellből a relációs modellbe történő átalakítás során az egyéneket rekordokká, az attribútumokat pedig mezőkké alakítjuk át. Az egy-egy és egy- sok kapcsolat esetén idegen kulcsok jönnek létre, míg a sok - sok kapcsolat egy új összekötő rekord és két idegen kulcs létrehozását eredményezi. Lehetőség van az attribútumok nevén kívül az attribútumok adattípusainak megadására is.



7. ábra Relációs modell

4.4 Az adatbázis elkészítése

Az alkalmazás működéséhez szükséges táblák tárolásához adatbázis szerverre van szükség. Két lehetőség közül választhatunk: vagy helyi adatbázist, másnéven localhostot használunk, vagy pedig egy távoli kiszolgálóhoz kapcsolódunk az interneten keresztül. Mindkét megoldásnak megvannak a saját előnyei és hátrányai. A helyi adatbázisok általában nyílt forráskódú, ingyenes licenszű szoftverek, szemben a távoli adatbázisokkal, amelyek általában csak valamilyen díjszabás keretében vehetők igénybe. Fontos különbség még, hogy a távoli adatbázis bármilyen gépről elérhető az internet segítségével, míg a helyi csak az aktuális gépen, azonban internetkapcsolat nélkül is használható. Végül, az adatbázis testreszabhatóságát is figyelembe kell venni, mivel a helyi megoldás teljes felügyelettel rendelkezik az adatbázis felett. Mivel fontosnak tartom az ingyenes használatot és a testreszabhatóságot, a választásom a helyi megoldásra esett. Bár az alkalmazás szempontjából a távoli adatbázis használata célszerű lett volna, de mivel nem éles rendszerről van szó, így nem tartottam indokoltnak a fizetős megoldás alkalmazását. A nyílt forráskódú rendszerek közül a MYSQL nevű, adatbáziskezelő rendszert választottam a megbízhatósága, stabilitása való megfelelése miatt.

Most felvázolom az általam létrehozott táblázatokat és azok létrehozási módszereit:

bizonylat_fej tábla:A "bizonylat_fej_ID" az elsődleges kulcsa, melyhez a "bizonylat_kod_ID" idegen kulcsként kapcsolódik. Ez a kulcs hivatkozik a bizonylat típusára, a számlaszámra (azaz a bejövő számla számára, melyet a beszállító hozott), valamint a szállítólevél kiállítási dátumára. A "bizdatum" a bevételezés vagy kiadás dátuma, míg a "megjegyzes" mezőben lehetőség van rögzíteni az átvételkor vagy kiadáskor felmerülő megjegyzéseket. A "bizonylat_kod_ID" pedig tartalmazza magának a bizonylat típusát (például bevételezési, kiadási, selejtezési, hiányzó stb.)

1) CREATE TABLE `bizonylat_fej`
   (bizonylat_fej_ID` int (11) PRIMARY KEY,
   szamlaszam` varchar (50) NOT NULL,
   szladatum` date NOT NULL,
   bizdatum` date NOT NULL,

```
megjegyzes` varchar (250) NOT NULL,
bizonylat_kod_ID ` int (11) NOT NULL
FOREIGN KEY (bizonylat_kod_ID) REFERENCES
bizonylat_kod (bizonylat_kod_ID))
ENGINE=InnoDB DEFAULT CHARSET=utf8mb4
COLLATE=utf8mb4_general_ci;
```

bizonylat_kod tábla: A "bizonylat_kod_ID" az elsődleges kulcs, mely a bizonylat típusát jelöli (például bevételezési, kiadási, selejtezési stb.). A "megnevezes" mezőben pedig részletes információk találhatók a bizonylat típusáról és jellemzőiről. Ez a kód meghatározza, hogy a bizonylat milyen típusú (például bevételezési, kiadási stb.), és a "megnevezes" mező segítségével tájékoztatást ad a bizonylat típusának jellemzőiről és funkcióiról.

2)
```
CREATE TABLE `bizonylat_kod`
(`bizonylat_kod_ID` int (11) PRIMARY KEY,
`megnevezes` varchar (50) NOT NULL
 ENGINE=InnoDB DEFAULT CHARSET=utf8mb4
COLLATE=utf8mb4_general_ci;);
```

bizonylat_tetel tábla: Az elsődleges kulcs a "bizonylat_tetel_ID", melynek idegen kulcsa a "bizonylat_fej_ID" bizonylat táblához kapcsolódik. Ezen kívül tartalmaz továbbá még egy idegen kulcsot a "termek_ID”, ami termék táblához kapcsolódik. Emellett tartalmazza a "mennyiseg_egyseg" és "beszerzési_ár" at, melyek kifejezik a termék mennyiségét és beszerzési árát. A "darabszám" változó pedig további információt ad a termék darabszámáról.

3)
```
CREATE TABLE `bizonylat_tetel` (
`bizonylat_tetel_ID` int (11) PRIMARY KEY,
`bizonylat_fej_ID` int (11) NOT NULL,
`termek_ID` int (11) NOT NULL,
`mennyiseg_egyseg` int (11) NOT NULL,
`beszerzesi_ar` int (11) NOT NULL,
`darabszam` int (11) NOT NULL,
FOREIGN KEY (bizonylat_fej_ID) REFERENCES
bizonylat_fej (bizonylat_fej_ID),
FOREIGN KEY (termek_ID) REFERENCES
```

```
teremekek (termek_id),
FOREIGN KEY (mennyiseg_egyseg_id) REFERENCES
termekek ((mennyiseg_egyseg_id))
ENGINE=InnoDB DEFAULT CHARSET=utf8mb4
COLLATE=utf8mb4_general_ci;
```

forgalmi_ado tábla: Az elsődleges kulcs az "id", mely egyedileg azonosítja az adott rekordot. A "afa" mező pedig a forgalmi adó mértékét határozza meg. Ezáltal az "afa" mező segítségével meghatározható, hogy az adott rekordhoz milyen mértékű forgalmi adó tartozik.

4)
```
CREATE TABLE `forgalmi_ado` (
    `id` int (11) PRIMARY KEY,
    `afa` int (11) NOT NULL)
    ENGINE=InnoDB DEFAULT CHARSET=utf8mb4
    COLLATE=utf8mb4_general_ci;
```

készlet tábla: A készleten lévő termékek összefoglalója azt mutatja, hogy miből mennyi található a raktárban. Az elsődleges kulcs a "keszlet_id", ami egyedi azonosítót biztosít a készlet táblában. Az idegen kulcs pedig a "termek_id", ami a termék táblából származik és a termékekhez kapcsolódik. A "keszlet_db" mező pedig azt jelzi, hogy a raktárban fizikailag hány darab található az adott termékből. Ezáltal könnyen áttekinthető, hogy mennyi termék van készleten.

5)
```
CREATE TABLE `keszlet` (
    `keszlet_id` int (11 PRIMARY KEY,
    `termek_id` int (11) NOT NULL,
    ` keszlet_db ` int (11) NOT NULL,
    FOREIGN KEY (termek_id) REFERENCES
    termekek (termek_id))
    ENGINE=InnoDB DEFAULT CHARSET=utf8mb4
    COLLATE=utf8mb4_general_ci;
```

mennyisegi_egyseg tábla: Az adott termék kiszerelésének egységtartalmát tartalmazza, például a zsömlék száma vagy az üdítők űrtartalma alapján. Az "elsődleges kulcs" az "id" mező, mely egyedi azonosítót jelöl a "mennyisegi_egyseg" táblában. A "mennyiseg_nev" mező az SI (International System of Units) mértékegységrendszer használatát tükrözi, míg a

"mennyiseg_megnevezes" a kiválasztott mennyiség közérthető megnevezését tartalmazza. Ezáltal lehetőség van a termék kiszerelésének pontos és könnyen érthető leírására.

6) CREATE TABLE `mennyisegi_egyseg` (
`id` int (11) PRIMARY KEY,
`mennyiseg_nev` varchar (50) NOT NULL,
`mennyiseg_megnevezes` varchar (50) NOT NULL
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4
COLLATE=utf8mb4_general_ci;

szamlafej tábla: Tartalmazza a kiállításra kerülő számla adóügyi bizonylatát, melynek "elsődleges kulcsa" a "szla_id", azaz a számlafej tábla egyedi azonosítója. Az "idegen kulcs" pedig a "vevo_id" a vevő táblából, mely a vásárlók adatait tárolja. Továbbá ide tartozik a "fiz_mod" idegen kulcs is, mely a számla összegének kiegyenlítésére vonatkozó információkat tartalmazza, például készpénz esetén. A "nyomtat" mező visszajelzi a sikeres nyomtatást, míg a "datum" az adott számla kiállításának dátumát rögzíti. Ezáltal teljes körű információt nyújt a kiállított számlákról és azok adóügyi hátteréről.

7) CREATE TABLE `szamlafej` (
`szla_id` int (11) PRIMARY KEY,
`vevo_id` int (11) NOT NULL,
`fiz_mod` int (11) NOT NULL,
`datum` timestamp NOT NULL DEFAULT current_timestamp () ON
UPDATE current_timestamp (),
`nyomtat` int (11) NOT NULL,
FOREIGN KEY (vevo_id) REFERENCES
szamlafej (vevo_id))
ENGINE=InnoDB DEFAULT CHARSET=utf8mb4
COLLATE=utf8mb4_general_ci;

szamlatetel tábla: A számlafej táblához tartozó számlatételek részletezik a számla tételeit. Az "elsődleges kulcs" az "szlatetel_id", ami egyedi azonosítót jelöl a "szamlatetel" táblában. Az "idegen kulcs" pedig a "szamla_id", ami a számlatételhez tartozó számlafej táblához van kapcsolva az elsődleges azonosítóját tárolja. Ezenkívül tartalmaz egy másik "idegen kulcsot", a "termek_id"-t, ami a Termék táblához van kapcsolva. A "termek_db" pedig az értékesített

termék darabszámát fejezi ki. Így a számlatétel részletes információkat nyújt a számla tételéről és a hozzá kapcsolódó termékekről.

8) CREATE TABLE `szamlatetel` (
`szlatetel_id` int (11) PRIMARY KEY,
`szamla_id` int (11) NOT NULL,
`termek_id` int (11) NOT NULL,
`termek_db` int (11) NOT NULL
FOREIGN KEY (szamla_id) REFERENCES
szamlafej (szla_id),
FOREIGN KEY (termek_id) REFERENCES
termekek (termek_id))
ENGINE=InnoDB DEFAULT CHARSET=utf8mb4
COLLATE=utf8mb4_general_ci;

termekek tábla: A termékek adatait rögzíti, melynek elsődleges kulcsa a "termek_id", hiszen ez egyedi minden egyes termék esetében. Idegen kulcsként tartalmazza a "forgalmi_ado_id"-t, ami azt mutatja meg, hogy mekkora forgalmi adót kívánnak kiszabni az adott termékre. Ezenkívül egy másik idegen kulcs is szerepel, a "mennyisegi_egyseg_id", mely az egység mennyiségének tábla egyedi azonosítójára mutat rá. A "termeknev" mező az adott termék nevét tárolja, míg a "termekleiras" részletes leírást ad az adott termékről. Az "urtartalom" információt szolgáltat az adott termék mennyiségi paraméteréről, míg az "egysegar" a termék eladási egységárát tartalmazza. A "minkeszlet" azt jelzi, hogy mikor éri el a termék minimális készletét, és jelezheti annak elérését is. A "vonalkod" mező az adott terméken található vonalkódot tárolja, míg a "leertekeles" mező az időszakhoz kapcsolódó árcsökkentést tartalmazza. Ezen adatok révén részletes információt nyújt a termékekkel kapcsolatos készletkezelésről, árazásról és egyéb fontos paraméterekről.

9) CREATE TABLE `termekek` (
`termek_id` int (11) PRIMARY KEY,
`termeknev` varchar (50) NOT NULL,
`termekleiras` varchar (100) NOT NULL,
`mennyisegi_egyseg_id` int (11) NOT NULL,
`forgalmi_ado_id` int (11) NOT NULL,
`urtartalom` int (11) NOT NULL,

```
`egysegar` int (11) NOT NULL,
`minkeszlet` int (11) NOT NULL,
`vonalkod` text NOT NULL,
`leertekeles` int (11) NOT NULL
FOREIGN KEY (mennyiseg_egyseg_id) REFERENCES
mennyisegi_egyseg ((id),
FOREIGN KEY (forgalmi_ado_id) REFERENCES
forgalmi_ado ((id))
ENGINE=InnoDB DEFAULT CHARSET=utf8mb4
COLLATE=utf8mb4_general_ci;
```

ugyfel tábla: Az ügyfelek adatait tárolja, melynek elsődleges kulcsa az "ugyfel_id". Ez az azonosító a regisztrációkor megadott felhasználónevet jelöli, hiszen minden esetben egyedinek kell lennie. Ezáltal biztosítja, hogy minden ügyfélnek egyedi azonosítója legyen a rendszerben. Az adatok között szerepel a cégnev, az irányítószám, a város, az utca, az adószám, a kapcsolattartó neve, a telefonszám és az e-mail cím. Ezek az adatok lehetővé teszik az ügyfelek részletes nyilvántartását és kapcsolattartását, valamint segítenek az üzleti folyamatok zökkenőmentes lebonyolításában.

10)
```
CREATE TABLE `ugyfel` (
`ugyfel_id` int (11) PRIMARY KEY,
`cegnev` varchar (50) NOT NULL,
`irsz` int (11) NOT NULL,
`varos` varchar (50) NOT NULL,
`utca` varchar (50) NOT NULL,
`adoszam` int (11) NOT NULL,
`kontaktnev` int (11) NOT NULL,
`tel` int (11) NOT NULL,
`email` int (11) NOT NULL
FOREIGN KEY (uygfel_id) REFERENCES
szamlafej ((vevo_id))
ENGINE=InnoDB DEFAULT CHARSET=utf8mb4
COLLATE=utf8mb4_general_ci;
```

User tábla: Engedélyezi a programhoz való hozzáférést. Az "elsődleges kulcs" az "id", mely egyedi azonosítót jelöl, és lehetővé teszi a felhasználók egyedi azonosítását a rendszerben. Ezáltal biztosítja a felhasználók számára a megfelelő jogosultságokkal való hozzáférést és azonosítást a programban. A többi mező tartalmazza a vezetéknév, keresztnév, munkahely, bejelentkezési név (login), valamint a jelszó adatait. Ezek az információk biztosítják a felhasználók hitelesítését és a program biztonságos elérését.

11) CREATE TABLE `user` (
`id` int (5) PRIMARY KEY,
`veznev` varchar (50) NOT NULL,
`kernev` varchar (50) NOT NULL,
`munkahely` int (5) NOT NULL,
`login` varchar (10) DEFAULT NULL,
`password` varchar (30) DEFAULT NULL
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4
COLLATE=utf8mb4_general_ci;

Feltételek, amelyeket a táblák létrehozásakor használtam:

- PRIMARY KEY (elsődleges kulcs): értéke egyedi, célja egy tábla egyértelmű azonosítása.
- FOREIGN KEY (idegen kulcs): mutató egy másik tábla elsődleges kulcsára. Formája: foreign key (mező_név) references tábla_név (mező_név)
- NOT NULL: egy mező ne lehessen null, azaz kötelező legyen értéket adni.

A táblák létrehozása során fontos figyelembe venni, hogy azokat, amelyek tartalmaznak idegen kulcsokat, csak azután hozzuk létre, miután az idegen kulcsra mutató elsődleges kulcsot tartalmazó táblát elkészítettük. A táblák sorrendisége nem számít azok létrehozásakor, ha azokban nem található idegen kulcs. Ha a törlés gombra kattintunk, az összes tábla és annak tartalma törlődik. Ha a tesztadatok ki vannak jelölve a létrehozás előtt, akkor a táblák létrejötte után az alkalmazás különböző rekordokat fog beszúrni minden táblába.

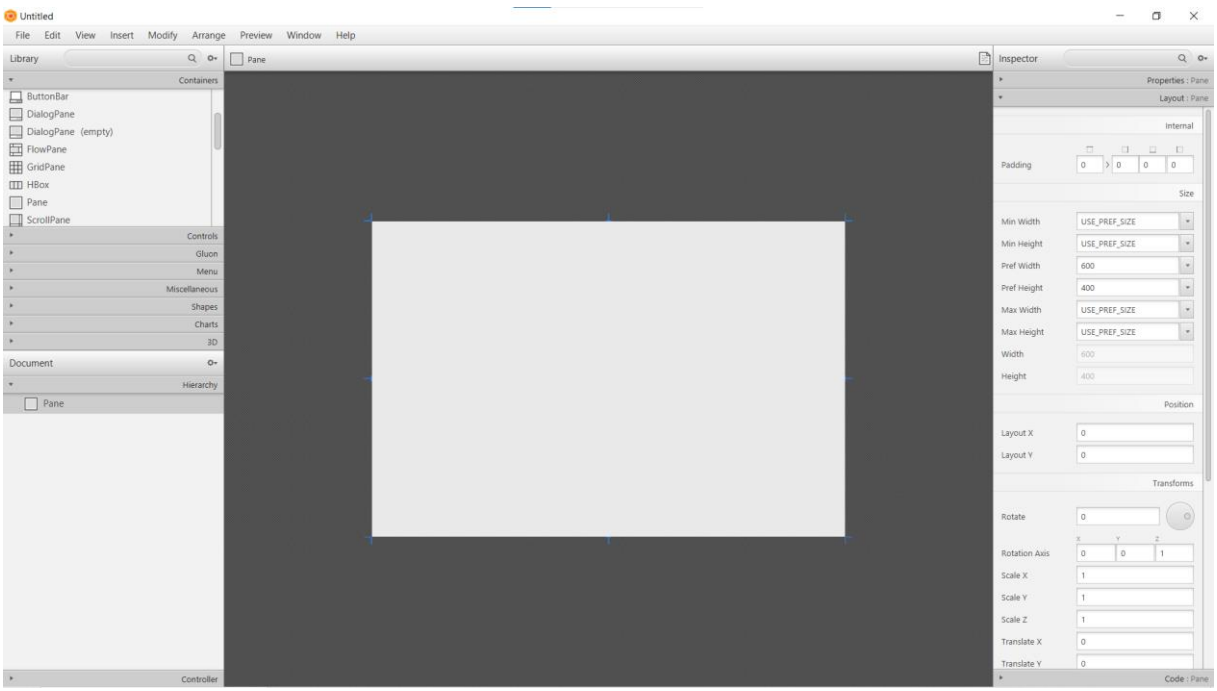
5. Fejlesztőeszköz

Az alkalmazást Java nyelven készítettem, ezért a futtatásához szükséges a Java futtatókörnyezet. Ha csak futtatni szeretnénk az alkalmazást, elég a JRE (Java Runtime Environment), de a fejlesztéséhez a JDK-ra (Java Development Kit) van szükség.

A program elkészítéséhez több Java fejlesztőkörnyezet közül választhattam, például az Eclipse, a NetBeans vagy az IntelliJ IDEA. Az Eclipse és a NetBeans ingyenes licenceléssel rendelkezik, azonban én az IntelliJ IDEA-t választottam, mivel tanulói minőségem fennállása esetén egy éves ingyenes használatot biztosít, és hatékony, átlátható fejlesztést tesz lehetővé számomra. Emellett három további funkcióját szeretném kiemelni: alapértelmezetten támogatja a JavaFX alkalmazások fejlesztését, autocomplete funkcióval rendelkezik (automatikusan kiegészíti a gépelést, ezáltal felgyorsítva és hatékonyabbá téve a fejlesztést), valamint automatikusan menti a változtatásokat.

A grafikus felület elkészítéséhez a JavaFX-hez tartozó SceneBuilder-t használtam, amelyet a következő címről lehet letölteni:

https://gluonhq.com/products/scene-builder/



8. ábra: Grafikus felület tervezése scene builder segítségével

Tanulmányaim során részt vettem választható tárgyon, amelynek címe " Java programozás ". Ezen a kurzuson ismertem meg a Java Swing fejlesztőeszközt, amely lehetővé tette, hogy az objektumorientált programozási ismereteim alapján grafikus felületeket hozzak létre Java programokhoz. A kurzus során megszereztem a grafikus felületek létrehozásához szükséges alapismereteket**.** A kurzus befejeztével szerettem volna mélyebben elmerülni a grafikus felületek világában, de mivel a Java Swing már elavult technológiának számít, egy modernebb eszköz után néztem. Végül a JavaFX-et választottam, mivel ez a Swing modernebb változata, és a fejlesztők is megerősítették, hogy a JavaFX a jövőbeli szabvány. A JavaFX-ben a felhasználói felületek elemeit egy külön FXML fájl írja le, míg a felületek formázásához a Scene Builder programot használhatjuk. Ez a technológia lehetővé teszi a fejlettebb, modern grafikus felületek létrehozását a Java alkalmazásokhoz.

A fejlesztés során igyekeztem az alkalmazást az MVC (Model-View-Controller) architektúra szerint felépíteni. Ez a szerkezeti minta az adatok kezelését (model) és a felhasználói felületet (view) különválasztja, biztosítva ezzel, hogy a megjelenési logika és az üzleti logika egymástól függetlenül működjön. Az összekapcsolásról a vezérlő, vagyis a controller gondoskodik. Az alkalmazásomban az MVC struktúrát követve a három fő komponenshez kapcsolódó nevű csomagokat hoztam létre. A "model" csomag az adattagokat, valamint a getter és setter metódusokat tartalmazza. A "view" csomagban az FXML fájlok találhatók, amelyek a felhasználói felületet definiálják. A "controller" csomag pedig azokat az osztályokat tartalmazza, amelyek az Initializable interfész alapján vannak megvalósítva, mivel az FXML fájlok vezérlőjeként csak ilyen osztályokat lehet megadni.

Az alkalmazás elindításához szükség van egy olyan osztályra, amely az Application osztályból származik le, ez lesz az alkalmazás belépési pontja. A JavaFX használatának másik oka a Scene Builder beépített CSS-támogatása. Minden grafikus elemhez rendelhető külön CSS fájl, amelynek segítségével könnyen és hatékonyan lehet kialakítani a grafikus felületet. A CSS (Cascading Style Sheets) segítségével beállíthatók az egyes elemek megjelenési tulajdonságai, például a háttérszín, a szegélyvastagság stb. Ezáltal a grafikus felület személyre szabható és könnyen változtatható.

JDBC: Az alkalmazás adatbázissal való összekapcsolásához szükség van egy JDBC (Java Database Connectivity) nevű API-ra, amely meghatározza a felhasználó adatbázishoz történő kapcsolódását. A JDBC driver használatához a projekt függőségei közé kell importálni. Az API három fő szolgáltatást kínál: adatbázis-kapcsolat kiépítése, SQL utasítások végrehajtása, valamint a lekérdezések eredményeinek feldolgozása. Mivel az alkalmazás működése főként a felhasználó és az adatbázis közötti kommunikáción alapul, az adatbázis-kapcsolat gyakran használt elem lesz.

```
#mysql DB properties
MYSQL_DB_DRIVER_CLASS=com.mysql.cj.jdbc.Driver
MYSQL_DB_URL=jdbc:mysql://localhost:3306/bolttteszt
MYSQL_DB_USERNAME=Bence
MYSQL_DB_PASSWORD=B
```

9. ábra Adatbázis kapcsolódás

IdeaProjects\belep\src\main\resources\db.properties állomány útvonala. A képen látható program egy MySQL adatbázishoz való kapcsolódás beállításait konfigurálja. Meghatározza a driver osztályt, az URL-t, a felhasználónevet és a jelszót, amelyek szükségesek a kapcsolat létrehozásához. Az adatbázis kapcsolat létrehozásához az első sorban regisztráljuk a JDBC driver-t a programban. Második sorban az adatbázis lokális elérhetőségét adjuk meg, harmadik és negyedik sor az adatbázishoz való csatlakozáshoz szükséges felhasználónév és jelszót jelenítjük meg.

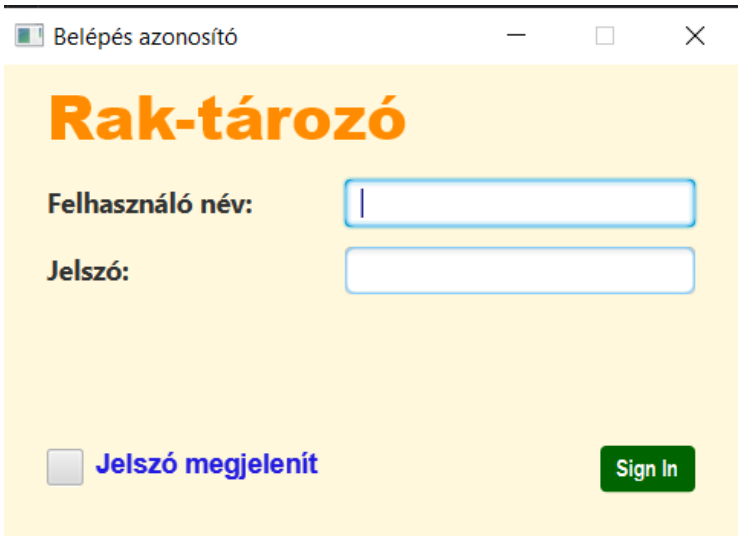
A MyDataSourceFactory osztály getMySQLDataSorce() függvénye használja a fentiekben említett konfigurációs fájlt, amely függvény visszatérési értéke a Datasorce típusú adatforrás. Ehhez kapcsolódóan a DbHandler osztályban a getConnection() függvény a bejövő adatforrás segítségével létrehozza az adatbázis kapcsolatot és Connection típussal tér vissza. A DbHandler osztályban implementáltam még a nagyon fontos adatkapcsolat lezárást és objektum lezárásokat.

```
public static Connection getConnection() throws SQLException {
DataSource dataSource = MyDataSourceFactory.getMySQLDataSource();
Connection conn = dataSource.getConnection();
// conn.setAutoCommit(false);
return conn;
}

public static void close(Connection conn) throws SQLException {
if (conn != null) {
conn.close();
}
}
```

6. Fejlesztői dokumentáció

6.1 Bejelentkezés



10. ábra Bejelentkező felület

A program inditása a HelloApplication osztáyban talalhato start() metudos a java FX kotelezö eleme és ez meghivja HelloApplicationlogin osztályt. Ebben az osztályban a login() metódus létrehoz egy új Stage elemet, amely az FXMLLoader osztály segítségével beállítja a hello-view.fxml fájlt. A jelszó megjelenítés opcióra is van lehetőség, ami a burkolt karaktereket átváltja látható karakterekre. A felhasználó a bejelentkezési adatainak megadásával képes belépni a menürendszerébe a bejelentkezési felületen keresztül. A SubmitButtonAction() metódus aktiválódik, amikor a bejelentkezés gombot megnyomják.

Program elindításához szükséges egy belépési pont, amelyet a HelloApplication java fájlban található. A belépési pontnak van egy vezérlője, amely a HelloController.java fájlban van ez a vezérlő kezeli a felhasználói felületen történő eseményeket például a már említett gombot figyeli. A felhasználói felületet a hello-view.fxml fájlból vesszük. A bejelentkező ablakon ellenőrizzük a loginnév és a jelszó megadását és az adatok érvényességét, amit egy adatbázisban tárolt értékkel hasonlítunk össze. A HelloController osztályban a SubmitButtonAction () metódussal ellenőrizzük a felhasználó név és jelszó párt még hozzá adatbázishoz kapcsolódunk és azzal az adatbázissal méghozzá a User táblához kapcsolódva ellenőrizzük a megadott adatok helyességét. A helytelenül megadott akár felhasználónév jelszó esetén hibaüzenetet adunk megfelelő érték párok használatával pedig rögtön a főmenübe jutunk.

6.2 Főmenü

11. ábra Menü részleg

A HelloController osztály submitButtonAction metódusa készít egy új Stage elemet, ez az FXMLLoader osztálynak állítja be a fomenu.fxml fájlt. A főmenű ablak bezárásának gombjára beállítjuk a Platform.exit() metódust. Az ablak példányosítása után, megjelenítése előtt beállításra kerülnek az ablak paraméterei. Ezeknek a paraméterek meghatározásához szükségünk van az adott kijelző fizikai felbontására. A kijelző felbontása ábrázolható oszlopok és sorok szorzatával adható meg. Erre a célfeladatra optimalizálva készítettem el a Controller osztályban található screen() és screenResolution() függvényeket. Ez utóbbi függvény visszatérési értéke egy Double típusú List. Ebben a Listben az aktuális képernyő szélességét és magasságát adjuk vissza.

```
4 usages
public List<Double> screenResolution(String s){
    // A képernyőfelbontás megállapítása a screen() függvénny visszatérési eredményével ami itt bemenő érték
    List<Double> list = new ArrayList<>();
    String [] tomb = s.split( regex: "[ ,=]");   //Szeleteljük a függvény eredményét
    double szelesseg = Double.parseDouble(tomb[14].toString()); //A felszeletelt eredmény String tömbből kiolvasása és Double típusúvá alakítva
    String [] temp = tomb[17].toString().split( regex: "}");    //Egy partizán kapocs eltávolítása
    double magassag = Double.parseDouble(temp[0].toString());
    list.add(szelesseg);
    list.add(magassag);
    return list;
}
```

12. ábra képernyő felbontás megállapítása

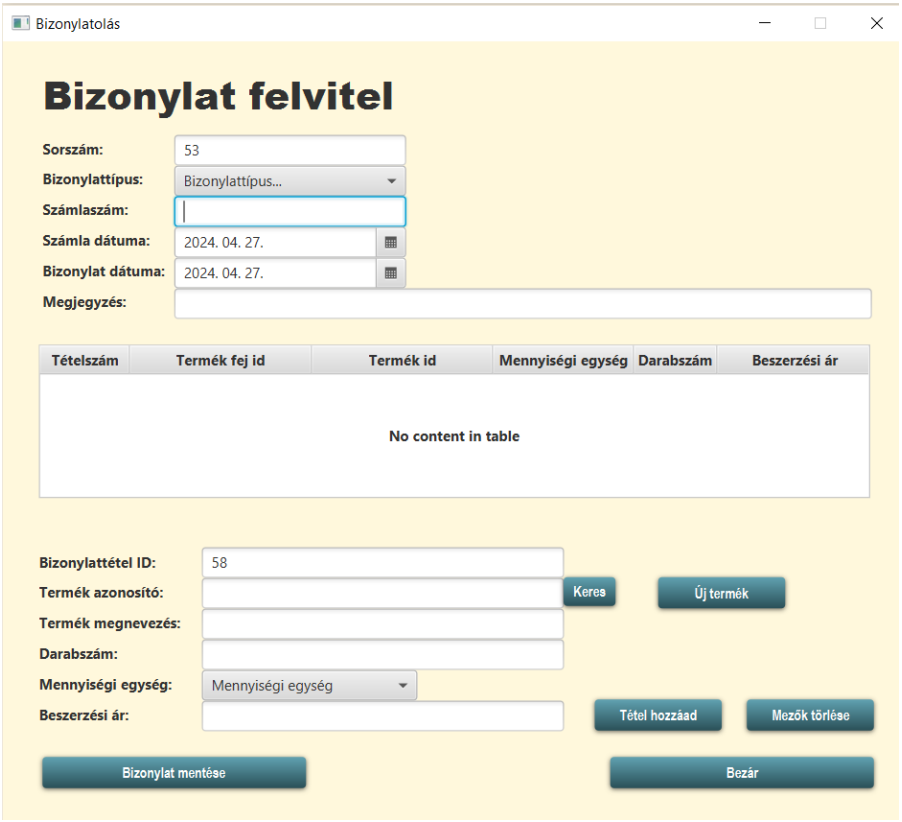
A HelloController osztályból megnyitni készülő főmenü ablak paraméterezése.

```
// A képernyő felbontásának megállapítás a Controller osztályból a screen() függvénnyel majd annak értékét használja screenResolution() függvény
stage1.setScene(new Scene(root1, ctr.screenResolution(ctr.screen().toString()).get(0),  height: ctr.screenResolution(ctr.screen().toString()).get(1)-50));
stage1.show();
```

13. ábra Ablak méretezés

6.3 Feldogozások

Bizonylat felvitel:



14. ábra Bizonylat felvitel ablak

A feldolgozások/bizonylatolás menüpontra kattintva a program új ablakot nyit, amely stage-hez bizonylat fxml fájlt állít be. A bizonylatolás ablakban a raktárkészlet manipulálására alkalmas bizonylatokat készíthetünk. A bizonylat típus beállításához jelenleg két értéket választhatunk (bevétel, kiadási). A bizonylat típus kiválasztásával a raktár készlet növelése vagy csökkentése valósítható meg. A bizonylat sorszáma egyedi, nem módosítható. A számla szám rovatban az árukísérő okmány azonosító száma kerül. A számla dátuma a számla készítési dátuma. A bizonylat dátum pedig a rögzítés napjának dátumát tartalmazza. A megjegyzés rovatban a bizonylatolás közben felmerülő problémák rögzítésére használt rovat. A bizonylat dátum kiválasztásának elősegítésére datepicker() eljárást választottam amely a felugró ablakból választható dátummezőt biztosít. Alapértelmezés szerint az aktuális dátumot hozza.

```
@FXML private TableView<BizonylatTetel> bizonylat_tetel_table;
```

15. ábra Bizonylat tételek megjelenítése

Bizonylat tétel megjelenítését TableView típus segítségével látjuk el. A tábla feltöltése a getBizonylatTetelList() metódussal történik

```
bizonylat_tetel_table.setItems(getBizonylatTetelList());
```

16. ábra tábla feltöltése

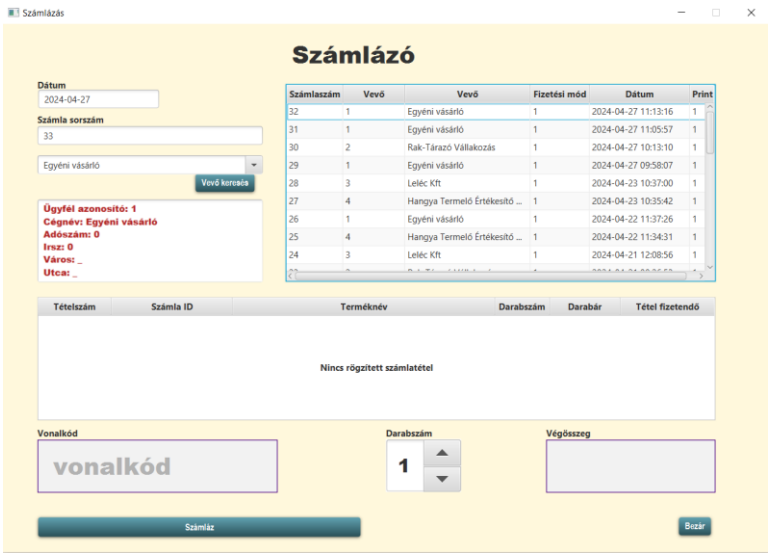
A bizonylattétel ID egyedi típus, amit nem lehet megváltoztatni, ezt a bizonylat_tetel táblából nyerjük ki. A Termék azonosító a kívánt termék vonalkódjára szolgáló beviteli mező. A keres gombbal a ButtonTetelKeres() metódust hívjuk, mely a termékek táblában végez keresést.

```
@FXML protected void ButtonTetelKeres(){
    Boolean ok;
    ok = false;
    for (Termek t:dbm.getTermekListazo()) {
        if (t.getVonalkod().equals(tftermek_ID.getText())){
            tftermek_megnevezes.setText(t.getTermeknev());
            for (MennyisegiEgyseg m : dbm.mennyisegListazo) {
                if (m.getId() == t.getMennyisegi_egyseg_id()) {
                    cbmennyiseg_egyseg.setValue(m.getGetMennyiseg_megnevezes());   //a mmennyisegi_egyseg_id kikeresése
                }
            }
            ok = true;
            cikkOk = true;
        }
    }
    if (ok == false){
        ctr.SM(ctr.smType[0], tytle: "Nemlétező cikkszám! \n", header: "vonalkód");
        tftermek_ID.setStyle(piros);
    }else {
        tftermek_ID.setStyle(zold);
    }
}
```

17. ábra Termék keresés

Találat esetén a kereső mező alatti beviteli mezőket kitölti az adott termékek adatbázisban tárolt tulajdonságaival. Ha még nem rögzített termék esetén hibaüzenetet dob és a keresőmező keretét pirosra színezi. Ebben az esetben az újtermék gombbal új ablakot nyitunk, amely stage-hez a termek_lista.fxml fájlt állítja be. A termék megnevezése és mennyiségi egység mindkettő adatbázisból kerül feltöltésre. A darabszám és a beszerzési ár az áru kísérő okmányokról kerülnek átvételre és rögzítésre.

Számlázó:



18. ábra Számlázó ablak

A számlázó osztályban az initialize() metódusban több adatbázis lekérdezést is elindítunk amivel feltöltjük a tevékenységhez szükséges listáinkat (szamlafej, szamlatetel, ugyfelList…):

```
@Override
public void initialize(URL url, ResourceBundle resourceBundle) {
    dbm.szamlaFejlList( query_: "select * from szamlafej");
    dbm.szamlaFejlList_fejview( query_: "SELECT sz.szla_id, sz.vevo_id, u.cegnev, sz.fiz_mod,
    dbm.termekTableList( query_: "select * from termekek");
    dbm.afaTableList( query_: "select * from forgalmi_ado");
    dbm.ugyfelList( query_: "select * from ugyfel");
```

19. ábra initialize() metódus

A feldolgozások/számlázó menüpontra kattintva a program új ablakot nyit, amely stage-hez szamlazo.fxml fájlt állít be. A számlázó menüpontban a raktárkészlet erejéig készíthetünk számlabizonylatokat. A számlázó felületen az aktuális dátum, a következő számla sorszáma, és eddig számlázott számlák fejrészei jelennek meg. Alapértelmezésként a vásárló egyéni vásárlóként kerül beállításra mivel a cég a profijához kapcsolódóan nagy valószínűséggel magánszemélyek kerülnek kiszolgálásra. A vevő kereső mezőben alacsony ügyfélkörnél Comboboxból választunk vagy a teljes ügyfél név megadása esetén a keres gomb ad találatot.

```
private int vevoCombo(){
    String cmbvalue = "";
    int uID = 0;
    cmbvalue = (String) cmbugyfel.getValue();
    for (Ugyfel ul: dbm.getugyfelListazo()) {
        if(cmbvalue.equals(ul.getCegnev())){
            txtugyfel.setText(ul.toString());
            uID = ul.getUgyfel_id();
        }
    }
    return uID;
}
```

20. ábra vevő kereső

A vonalkód kereső texfield ben figyeljük a setOnKeyReleased eljárással a billentyűzet felengedést és ha tfvonalkod hosszával megegyezik a konstans vonalkódhossz akkor keresést hajt végre. Érvényes vonalkód és megfelelő mennyiségű rendelkezésre álló készlet esetén a cikk bekerül a számlatételek közé. A számláz nyomógomb ellenőrzést végez az ablakon. Tétel híján esetén tétel hiba üzenetet add. Ellenkező esetben a Controller osztályban lévő manipulatePdf4() metódus a számla sorszámmal ellátott PDF állományba elkészíti a számlaképet.

A számla pdf fájl eltárolására az operációs rendszerbe bejelentkezet felhasználó saját könyvtárát (account directory) használjuk fel. Az account név kiderítésére System.getProperty() beépített függvényt használjuk fel. A vissza térési értékét a dest Stringbe illesztve kapjuk a pdf fájl elhelyezési könyvtárát.

```
String useraccount = System.getProperty("user.name");
String dest = "C:/Users/"+useraccount+"/szamla_"+ szamlaF.getFirst().getSzla_id()+".pdf";
```

21. ábra számla pdf tárolási útvonal kiválasztás

A pdf generálása után célszerűnek tűnt, hogy megjelenítsük a számla tartalmát. Windows rendszerben a „rundll32” paranccsal elindítjuk az elkészített számla PDF-fájlt (lásd 22. ábra) ez által az operációs rendszerben alapértelmezett pdf állomány olvasására alkalmas szoftver indul, ami esetünkben lehet még akár az Edge böngésző is.

```
if ((new File(dest)).exists()) {
    Process proc = Runtime
            .getRuntime()
            .exec( command: "rundll32 url.dll,FileProtocolHandler "+dest);
    proc.waitFor();
```

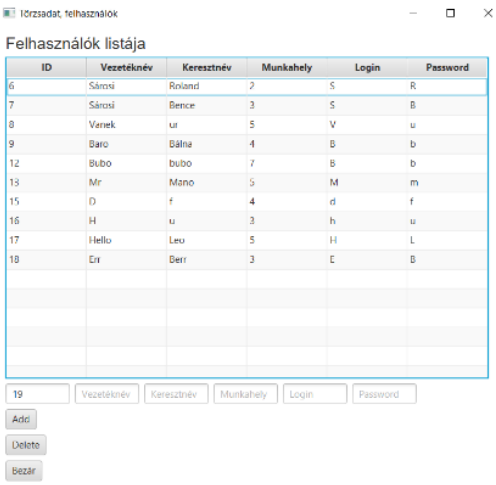
22. ábra

6.4 Törzsadatok

Felhasználók listája:

A programhoz való belépéshez adható felhasználónevek és jelszavak tárolásra szolgáló adatt tábla. A tábla bővíthető újabb felhasználók adataival kitöltöttség ellenőrzésé mellett.

A DbMethods osztályban lévő WriteDB()metódust használjuk a user tábla bővítésére.



23. ábra Felhasználó lista ablak

```
try {
    dbm.WriteDB(addList, insertString: "INSERT INTO User (id, veznev, kernev, munkahely, login, password) VALUES (?, ?, ?, ?, ?, ?)");
} catch (SQLException ex) {
//    throw new RuntimeException(ex);
    ctr.SM(ctr.smType[0], tytle: "Lezárás hiba! \n"+ex, header: "WriteDB");
}}
```

24. ábra user tábla bővítés (insert sql utasítással)

Áfa lista:

A termékek adó tartalmához kapcsolható áfa lista, ami bővíthető és azon belül beviteli ellenőrzést végzünk. Az AfaList osztály meghívásakor ez a initialize() metódus lefut

```
@Override
public void initialize(URL url, ResourceBundle resourceBundle) {
    dbm.afaTableList( query_: "select * from forgalmi_ado");
    id.setCellValueFactory(new PropertyValueFactory<Afa, Integer>("id"));
    afa.setCellValueFactory(new PropertyValueFactory<Afa, Integer>("afa"));
    afatable.setItems(dbm.getAfaListazo());
    afatable.setEditable(true);
}
```

25. ábra A initialize() beltartalma

26. ábra Áfa ablak

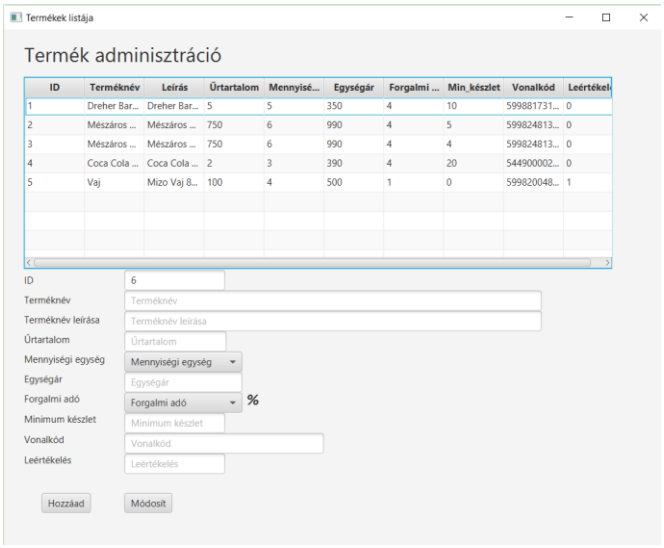
A hatályos magyar adójogszabály előírja a kerekítést. Ez azt jelenti, hogy a számlázási rendszernek képesnek kell lennie a számla összegének kerekítésére a legkisebb törvényes címletre. Ez a kerekítési szabály segít a pénzkezelés egyszerűsítésében és a tranzakciók gyorsaságának növelésében. A kerekítési szabályok betartása elengedhetetlen a jogi megfelelőség érdekében.

A képen látható kód egy példa arra, hogyan lehet implementálni a kerekítési szabályokat a számlázási rendszerben. Ez a kód egy olyan függvényt tartalmaz, amely egy adott számot kerekít bizonyos feltételek szerint.

```
private  double kerekitVegosszeg(double bruttoOsszeg)
{
 if (bruttoOsszeg % 5 ==0)
 {
     return  bruttoOsszeg;
 }
 else {
     return (double)  (Math.round(((double)bruttoOsszeg) / 5) *5);
 }
}
```

27. ábra kerekítésére

Termékek listája:



28. ábra Termék adminisztrációs ablak

A TermekList osztályban megvalósult a forgalmazni kívánt termékek listáját és egyedi tulajdonságait tartalmazó űrlap. A listában található sorok kattintással kijelölhetők és ezen sor adatai módosíthatók a bekérőmezőkben.

```
Termek selectedRow = termek_lista_tableview.getSelectionModel().getSelectedItem();
```

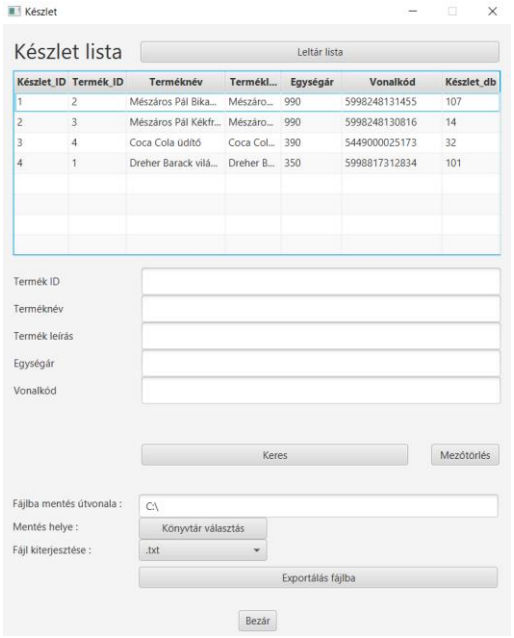
29. ábra TermekList sorszáma

A terméklista bővíthető, új még nem forgalmazott termékekkel. Bizonyos bekérő mezőkhöz nem engedünk önálló adatbevitelt csak választható adattartalmak érhetőek el pl: mennyiség egység, forgalmi adó.

```
for (MennyisegiEgyseg m : dbm.mennyisegListazo) {
    if (m.getId() == selectedRow.getMennyisegi_egyseg_id()) {
        add_mennyiseg_egyseg_id.setValue(m.getGetMennyiseg_megnevezes());   //a mmennyisegi_egyseg_id kikeresése
    }
```

30. ábra mennyiség egység keresése

Készlet lista / leltár:



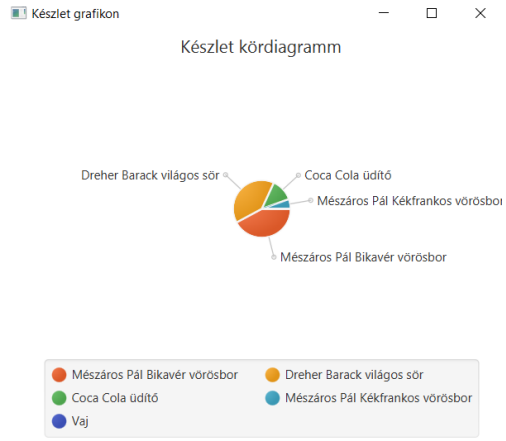
31. ábra Készlet lista / leltár ablak

Az évvégi leltározáshoz vagy évközi készlet ellenőrzéshez kaphatunk fontos információt. Az űrlapon a táblázat feletti gombbal válthatunk a készlet lista és a leltár lista közözött. A táblázat alatt a táblázat oszlopaiban történő keresésre szolgáló bekérő mezők találhatók. Az űrlap alsó szélén a készlet lista mentésére szolgáló kezelő szervek találhatók. A készlet lista mentésére a mentési könyvtárat kiválaszthatjuk. A Controller osztályból a DirectoryKezelo() függvénnyel állapítjuk meg a mentésnek a célhelyét. A kiexportálandó állomány fájlnevét az aktuális dátummal összefűzzük. A mentendő állományokhoz jelenleg két fájlkiterjesztés társulhat : .txt, .csv.

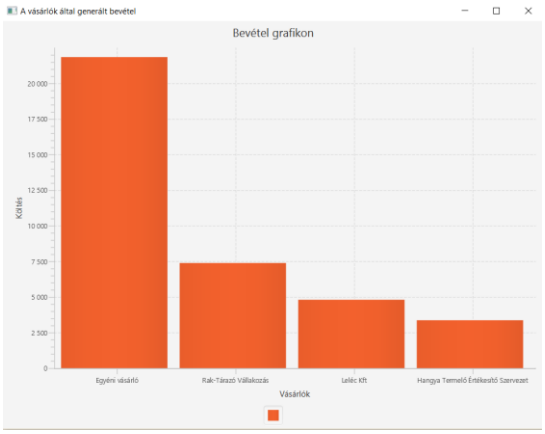
```
directoryChooser.setInitialDirectory(new File(kltl.path));
```

32. ábra fájl ki exportálása

6.5 Statisztika



33. ábra Készlet diagram



34. Bevétel diagram

Statisztikai adatokkal szolgálunk a készlet nyilvántartásával kapcsolatban.

```
pieChartData = dbm.serializedPie( query_: "SELECT t.termeknev, k.keszlet_db, t.termek_id FROM termekek t, keszlet k " +
        "WHERE t.termek_id = k.termek_id ORDER BY keszlet_db DESC;");
```

35. ábra Készlet lekérdezés

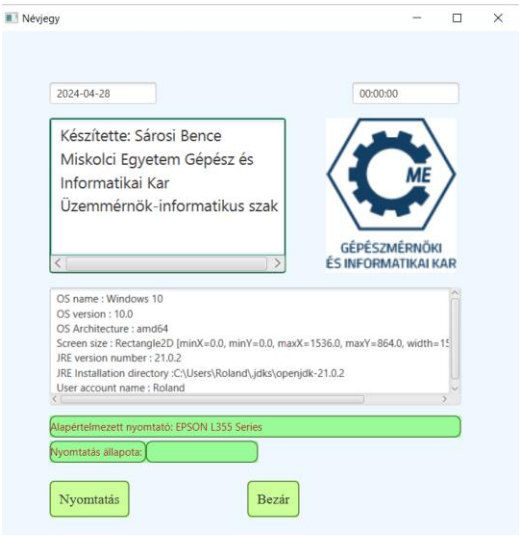
Az ügyfelektől származó bevételek kimutatására szolgáló oszlop diagram és a diagram felépülését szolgáló 3 táblás lekérdezés.

```
bc.getData().add(dbm.serializedBar( query_: "SELECT ugyf.cegnev,SUM(szlat.ar) 'osszar' FROM ugyfel ugyf inner join szamlafej szlafj "
        +"on ugyf.ugyfel_id = szlafj.vevo_id inner join szamlatetel szlat " + "on szlafj.szla_id = szlat.szamla_id inner join termekek trmk "
        + "ON szlat.termek_id = trmk.termek_id " +"GROUP by ugyf.cegnev ORDER BY 'osszar'"));
```

36. ábra Bevétel lekérdezés

6.6 Névjegy

A névjegy a készítő nevét tartalmazza plusz néhány kísérleti ötlet került megvalósításra, melyek közül a képernyőfelbontás és a felhasználónév további felhasználására sor került. Például az óra és a dátum folyamatos frissül, a nyomtatás az egész stage képét nyomtatóra küldi.



37. ábra Névjegy ablak

```
AnimationTimer timer = (now) -> {
    //    time.setText(LocalDateTime.now().format(DateTimeFormatter.ofPattern("yyyy-MM-dd HH:mm:ss")));
        time.setText(LocalDateTime.now().format(DateTimeFormatter.ofPattern("HH:mm:ss")));
        date.setText(LocalDateTime.now().format(DateTimeFormatter.ofPattern("yyyy-MM-dd")));
};
```

38. ábra Dátum és idő

Az operációs rendszer nevét, típusát, verziószámát, a képernyő méretét, felbontását, a java futtatókörnyezet verzióját illetve a felhasználó operációs rendszerbeli belépési nevét jelenítjük meg textfieldben.

```
protected void os_javaDetect(){
    txta_osJava.setText("OS name : " + System.getProperty("os.name")+"\n"+
            "OS version : " + System.getProperty("os.version")+"\n"+
            "OS Architecture : " + System.getProperty("os.arch")+"\n"+
            "Screen size : "+(ctr.screen())+"\n"+
            "Screen resolution : "+(ctr.screenResolution(ctr.screen().toString()).get(0)) +"*"
            +(ctr.screenResolution(ctr.screen().toString()).get(1)) +"\n"+
            "JRE version number : " + System.getProperty("java.version")+"\n"+
            "JRE Installation directory :" + System.getProperty("java.home")+"\n"+
            "OS User account name : " + System.getProperty("user.name"));
}
```

39. ábra Számitógép specfikációi

Külön érdekesség, hogy ki lehet nyomtatni külön az aktuális stage.

```
private void printSetup(Node node, Stage owner)
{
    // Create the PrinterJob
    PrinterJob job = PrinterJob.createPrinterJob();
    if (job == null)
    {
        return;
    }
    // Show the print setup dialog
    boolean proceed = job.showPrintDialog(owner);
    if (proceed)
    {
        print(job, node);
    }
}
```

40. ábra Nyomtatás

7. Összefoglalás

Szakdolgozatom célja egy olyan raktár nyilvántartó asztali alkalmazás elkészítése volt, amely segítségével a raktárban található termékek adatait könnyedén és hatékonyan lehet kezelni. Az alkalmazás lehetővé teszi a termékek adatainak rögzítését, módosítását és törlését, valamint a készleten lévő termékek mennyiségének nyomonkövetését. Ezenkívül a programban implementáltam egy kereső funkciót is, amely segítségével a felhasználók gyorsan megtalálhatják a keresett terméket. Dolgozatomat a funkcionális és nem funkcionális követelmények bemutatásával indítottam, amelyek segítségével világos képet készítettem a programtól elvárt működésről.

A következő fejezetben a tervezési folyamatot ismertettem. Különböző diagramok és modellek definiálása után, ezek segítségével fogtam hozzá az adatbázis tervezéséhez. Itt határoztam meg, milyen adatokat kell tárolni, és hogyan nézzenek ki a tárolásukat szolgáló táblák. Mivel az alkalmazás alapját a jól megtervezett és elkészített adatbázis képezi, ezért minden lehetőséget alaposan átgondolva hoztam létre a táblákat A fejlesztőkörnyezet fejezetben bemutatom az összes eszközt és technológiát, amelyeket a fejlesztés során használtam, így betekintést nyújtva az olvasónak az alkalmazás létrehozásának hátterébe.

A dolgozatom utolsó fejezete a fejlesztői dokumentáció. Miután bemutattam a tervezési folyamatot és a felhasznált eszközöket, fontosnak tartottam, hogy az olvasóval is megismertessem a fejlesztési folyamatot. Az alkalmazásban négy különböző szerepkört határoztam meg, ezeket alfejezetekben bemutatva ismertetem minden modul létrehozásának lépéseit. Mivel a JavaFX nyelv alapjait a dolgozat írása során sajátítottam el, előfordultak nehézségek, így egyes funkciók fejlesztése nem teljesen úgy valósult meg, ahogy azt eredetileg elképzeltem. Ennek ellenére számos új és hasznos ismeretre tettem szert a fejlesztés során. Sikerült egy számomra új programozási nyelv alapjaival részletesen megismerkednem, és korábbi ismereteimet felhasználva elkészítettem egy raktárkezeléssel kapcsolatos alkalmazást. A kész alkalmazás teljes mértékben megfelel a kitűzött elvárásoknak, minden tervezett funkció megvalósítása sikerült.

A közeljövőben tervezem az alkalmazás továbbfejlesztését és tökéletesítését: a felhasználóknak a jogkör beállítása, titkosított jelszókezelés, felhasználóváltás. A termékekhez tartozó áfa körrel már számoljuk a bruttó összeggel, de ezt még nem jelenítjük meg. Bővíteni a statisztikai adattközlést idő arányos sql lekérdezésekkel. Ezenkívül szeretném a programot több olyan funkcióval bővíteni, amelyeket a szakdolgozat írása során nem tartottam elengedhetetlennek.

8. Summary

The aim of my thesis was to create a desktop application for inventory management, which allows the easy and efficient management of product data in the warehouse. The application allows the recording, modification, and deletion of product data, as well as tracking the quantity of products in stock. In addition, I implemented a search function in the program, which allows users to quickly find the desired product. I started my thesis by presenting the functional and non-functional requirements, which helped me to create a clear picture of the expected operation of the program.

In the next chapter, I described the design process. After defining various diagrams and models, I used these to start designing the database. Here I determined what data needs to be stored and what the tables serving their storage should look like. Since the basis of the application is a well-designed and implemented database, I created the tables by thoroughly considering all possibilities. In the development environment chapter, I present all the tools and technologies I used during development, thus giving the reader an insight into the background of creating the application.

The last chapter of my thesis is the developer documentation. After presenting the design process and the tools used, I thought it was important to introduce the development process to the reader as well. In the application, I defined four different roles, and in subchapters, I present the steps of creating each module. Since I learned the basics of the JavaFX language while writing the thesis, there were difficulties, so the development of some functions did not fully realize as I originally imagined. Despite this, I gained a lot of new and useful knowledge during development. I managed to get to know the basics of a new programming language in detail, and using my previous knowledge, I created a marketable application related to lending. The finished application fully meets the set expectations, the implementation of every planned function was successful.

I plan to further develop and improve the application in the near future. Setting authority for users. We already calculate the gross amount with the VAT round for the products, but we do not display this yet. Expand the statistical data communication with time-proportional sql queries. I would like to expand the program with a function that I did not consider essential during the writing of the thesis.

9. Használati útmutató

A szakdolgozathoz mellékletként csatolt CD-n található egy Forrás nevű könyvtár, amely tartalmazza az alkalmazás forrásfájljait. A program fejlesztéséhez az IntelliJ Idea fejlesztői környezetet használtam, de bármilyen más IDE (Eclipse, Netbeans) segtségével is könnyedén importálható a forráskód.

Az alkalmazás használatához szükséges telepíteni a MYSQL XAMPP adatbáziskezelő szoftver 8.2.12 verzióját, melynek telepítéséhez szükséges fájlokat a következő címről lehet letölteni: https://www.apachefriends.org/hu/index.html

A telepítés során egy jelszó megadása kötelező, amely az alapértelmezett adatbázis jelszava lesz. Az általam készített adatbázis felhasználó neve: Bence a jelszó B lett, ezért csak ezen felhasználónév és jelszó használatával működni a raktár nyilvántartó program. Az adatbázis táblák pedig a csatolt sql scriptek adatbáziskezelő rendszerbe történő importálása után lesznek elérhetőek. Bejelentkezéshez minden felhasználónak egyéni jelszót adtam meg, a tesztadatok között szereplő személyek felhasználónevei pedig a következők:

felhasználónév: S jelszó: B

felhasználónév: M jelszó: m

stb. …….

A CD-n megtalálható még Szakdolgozat néven docx és pdf formátumban is a szakdolgozatom digitális változata.

Tartalomjegyzék

[1] Raktárkezelő programok

https://boxy.hu/blog/raktarkezelo-program

[2] JavaFX Scene Builder

https://docs.oracle.com/javafx/scenebuilder/1/installation_1-1/jsbpub-installation_1-1.htm

[3] IntelliJ IDEA

https://valdar.web.elte.hu/szofttech/#!/tools/intellij-idea

[4] MySQL

https://www.arubacloud.hu/tutorial/bevezetes-a-mysql-adatbazis-hasznalataba.aspx

[5] Ficsor Lajos, Krizsán Zoltán, Mileff Péter – Szoftverfejlesztés

https://gyires.inf.unideb.hu/KMITT/c02/

Melléklet

1. Számú melléklet: ER (Entity-Relationship) modell