A képen szöveg látható

Automatikusan generált leírás

Egyedi gyártástámogató szoftver kivitelezése

Custom production support software implementation

Badinszky Dániel Bence

HLISZU

Dr. Pap-Szigeti Róbert, főiskolai tanár

2025

1. Tartalomjegyzék

[1. Tartalomjegyzék 2](#_Toc196410005)

[2. A projekt előzménye és ismertetése 4](#_Toc196410006)

[3. Fejlesztési fázisok és szoftverkövetelmények 6](#_Toc196410007)

[3.1. Felhasználóbarát felület és gazdasági tényezők 8](#_Toc196410008)

[4. Szoftverek hasonló funkcionalitással 10](#_Toc196410009)

[4.1. SAP Business One 10](#_Toc196410010)

[4.2. Autodesk Fusion 360 10](#_Toc196410011)

[4.3. MRPeasy 11](#_Toc196410012)

[4.4. OpenBOM 11](#_Toc196410013)

[4.5. Katana 12](#_Toc196410014)

[4.6. Összegzés 12](#_Toc196410015)

[5. A szoftver tervezése 14](#_Toc196410016)

[5.1. Fejlesztési és futtatási környezet tervezése 14](#_Toc196410017)

[5.1.1. Architektrúrális tervezés 16](#_Toc196410018)

[5.1.2. DevOps módszertan 16](#_Toc196410019)

[5.1.3. Alkalmazást futtató konténer 17](#_Toc196410020)

[5.1.4. Adatbázist futtató konténer 18](#_Toc196410021)

[5.1.5. Felhasználói felület 18](#_Toc196410022)

[5.2. Adatbázis tervezése 18](#_Toc196410023)

[5.2.1. EK diagram 19](#_Toc196410024)

[5.2.2. Végső adatbázisszerkezet kialakítása 20](#_Toc196410025)

[5.2.3. Adatbázis méretezése 22](#_Toc196410026)

[5.2.4. Különlegesen terhelt időszakok 22](#_Toc196410027)

[5.2.5. Adatbázis rendszer kiválasztása 23](#_Toc196410028)

[5.3. Felhasználói felület tervezése 24](#_Toc196410029)

[5.4. OOP modellek tervezése 26](#_Toc196410030)

[5.4.1. Elemek osztály 27](#_Toc196410031)

[5.4.2. Media osztály 28](#_Toc196410032)

[5.4.3. Beszállító osztály 29](#_Toc196410033)

[5.5. Tesztelés tervezése 30](#_Toc196410034)

[6. Teljesítmény tesztelés 33](#_Toc196410035)

[6.1. Böngésző – Webszerver kapcsolat 34](#_Toc196410036)

[6.2. Webszerver – Adatbázis kapcsolat 35](#_Toc196410037)

[6.3. Teljes feldolgozási lánc tesztelése 37](#_Toc196410038)

[7. Esetleges áteresztőképesség növelése 38](#_Toc196410039)

[8. Fejlesztés közben felismert és megoldott hibák 39](#_Toc196410040)

[8.1. Raktárkészlet kezelése 39](#_Toc196410041)

[8.2. Tesztkörnyezet implementálása kódba 40](#_Toc196410042)

[8.3. Böngészők cache eljárásainak támogatása 40](#_Toc196410043)

[9. Felhasználói útmutató 41](#_Toc196410044)

[9.1. Telepítés 41](#_Toc196410045)

[9.2. Adatbázis beállítás 41](#_Toc196410046)

[9.3. Futtatás 42](#_Toc196410047)

[9.4. Kezdőképernyő 42](#_Toc196410048)

[9.5. Vezérlőpult képernyő 42](#_Toc196410049)

[9.6. Termékek megjelenítése és hozzáadása 43](#_Toc196410050)

[9.7. Elem megtekintése 43](#_Toc196410051)

[10. Jövőbeni fejlesztési lehetőségek 46](#_Toc196410052)

[11. Bevezetés utáni tapasztalatok 47](#_Toc196410053)

[12. Summary 48](#_Toc196410054)

[13. Mellékletek 50](#_Toc196410055)

[13.1. Osztály specifikációk 50](#_Toc196410056)

[13.1.1. Elements osztály 50](#_Toc196410057)

[13.1.2. Vendor osztály 54](#_Toc196410058)

[13.1.3. Media osztály 55](#_Toc196410059)

[13.2. Képernyőképek tervei 57](#_Toc196410060)

[13.2.1. Általános menü 57](#_Toc196410061)

[13.2.2. Bejelentkező képernyő 58](#_Toc196410062)

[13.2.3. Cégbemutató oldal 59](#_Toc196410063)

[13.2.4. Vezérlőpult oldal 59](#_Toc196410064)

[13.2.5. Elemek listája oldal 60](#_Toc196410065)

[13.2.6. Új elem hozzáadása felugró ablak 60](#_Toc196410066)

[13.2.7. Elem részletei oldal 61](#_Toc196410067)

[13.2.8. Elemek részletei oldal / Media oldal 63](#_Toc196410068)

[13.2.9. Elemek részletei oldal / Ikon beállítása 64](#_Toc196410069)

[13.2.10. Elemek részletei oldal / Beszerzési lehetőségek aloldal 65](#_Toc196410070)

[13.2.11. Elemek részletei oldal / Beszerzési lehetőségek aloldal / Felugró ablak 66](#_Toc196410071)

[13.2.12. Elemek részletei oldal / Tartalmaz lista 67](#_Toc196410072)

[13.2.13. Elemek részletei oldal / Tartalmaz lista / Új elem hozzáadása 68](#_Toc196410073)

[13.2.14. Elemek részletei oldal / BOM lista aloldal 69](#_Toc196410074)

[13.2.15. Elemek részletei oldal / Fa nézet aloldal 69](#_Toc196410075)

[13.2.16. Elemek részletei oldal / Ár aloldal 70](#_Toc196410076)

[13.2.17. Média oldal 70](#_Toc196410077)

[13.2.18. Beszállítói oldal 71](#_Toc196410078)

[13.2.19. Beszállítói oldal / Új beszállító hozzáadása felugró ablak 72](#_Toc196410079)

[13.2.20. Raktárkészlet oldal 73](#_Toc196410080)

[13.2.21. Jelszókezelő oldal 74](#_Toc196410081)

[13.2.22. Felhasználók oldal 75](#_Toc196410082)

[14. Szakirodalom 76](#_Toc196410086)

1. A projekt előzménye és ismertetése

A XX. század technológiai fejlődése nagy változást hozott az emberek és ezáltal a cégek életébe. A kifejezetten gyártással foglalkozó cégek előbb-utóbb igénybe veszik a számítástechnika által nyújtott lehetőségeket, hogy segítse saját munkáját. Ezzel erőforrásokat és időt tudnak megtakarítani. A konzervatív KKV-k ezt egész eddig valamelyest könnyen figyelmen kívül tudták hagyni, de a technológia fejlődésével idővel nekik is adoptálniuk kell valamilyen gyártástámogató rendszert. A nagyobb cégek ezt a funkcionalitást integrálva használják a vállalatirányítási rendszerükkel. Ezek a rendszerek egy teljes vállalkozás működését segítik. Leggyakrabban a telepített modulok tartalmazzák a HR, pénzügy és beszerzés kezelését is. Mivel egy ilyen rendszer bevezetése nagy költséggel jár, ezért egy kisebb vállalatnak túl hosszú a megtérülési ideje és nem teljes rendszerként alkalmazza, csupán egy-egy modult vásárol belőle.

Ez a szakdolgozat egy ilyen középvállalkozás gyártástámogatásáról szól. Az alkalmazást egy magyarországi székhellyel rendelkező, de amerikai háttérrel rendelkező gazdasági társaság fogja használni. A cég fő tevékenysége ipari méréstechnológia automatizálása. Ez magába foglalja teljes automatizált robotcellák tervezését és kivitelezését is. Az ehhez szükséges technológiák egy részét partneri hálóján keresztül viszonteladóként csak integrálja, egy másik részét viszont házon belül tervezi, fejleszti és gyártja. Ezekre a termékekre a továbbiakban saját termékekként hivatkozok. A cég saját termékei viszonylag széles palettán terülnek el. A termékkatalógus tartalmaz apróbb elektronikai rendszereket és nagy, asztal méretű mérőcellákat is. A cég világszerte kevesebb, mint 100 alkalmazottat foglalkoztat. A magyarországi telephely van kijelölve K+F és gyártóközpontnak. A hazai telephelyen 15-20 alkalmazott van foglalkoztatva. Ez eddig a pontig kellőképpen kevés volt ahhoz, hogy a gyártást konzervatív módon menedzseljék. Ez a gyakorlatban Excel táblákban tárolt BOM listákat, rendelési összesítőket és raktárkészlet-nyilvántartást jelent. A különböző táblázatok naprakészen tartása és kezelése emberi-erőforrás és idő igényes. Nehezen reagál a külső változásokra, mint a beszállítóktól való függésre. Egy-egy harmadik féltől vásárolt termék árváltozása vagy helyettesítése más termékkel sok helyen indikál változásokat. Ezek redundánsan tárolt adatok, amik nagy lehetőséget adnak hibázni a munkafolyamat során.

Minden bejövő megrendelés során egy teljesen új számítás készül el az éppen eltárolt alkatrész-készlet és ár összesítése alapján. Ebből lehet kiszámolni a várható elkészülési időt és bekerülési költséget.

A cég elkötelezett, hogy a teljes gyártási folyamat átláthatóbb és könnyebben kezelhető legyen. Szeretne egy alkalmazást, ami képes a saját termékeinek a gyártásához szükséges metaadatok központi helyen történő tárolására. Ez eliminálja a redundáns adattárolás problémáját. Amennyiben az adatok egy helyen vannak tárolva, onnan különböző algoritmusok képesek lehetnek projekt költséget és megvalósítási időt becsülni. A leghamarabb megoldandó problémájuk közé tartozik az elektronikai alkatrészeik regisztrálása. A saját termékeik közé tartoznak egyedi nyomtatott áramkörök, amelyek több 100 alkatrészt is tartalmazhatnak. Egy-egy projekt több 10 ilyen áramkört foglal magába. Ezeknek a megrendelése és raktárkészletének nyilvántartása egyszerű táblázatokban kényelmetlen és ezáltal nem is hatékony.

Szükségük van egy szoftverre, ami képes a teljes gyártási folyamatukat az ő középvállalkozás méretükhöz képest támogatni. A cég által használt informatikai megoldások nagyon szélesek. Windows, Linux és Mac operációs rendszerek is gyakran használva vannak. Ezen túl gyakran előfordul, hogy telefonon kell elérni ezeket az információkat. Ez előrevetíti, hogy egy teljesen cross-platform alkalmazásra van szükségük. A vállalkozás egy webes alkalmazás mellett döntött. Ez teljesíti azt az igényüket, hogy gyorsan lehessen fejleszteni és frissíteni az alkalmazást működés közben. A fejlesztői környezetük, amit a fejlesztők használnak, Linux rendszer és alapvetően Python programozási nyelv.

Ez a webalkalmazás fogja kezelni a cég által gyártott összes termék BOM listáját, amelyek allistákat is tartalmazhatnak. Az adataik rekurzív hierarchiában rendezkednek el. A szoftvernek ezt a struktúrát kell tudja megvalósítani. Egy ilyen rekurzív hierarchia összegzése egy projektre vetítve lehetőséget ad bekerülési költség és gyártási idő becslésére.

1. Fejlesztési fázisok és szoftverkövetelmények

A projektmegvalósítási fázisok első lépéseként meg kell érteni a megrendelő megoldandó problémáját. Tisztában kell lenni a jelenlegi munka-technológiájukkal és ezután meglehet állapítani az optimalizálandó folyamatokat. Ehhez első lépésként a legcélravezetőbb megoldás egy gyárlátogatás és beszélgetés az ottani dolgozókkal. Erre egy legalább 1-2 napos időintervallumot éri meg fenntartani a projekt elején. Ha nem tiszta a megrendelő problémája az a megvalósítási szakaszban újra tervezést igényelhet, ami extra időt és pénzt jelent a projekt életciklusában.

Ebben az időben a helyszínen való tartózkodás során a cég hierarchia felső részétől lefelé haladva próbáljuk megérteni a cég mindennapi életét. Először az ügyvezetővel vagy a csoportvezetővel való beszélgetés során egy átfogó és menedzser szemléletű áttekintést kapunk. Ez a fajta álláspont áll a legközelebb a valóban optimális munkavégzéshez, de sajnos a tapasztalat azt mutatja, hogy a hierarchiában lefelé haladva és azokkal beszélve, akik a mindennapi munkát végzik ezek a folyamatok teljesen másképp vannak kivitelezve.

A csoportvezetővel való egyeztetés során átfogó ismereteket kaphatunk az aktuális munkafolyamatokról. Ekkor kerülnek szóba a megoldandó problémák általános szintű jellemzései.

Ebben a konkrét esetben a problémát a cég növekedése okozza. 5 év leforgása alatt a világszerte alkalmazottak száma 30-ról kevesebb mint 100-ra nőtt. Ezt a humán faktort és az ezzel járó extra gyártási kapacitást már nem képes a konzervatív Excel táblás gyártástámogatás kezelni és egy komplexebb rendszer bevezetése szükséges.

A cég az egyedileg gyártott alkatrészeit és a külső gyártótól származó alkatrészeket is táblázatos formában tárolja. Az ezekhez tartozó belső kódok, külső beszállítói kódok és árak rendelkezésre állnak. Egy érkező megrendelés esetén minden alkalommal egy új táblázat létrehozása szükséges, amibe az adott projektre jellemző alkatrészek átmásolása történik. Első megoldandó problémaként feljegyezzük az ezzel járó adat duplikálást, ami hibákhoz vezet hosszútávon. A megrendelő megerősítette, hogy sajnos ez a probléma valós és a táblázatokban lévő másolt adatok nem tudnak 100%-ban megegyezni. Egy-egy megvásárlandó termék esetén annak az árát frissíteni kell az összes táblázatban. Ennek a módszernek előnye, hogy a futó projektek nem keveredhetnek, hiszen külön fájl tartalmazza az adataikat. Habár ez megnehezíti a beszerzés munkáját, mert minden projekt megrendelési igényét össze kell fésülni egy megrendelendő táblázatba, amit később össze lehet vetni az adott raktárkészlettel.

A csoportvezetői szinten a projektmenedzselés fontos lépése a folyamatos projektkövetés idő és kiadás tekintetében. Ez a módszer egy projekt időbeni igényének meghatározására teljesen alkalmatlan, mert a munkafolyamatok időigénye nincs meghatározva és összesítve, így csak a tapasztalataikra és statisztikára tudnak támaszkodni. Kiadás tekintetében pontos képet tud festeni a jelenlegi módszer is a projektről, de ennek feltétele, hogy a beszerzési oldal frissítse a táblázatokat az aktuális árakkal.

A gyártói csoporttal beszélgetve fény derül a mindennapi munkát befolyásoló problémákra is. Ez a fajta mindennapos adminisztráció rengeteg extra munkaidőbe telik és túl monoton ahhoz, hogy hiba nélkül lehessen végezni. Egy-egy hiba keresése a számos táblázat között pedig több órás folyamat is lehet. Tudják, hogy ez a munka automatizálható, ami pontosabb és gyorsabb munkavégzést tenne lehetővé.

A beszélgetések során több alkalmazottnak is voltak korábbi tapasztalatai külsős szoftverekkel, amiket megemlítettek, de abban egyetértettek, hogy a cég még nem nőtt akkorára, hogy egy teljes vállalatirányítási rendszer bevezetése legyen szükséges. Ennek sem az anyagi terhét nem tudja állni, sem az emberek betanítására szükséges idő nem áll rendelkezésre. További aggályként említették a fejlődés lehetőségét is. Amennyiben egy dobozos terméket vesznek meg, annak a későbbi testre szabása nehézkes lehet, ami akár külsős támogatást is igényelhet. A külsős támogatást pozitívumként lett megemlítve, mert a cégnek is biztonságot ad, ha egy ezzel foglalkozó társaság karbantartja és fejleszti a szoftvert. A cég nem szeretne ennek a fejlesztésébe extra munkaidőt beletenni, inkább a külsős fejlesztést támogatja.

A probléma megértése után a következő fázisban meghatározzuk a megoldást jelentő szoftver specifikációját. Annak alapvető paramétereit, funkcióját és felhasználási módjait.

Elsőként a felhasználás módjának tisztázására kerül sor. Listába kell venni a cég által üzemeltetett összes olyan számítógépet, amin a szoftvernek később futnia kell. Ebben a listában mind a három gyakran használt operációs rendszer megtalálható volt. A beszerzési oldalon jellemzően Windowst használnak, a gyártói gépeken Linux operációs rendszer van és a management pedig MacOS rendszereket futtat. Ezen a ponton két javaslatot vetettünk fel: egy cross-platform alkalmazást vagy egy web alapú applikációt.

Második fontos pontként megállapítjuk a szoftver alapvető funkcióit és megemlítjük a későbbeikben esetlegesen hozzáfejlesztendő egyéb modulokat is.

Alapvető funkcionalitás már az első verzióban:

1. Különböző bejelentkezési jogosultsági szintek megléte. Külön tervezői és adminisztrátori szintek kijelölése.
2. Az összes termék hierarchiában történő tárolása. és egy helyen történő kezelése. Ez oldja meg a duplikált adat problémáját és egyszerűsíti az adatok karbantartását egyidejűleg. Az ehhez tartozó hierarchia részletezésére később kerül sor.
3. Egy projektre képes kell legyen költség és munkaerőigény becslést adni.
4. Alkatrészekhez/termékekhez tartozó dokumentumok tárolása.
5. Alkatrészekhez/termékekhez tartozó képek tárolása a könnyebb beazonosítás végett.
6. Raktárkészlet nyilvántartás a rendelés optimalizálása céljából.

Későbbiekben hozzáadható modulok, melyek előkészítése már most is szem előtt tartandó:

1. Munkafolyamatok emberekhez rendelése
2. Termék utókövetés. Minden legyártott termék egyedi rekordként szerepelne az adatbázisban a későbbi visszakövethetőség céljából.
3. Automata alkatrész rendelés külső partnerektől

A rendszer üzemeltetése szempontjából fontos információ a szoftver futtatókörnyezete és annak limitációi. A már meglévő informatikai infrastruktúrába integrálhatóság elengedhetetlen. Ez egy szervert jelent, amin több virtuális gép futtatására van lehetőség. Ehhez a rendszerhez egy dedikált virtuális gép rendelhető. Javaslatként gyártói oldalról felmerült a dockerizáció lehetősége, amely szintén megvalósítható.

* 1. Felhasználóbarát felület és gazdasági tényezők

A szoftver létjogosultságának két fő oka a rendelkezésre álló munkaerő hatékonyabb felhasználása, ami gazdasági szempontokból is megközelítendő. Jelenleg több dolgozó is heti szinten több órát tölt a jelenlegi rendszer naprakészen tartásával és üzemeltetésével. Alapvető követelmény, hogy az új rendszer napi használata ennél kevesebb munkaórát vegyen igénybe. Ezzel munkaerőt tehermentesít, ami gazdasági oldalról nézve kedvező. A KSH hivatalosan elérhető adatai szerint [1] egy Általános irodai adminisztrátor a 2021-es évben átlagosan 313.730 Ft-os havi fizetéssel bírt. Ezt az elmúlt évek hivatalos inflációs adataival szorozva, feltételezve, hogy legalább ekkora fizetésemelést kaptak a munkavállalók fizetése 2025-re 438.074-ra emelkedett (1. ábra). Havonta átlagosan 20 munkanappal számolva bruttó 2737 Ft-os órabérrel lehet számolni. Napi szinten a meglévő rendszernek a fenntartása az ügyfél tapasztalatai szerint naponta 2 munkaórát vesz igénybe. Ezen számítások szerint napi szinten jelenleg 5474 Ft a rendszer fenntartása. A szoftverrel szemben követelmény, hogy az új rendszer 5 éven belül megtérüljön. A megtérülésnél fontos számolni az új rendszer fenntartásához szükséges munkaerővel és a szoftver díjával is.

1.ábra. Éves infláció

1. táblázat. A technikai specifikációk összegzése

|  |  |
| --- | --- |
| Architektúra | Virtuális gép, docker konténer |
| Operációs rendszer | Virtuális gép: Debian linux |
| Docker: - |
| Programozási nyelv | Python |
| Használt keretrendszer | Django |
| Megjelenítésért felelős | Beépített webszerver |
| Adatbázis | MariaDB |

1. Szoftverek hasonló funkcionalitással

A projekt részeként egy piackutatás elvégzése is szükségessé vált. Ezzel számba lehet venni a már ilyen funkcionalitással bíró egyéb szoftvereket, azok erősségeit és gyengeségeit. Mivel a projekt összetett és nem egy egyszerű BOM lista nyilvántartó funkcionalitással bír egy sokkal komplexebb szinten kell vizsgálódni, ami már-már a vállalatirányítási rendszerek felé indul el, mivel beszerzési és raktárkészlet-nyilvántartási célokat is szolgál.

* 1. SAP Business One

A vállalatirányítási rendszerek körében megkerülhetetlen szereplő az SAP. A céget 1972-ben alapították Németországban és főleg a saját ERP rendszeréről vált ismertté. Teljeskörű megoldást kínál a cégek számára a könyveléstől kezdve, a beszerzésen át a munkaidő-nyilvántartásig. Teljesen testre szabható moduljaik vannak, amelyet a megrendelő ízlésére szabnak. Ezekhez a termékekhez gyakran a cégek támogatási szerződést kötnek, amivel további funkciókat kapnak a rendszerbe és a cégek életében történő változásokat naprakészen tartják a szoftveres oldalon is.

Természetesen nekik is van erre a problémára megoldásuk. Ez a megoldás némi testre szabást jelentene az ügyfél részére, de a cég ezt vállalja. Ennek a teljes körű eljárásnak olyan anyagi vonzata van, amit egy ekkora KKV nem tud vállalni, mert előre nem látható a megtérülése. Egy ilyen fajta szoftver évenkénti fenntartása €3.000 - €5.000-ba kerül, ami nem összehasonlítható a jelenlegi extra kiadásokkal.

Funkcionalitását tekintve képes többszintű BOM listák kezelésére. Képes rekurzívan kezelni az összeszereléseket, hogy ne legyen duplikált adat az adatbázisban. Számon tartja az anyagok beszerzési költségét és gyártási idejét, amiből a végfelhasználó teljes becslést kap a projektjére vetítve. Nincs olyan kitétel a projekt specifikációban, amit ez a termék ne teljesítene a megtérülésén kívül. Az összehasonlításhoz használt forrás a saját weboldalukon publikusan elérhető adatok. [2]

* 1. Autodesk Fusion 360

Az Autodesk főleg a mérnöki iparban ismert cég. 1982-ben alapították az Amerikai Egyesült Államokban. AutoCAD szoftverük a CAD modellezés meghatározó szoftverévé vált. Manapság az új trendeket követve a felhő alapú tervezés irányába tendál a cég. Marketingjük az elmúlt időszakban gyökeres fordulatot vett és az egyszeri termékdíjjak helyett a felhős megoldások révén inkább szolgáltatási díj ellenében lehet használni a megoldásaikat. Habár nem fő tevékenységük, de a 3D modellezéshez szükséges támogatás részeként nyújtanak BOM kezelő szolgáltatást is. Ennek funkcionalitása limitált. Ennek oka, hogy a szoftvert főként 3D tervezésre és gyártásra fejlesztették, mint az ehhez szükséges folyamatok támogatására. Képes több szintű és rekurzív BOM listák kezelésére. Azonban ezen alkatrészek beszerzésére már nem nyújt natívan megoldást. Ahogy natív modulként az emberi munkaórák számontartása sem megvalósítható. Ez csupán külső beépülő-modulokkal lehetséges, ami megnöveli a projekt komplexitását és költségeit.

Árazását tekintve olcsóbbnak tekinthető, mint az SAP, de a funkcionalitása is korlátozottabb és nem is teljes mértékben testre szabható. Egy ilyen rendszer éves szinten €495-s szolgáltatási díjat feltételez felhasználókként évente. [3]

* 1. MRPeasy

Az MRPeasy egy felhő alapú MRP (Manufacturing Resource Planning) rendszer. A 10-200 fővel bíró cégek a fő piaca. Az előzőktől eltérően ezt a szoftver erre az egy célra tervezték és fejlesztették. A cégnek az Egyesült Államokon túl Nagy Britanniában és Észtországban vannak irodáik. Elkötelezettek a hatékony gyártástámogatás mellett, amely környezetvédelmileg is fontos. Ezen túl saját tőkéjéből is erősítésbe fogott. Funkcionalitását tekintve a projekt első fázisában megvalósítandó feladatokat teljes mértékben ellátja. A projekt második fázisaként megemlített funkciókat azonban nem teljesíti. Nem képes az emberi erőforrásokat kezelni és munkafolyamatokat hozzárendelni. Így ennek lefejlesztése extra igénybevétel lenne a költségvetésben.

Árazását tekintve ez is egy szolgáltatás, ami havi vagy éves előfizetésként érhető el többféle csomagban. A specifikációban említett és a szoftver által támogatott funkciók a Professional csomagban érhetőek el havi szinten €59 költségért. Ebbe azonban a projekt második fázisának teljesítéséhez szükséges modulok nem tartoznak bele, amely extra fejlesztést igényel. [4]

* 1. OpenBOM

Nevével ellentétben ez a szolgáltatás nem egy nyílt forráskódú rendszerre utal. Egy piaci igényekre szabott gyártástámogató rendszert fed. Egy Egyesült Államokbeli cég fejleszti és értékesíti. A cég víziója a gyártástámogatás központosítása. Ez magába foglalja a szükséges modellek kezelését, a csapaton belüli kollaboráció megvalósítását, a beszerzés előkészítését és külső szoftverekkel való együttműködést is. Célközönsége ennek a szolgáltatásnak is a KKV szektor. Az elérhető moduljai lefedik a projekt első fázisában elvárt működést. Azonban a második fázis pontjai közt van teljesíthetetlen ennek a szoftvernek. Ebben az esetben specifikációs kompromisszumra lenne szükség. Árazását tekintve minden felhasználó után kell licenszt vásárolni és havi vagy éves szinten megújítani. Ennek havi költsége $100 felhasználónként.

* 1. Katana

Ez a cég egy teljeskörű ERP szolgáltatást nyújt. Ebbe beletartozik a beszerzés támogatása és automatizálása, a valós idejű raktárkészlet kezelés, a bejövé megrendelések nyomon követése és az értékesítési folyamtok támogatása is. Funkcionalitásában ez a szoftver sem támogatja az emberi erőforrások kezelését és azok költségének becslését, ami a projekt specifikációban szerepel. Árazását tekintve havi szinten $179 költséget jelent egy KKV számára az üzemeltetés.[5]

* 1. Összegzés

2. táblázat. Rendszerek összegzése

|  |  |  |  |
| --- | --- | --- | --- |
|  | **Alapvető funkcionalitás** | **Jövőbeni elvárt funkcionalitás** | **Becsült költség** |
| **SAP Business One** | Teljes mértékben alkalmas | A cég által nyújtott egyedi fejlesztéseket igénybe véve alkalmas | Évenkénti €3.000 - €5.000 költség |
| **Fusion 360** | Limitáltan alkalmas | Nem alkalmas | Havi €495 költség |
| **MRPeasy** | Teljes mértékben alkalmas | Nem alkalmas | Havi $59 költség |
| **OpenBOM** | Teljes mértékben alkalmas | Nem alkalmas | Havi $100 költség |
| **Katana** | Teljes mértékben alkalmas | Nem alkalmas | Havi $179 költség |

A fentiekben megvizsgált szoftverek mind részben vagy egészben alkalmasak lennének a projekt specifikációban foglaltak megvalósítására. Habár ennek jelentős anyagi és/vagy emberi erőforrás igénye lenne. Egy helyi KKV társaság számára az ekkora befektetés nem belátható időn belül térül meg ami a teljes beruházást megkérdőjelezi. Az itt említett szoftverek mind kezdeti költségek nélküliek, ámbár havi költséget generál. Az egyedi szoftver fejlesztése ennek az ellenkezője. Abban az esetben egy nagyobb összeget kíván a projekt bevezetése, ami a fejlesztési és bevezetési időszakot lefedi majd utána nem kíván extra költséget a helyben már jelenleg is üzemeltetett szerverpark okán.

A létező szoftverek összehasonlítása objektív módon eredményeképpen az MRPeasy szoftver fedi le leginkább a projekt specifikációkat. Ebbe az összehasonlításba 5 szempont került bele és részletezése az 2. ábrán látható.

2. ábra. Létező megoldások összehasonlítása

1. A szoftver tervezése

Az előzetes igényfelmérés alapján lehet elvégezni a szoftver tényleges tervezését. Ez több egymástól látszólag független részből áll, ám ezek nagyban függenek egymástól. A tervezés elhanyagolhatatlan részei:

1. Fejlesztési és futtatási környezet tervezése
2. Adatbázis tervezése
3. Felhasználói felület tervezése
4. OOP modellek tervezése
5. Tesztelés tervezése
   1. Fejlesztési és futtatási környezet tervezése

A szoftver végfelhasználójának javaslata a programozási nyelvre tekintve a Python lesz. Az Ügyfél oldalról ennek indoklása a későbbiekben való egyszerű továbbfejlesztés lehetősége akár házon belül is. Ennek lehetősége természetesen megvizsgálandó.

*„Python* is a programming language (with syntax rules for writing what is considered valid Python code) and the Python interpreter software that reads source code (written in the Python language) and performs its instructions.” [6]

A nyelv fő célja a programozási munka gyorsabbá és egyszerűbbé tétele akár a futtatási sebesség csökkenésének árán is. Interpretált nyelv, amely könnyen lehetővé tette a főbb operációs rendszerekre történő interpreter fejlesztését, így egy platformfüggetlen nyelv létrehozását. Napjainkban ez a programozási nyelv az egyik legnépszerűbb egyszerűsége, függetlensége és elérhető csomagjai okán. A mesterséges intelligencia fejlődése is ezen a nyelven vette kezdetét, mert a matematikusok ebben a környezetben tudták először legegyszerűbben implementálni algoritmusaikat.

A projekt kezdetén a legfrissebb elérhető verzió a 3.12.3 volt. Emiatt ez lett a fő fejlesztési környezet a szoftvernek.

A hardveres futtatókörnyezet egy hangsúlyos tervezési pont. Mindenképpen elegendő számítási kapacitással kell, hogy majd rendelkezzen a tervezett algoritmusok valós idejű futtatásához. Az előzetes egyeztetések kiterjedtek a hardveres lehetőségekre is. Ennek okán a futtatásra rendelkezésre álló hardverek képesek biztosítani teljesen elszeparált virtuális gépet és Linux alapú konténert is. A két futtatási lehetőség közüli választás a fejlesztő cég feladata.

A virtuális gépek lényege, hogy a használt operációs rendszer függetlenítve legyen az alatta elhelyezett fizikai hardvertől. Ezt egy extra virtualizációs réteg elhelyezésével éri el, amely szimulálja vagy emulálja az virtuális gép által igényelt hardver elemeket. Ennek előnye, hogy a virtuális gép képfájljai minden fizikai hardveren az adott architektúrán belül képes futni. Ennek a virtualizációnak a hátránya az erőforrásigény. Ezzel szemben előnye, hogy operációs rendszer független, így egy Linux szerver is képes Windows operációs rendszerrel ellátott virtuális gépek futtatására. [7]

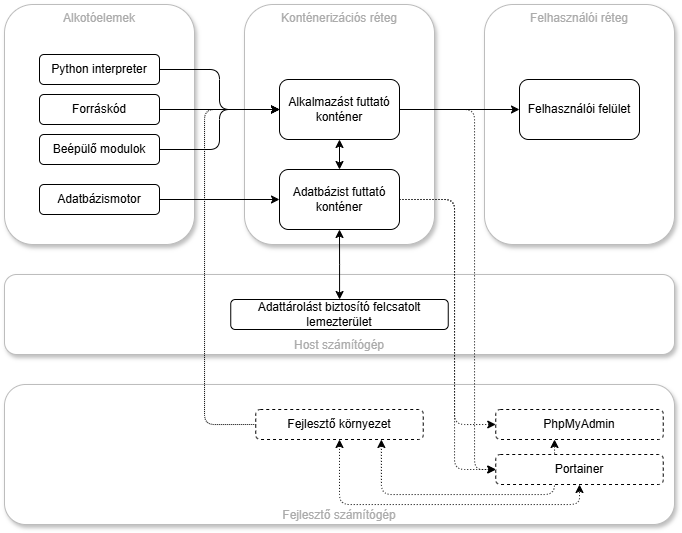
A konténerizációs technika ezzel szemben egy kevésbé erőforrásigényes, ám sokkal korlátozottabb technika. Ebben az esetben nem lesz virtualizált hardver és arra telepített teljesen szeparált operációs rendszer. Itt a konténer a host számítógép operációs rendszerét (és kernelét) használja, ám egy teljesen elszeparált és izolált környezetben. Ennek előnye, hogy nem igényel extra háttértárat ezen szoftverelemek tárolása, illetve a szoftver indulási ideje is lerövidül a kernel betöltődésének hiánya miatt. Ebben az esetben a host rendszerre telepített szoftverek és a kontérnerben lévő szoftverek teljesen el vannak különítve. Ennek a megoldásnak köszönhetően a függőségi problémák könnyen megoldhatóak lesznek, mivel egy fizikai host gép is képes lehet egy adott szoftver akár több verziójának egyidejű futtatására is. [7]

3. táblázat. Rendszerek összehasonlítása

|  |  |  |
| --- | --- | --- |
|  | Virtuális gép | Konténer |
| Erőforrásigény | Sok erőforrást igényel | Kevés erőforrást igényel |
| Indulási idő | Lassú indulási idő | Gyors indulási idő |
| Operációs rendszer függetlenség | Teljesen független a host rendszertől | A host rendszer kernelét használja |
| Dedikált erőforrás | Előre megszabott és dedikált erőforrások | Dinamikusan kezelt erőforrások |

A jelenleg elérhető host számítógép egy Linux alapú operációs rendszert futtat. Mivel a Python interpreter teljes mértékben kompatibilis ezzel a rendszerrel, így a konténerizációs technika a javasolt technológia. Ebben az esetben a megírt forráskód egy úgynevezett Dockerfile segítségével könnyen előállítható és nem szükséges a teljes alkalmazás tárolása, amely hosszútávon könnyebbséget jelent. Ebben a technológiában azonban akár több konténer összekapcsolása is szükségessé válhat a funkcionalitások szétválasztása végett.

* + 1. Architektrúrális tervezés

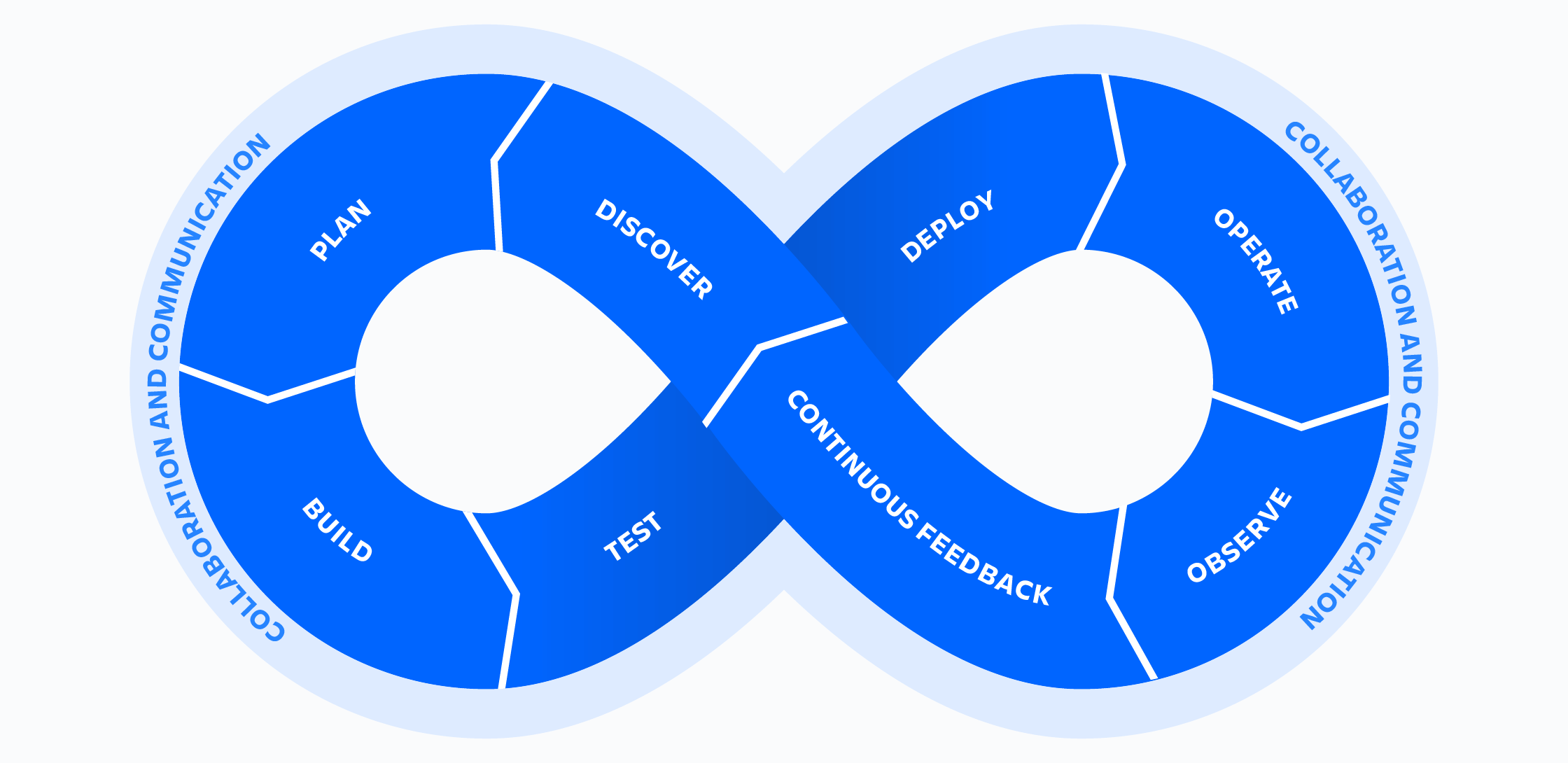


3. ábra. Architektúrális futtatási és fejlesztési terv

Az alkalmazás futtatásához 2 db konténerre lesz szükség. Az két különválasztandó funkcionalitás a fejlesztendő applikáció és a hozzá csatolt adatbázis rendszer. A fejlesztés közben szükségessé válhat két segédalkalmazás is (3. ábra). A PhpMyAdmin szolgáltatás [25] egy grafikus kezelőfelületet nyújt az adatbázishoz. Ez felhasználóbarátabbá teszi a fejlesztés folyamatát. A Portainer alkalmazás [8] grafikus felületen engedi kezelni a futó konténereket a számítógépen. Fejlesztés közben a paraméterek pontos beállításához nagy segítséget jelent. Ámbár ezek a futtatókörnyezetben nem szükségesek.

* + 1. DevOps módszertan

A DevOps egy agilis szoftverfejlesztési technika. [9] A fő előnye a gyors reagálóképessége. A fejlesztési és publikálási fázis gyakran követik egymást és apró módosításonként publikálás következik. Ekkor a hibakeresés könnyebb, a kevesebb módosított forráskód miatt. Továbbá ezzel a módszerrel lehet a leghamarabb új funkcionalitást szállítani. Ezt sok apró segédalkalmazással lehet elérni, melyeknek egyenkénti ismerete elengedhetetlen az alkalmazásához. [10]



*4. ábra. A DevOps ciklusa*

Ez a módszertan nagy fokú automatizálást enged meg. A tesztelési és publikálási fázis szinte teljes egészében azonnal történik egy git trigger esemény hatására. Egy teljesen integrált rendszerben akár az aktuálisan élesben futó alkalmazást is képes az újra cserélni beavatkozás nélkül. [10]

Ebben a projektben fontosak ezek a felsorolt értékek, így ezek az elvek figyelembevételével zajlik a fejlesztés.

* + 1. Alkalmazást futtató konténer

Szükség lesz egy olyan futó állományra, amely magába foglalja a teljes alkalmazás kódját, a futtató környezet kódját és a beépülő modulok kódját is. Ebben az esetben ezt egy egyedi Docker konténer fogja megvalósítani. Ezt a szolgáltatást a Git-tel összekapcsolva a forráskód fejlesztése közben a konténer automatikusan legenerálódik. Itt történik a beépülő modulok importálása is. Ilyen modul a Django keretrendszer is. [11] Ennek segítségével valósul meg a megjelenítési funkció egy beépített web szerver segítségével. Ebben a keretrendszerben több olyan funkció is natívan meg van valósítva, amely ebben a projektben elengedhetetlen, mint például a többszintű felhasználókezelés. Ehhez a Django egy külön adatbázist használ. A fejlesztés során nagy figyelmet kell fordítani arra, hogy a Django specifikus és a fő feladatot ellátó rendszerek adatbázismodelljei ne legyenek semmilyen kapcsolatban egymással. Ez egy jövőbeni keretrendszer váltás esetén egy segíthet, hogy az alkatrészeket tartalmazó adatbázis nem hoz függőségeket a Django modellből. Ez a keretrendszer egy MVC programtervezési mintát használ. Ennek fő célja, hogy 3 elkülönített réteg felel az adatokért és azok megjelenítéséért. [11]

A fent említett 3 réteg közül az adatelérési réteg szolgál arra, hogy a saját adatbázis szerkezet implementálásra tudjon kerülni. Ezzel egy bármelyik másik MVC modellre épülő keretrendszerbe is könnyen beilleszthető lenne.

* + 1. Adatbázist futtató konténer

Egy elszeparált funkcionalitás az adatbázis szolgáltatás. Az alkalmazás enélkül nem kap adatokat és nem képes működni. Az elszeparáltsága segít egy esetleges jövőbeni erőforrás bővítés esetében. Az alkalmazás konténer egy hálózati kapcsolaton kommunikál az adatbázissal. Ez ebben az esetben localhoston keresztül történik a leggyorsabb elérés érdekében.

* + 1. Felhasználói felület

A végfelhasználó egy webszervert ér el. Ez biztosítja, hogy telepítés nélkül korlátlan eszközről elérhető legyen a szolgáltatás. Ezt a funkcionalitást az alkalmazást futtató konténer adja egy nyitott porton keresztül. Amennyiben a teljesítmény csökkenése lenne tapasztalható egy teljesítmény elosztó bevezetése válhat szükségessé és több példány egyidejű és szinkronizált futtatása lenne elengedhetetlen.

* 1. Adatbázis tervezése

Az alkalmazás két külön adatbázist használ két különálló és független funkció biztosítására. Az egyik adatbázis a felhasználók adatait tartalmazza és a hitelesítéshez van rá szükség. Ennek struktúrája adott a Django dokumentációból. Ezen a ponton kell meghozni az első fontos döntést az adatbázissal kapcsolatban, mert a keretrendszernek támogatnia kell a használandó szolgáltatást. Szerencsére széles választék áll rendelkezésre, mint például a PostgreSQL, MySQL, MariaDB vagy akár az SQLite is. A választás során figyelembe kell venni az alkalmazásban lévő másik adatbázist is, hogy minden kellő funkcionalitás megtalálható legyen a választott motorban.

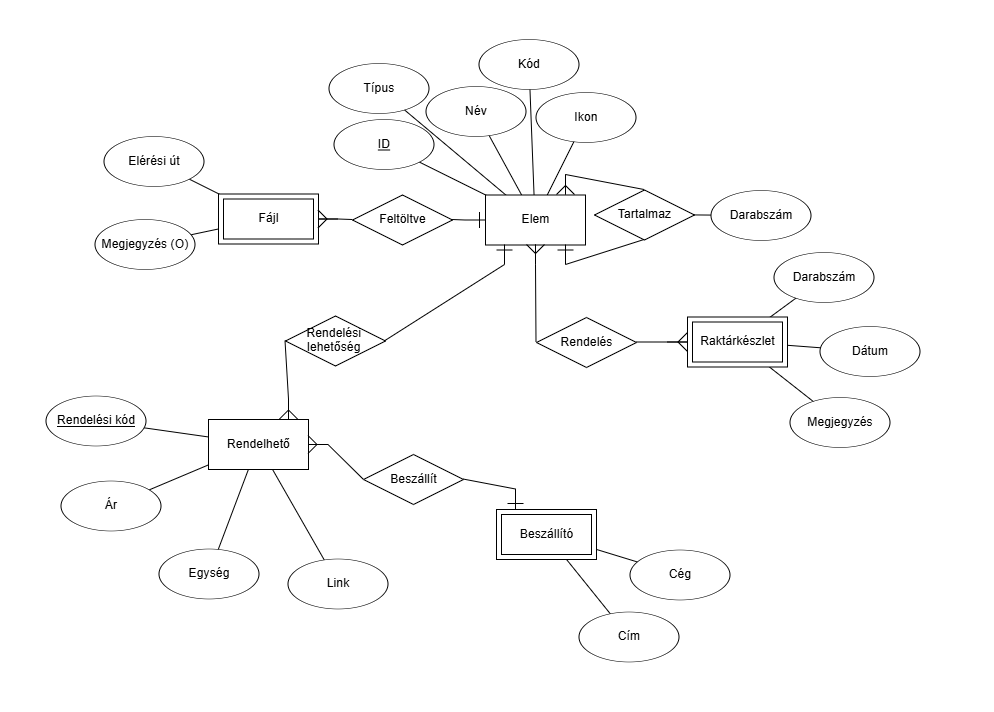
Az alkalmazás másik adatbázisa tartalmazza az alkalmazás futásához szükséges érdemi adatokat és struktúrákat. Ez tárolja az alkatrészeket, a beszállítókat és a raktárkészletet is. Az előzetes igényfelmérésből kiderült, hogy az Ügyfél a következő struktúrában szeretné tárolni a gyártását.

5. ábra. Gyártás hierarchia modellje

* + 1. EK diagram

Az adatbázis tervezéséhez szükség lesz az EK diagramra. Ehhez össze kell gyűjteni a tárolandó tulajdonságokat. Az adatbázis fő elemét az „Elem” fogja képezni. Ez tárolja az összes tulajdonságot, és ebből a táblából indulnak ki az altáblák az egyszerűsítés végett. Az Elem nevű egyednek tárolnia kell a következő tulajdonságokat:

1. Egyedi azonosító
2. Típus (Projekt, Termék, Összeszerelés, Művelet, Alkatrész)
3. Név
4. Gyártói kód
5. Ikon – Miniatűr kép
6. Feltöltött fájlok adatai
7. Rendelési lehetőségek, amennyiben vannak
8. A rendelési lehetőségekhez szükséges beszállítói adatok
9. Raktárkészlet információ

Ezekben a tulajdonságokban több is összetett vagy duplán tárolt lenne, így az egyed kapcsolat diagramban ez már szétbontásra került a minél könnyebb tárolás és használat céljából.

6. ábra – EK diagram

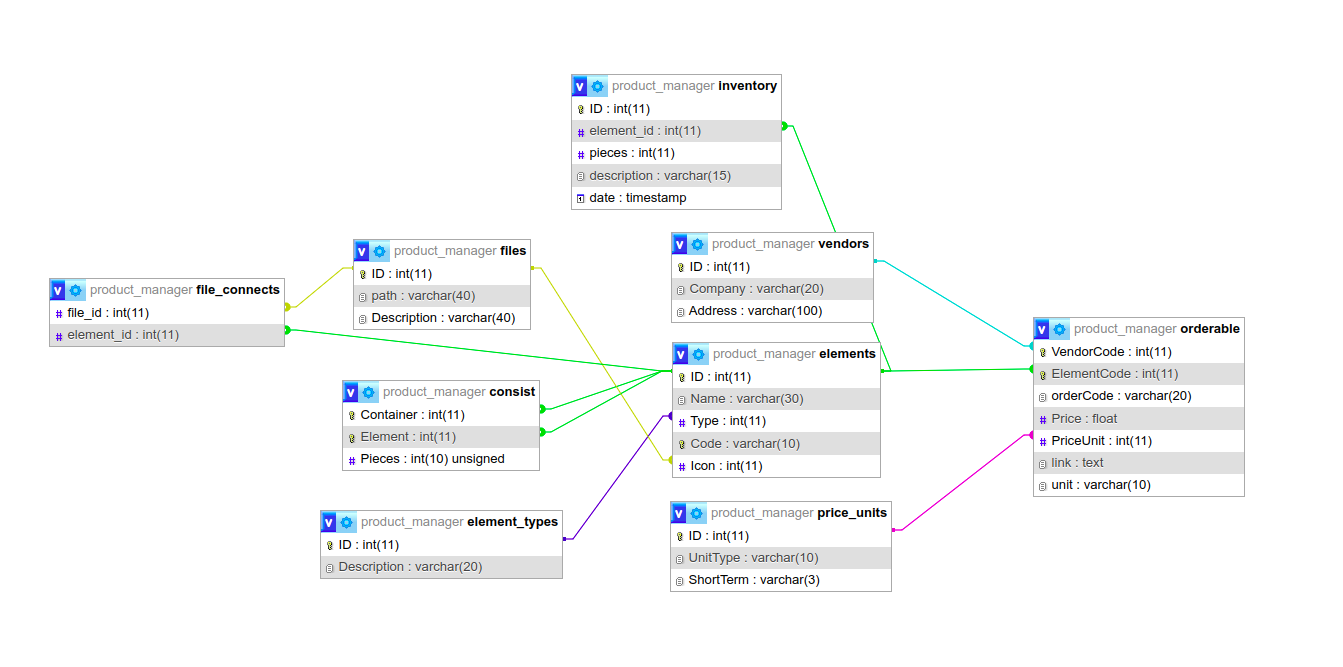
* + 1. Végső adatbázisszerkezet kialakítása

A fenti EK diagram szerinti megvalósítás lehetővé teszi az elemek rekurzív egymásba rendezését, több fájl egyidejű hozzácsatolását, több rendelési lehetőség rögzítését és a raktárkészlet kezelését is, rendelési történettel.

A normalizálási szabályok figyelembevételével kell elkezdeni a végső adatbázis tervezést. A jelenlegi EK diagram nem felel meg az első normálformának sem, mivel vannak olyan attribútumok, amelyek összetettek. [12]

A *fájlok* egy összetett attribútum, amely külön táblát igényel ezen szabályok szerint. Ugyan ezen logika mentén kerül a rendelési lehetőségek és a beszállítók is külön-külön táblába. Ez biztosítja majd, hogy ne legyen felesleges duplikált adat az adatbázisban, és csupán egyszerű attribútumok szerepeljenek. A raktárkészlet is külön táblát igényel, mivel több az egyhez kapcsolatot kell megvalósítani. Egy termék többször is bevételezésre kerülhet és többször kiszállítható külön-külön termékekben. A tervezés során azonban az indexelés- és lekérdezés tervezés során felmerülő problémák miatt nem lehet minden összetett kulcsot megszüntetni, az a használhatóság és karbantarthatóság rovására menne. A tervezés eredményeképpen a következő szerkezet állt elő:

7. ábra – Végső adatbázis szerkezet



A tartalmaz tábla (*consist*) tárolja a hierarchia rendszert az elemek között. Egy a többhöz kapcsolatot biztosít. Az adatbázis szerkezet megengedné bármilyen hierarchia szintek egymásba ágyazását, de ez akár végtelen ciklusokhoz is vezethet, így erre a programozás során figyelmet kell fordítani. Vagy egy trigger esemény kell, hogy ellenőrizze a rekordok beszúráskor, hogy ez a probléma nem áll fenn, vagy a backend szoftvernek kell ellenőriznie a query futtatása előtt. Ekkor egy lehetséges megkötés definiálásra is került az adatbázissal kapcsolatban: trigger események kezelése. Itt egy ID mező bevezetése és kulcsként használata lenne szükséges, hogy az összetett kulcsot megszűntessük. Ebben az esetben is javasolt lenne két további mező indexelése, mert ezek gyakran keresett mezők lesznek, így gyakorlati haszna az ID mező bevezetésének nincs. Hasonló képen járunk el a rendelési lehetőségeknél (*orderable*). Egy terméket egy beszállítótól csak egyszer veszünk fel az adatbázisba, így az mindenképp egyedi rekordot eredményez. Amennyiben megengedjük a kétszeri felvitelt inkonzisztens eredményeket kaphatunk a költségbecslésnél.

Az elem típus tábla (*element\_types*) tárolja ez előre definiált elemtípusokat, amely ebben az esetben a megállapodások szerint az 5 hierarchia típus. Ezek szövegkénti tárolása teljesen felesleges lenne, így segédtáblába lettek kiszervezve.

Egy elemhez két féle képen lehet fájl csatolni. Az első a fájlok (*files*) tábla használata. Egy a többhöz kapcsolattal egy, a fájlrendszerre feltöltött, fájlra mutatnak a rekordjai. A fájlrendszer és az adatbázis közti konzisztenciát sajnos az adatbázis nem tudja elvégezni, ez a backend szoftver feladata lesz. A másik mód az elem ikon (*icon*) mező használata. Mivel az elemek általában ikonjukkal együtt lesznek megjelenítve így egy külön hivatkozás mentése a fő táblába hasznos. Mivel ez egy különleges fájl, nem a gyártás része így ez a hivatkozás a fájlok táblába be sem kerül. Ebben a megvalósításban az elemek listázásakor nem kell a fájlok táblát is betölteni és használni, hogy az ikonok megjeleníthetőek legyenek, ami futásidőt takarít meg a szerver oldalon.

A BOM lista előállítását két tábla összekapcsolásával lehet kivitelezni. Az elemek (*elements*) és tartalmaz (*consist*) táblák összes eleme képes a hierarchia kialakítására. Ez egy rekurzív folyamat. Ennek megvalósítására két lehetőség áll rendelkezésre. Az elsőben a backend szoftver végzi a visszafejtést és minden rekurzív körre egy új query kérést indít. Ez nagyban megnöveli a futási időt és igénybe veszi a szervert, ami kevesebb egyidejű felhasználót jelent. Ámbár ennek előnye, hogy az adatbázisnak nem kell támogatnia a rekurzív lekérdezéseket. A második lehetőségnél a rekurzív kifejtést maga az adatbázis végzi el és a backendnek a teljes visszafejtett listát adja. Ez backed oldaltól szebb és átláthatóbb kódot eredményez. Ennek választása esetén viszont ez egy újabb megkötés az adatbázis motorra nézve.

* + 1. Adatbázis méretezése

Az előzetes egyeztetés alapján kiderült, hogy az adatbázis mérete nem lesz túl nagy. Maximum 1000 darab elem tárolására lesz szükség. Minden elemhez 1-2 fájl feltöltése lesz szükséges. Ezek alapján a legnagyobb táblázat a raktárkészlet lesz, ami visszamenőlegesen a teljes bevételi és eladási láncot tartalmazni fogja. Mivel nagy tételben vásárolnak így ez is évek alatt éri majd el az ezres nagyságrendet. Párhuzamos igénybevételre van szükség, mivel több felhasználó is használhatja egyszerre. Ám szerkeszteni ritkán szerkeszti több ember egyidőben. Mivel a lehetőséget meg kell adni, ezért ezzel is számolni kell a kivitelezés során és a kritikus műveleteket tranzakciókba kell szervezni. Az olvasás azonban korlátlanul megengedett több felhasználónak is. Előzetek kalkulációk alapján maximum 3-4 ember fogja egyszerre használni olvasásra az adatbázist.

* + 1. Különlegesen terhelt időszakok

Egy új termék bevezetésekor számos rekord felvitelére kerül sor. Ezt akár több mérnök is végzi egyidőben. Ekkor az adatbázis konzisztenciájára tekintettel kell lenni, és számolni kell a szükséges erőforrásokkal. Mivel ekkor a mérnökök célja a BOM lista digitalizálása, így ebben az időszakban az adatbázis rengetegszer fogja előállítani a rekurzív lekérdezést. Ez különösen terhelt időszak lehet. Mivel ez csupán a kezdeti adatbázis feltöltéskor fordul elő, így egy virtuális környezet alkalmazása könnyen bővíthető erőforrásokat kínál erre a rövidebb időre.

* + 1. Adatbázis rendszer kiválasztása

Az eddig említett adatok alapján négy megkötésnek kell, hogy megfeleljen a használandó adatbázis.

1. Django támogatás
2. Trigger utasítások használata
3. Rekurzív query lekérdezés futtatása
4. Linux támogatás

A megvizsgálandó szoftverek a lehetséges alkalmazásra: PostgreSQL, MySQL, MariaDB, SQLite. Ezekben a technikákban van jártassága az ottani üzemeltetésnek. Az összehasonlítást az alábbi táblázat tartalmazza:

4. táblázat. Adatbázis összehasonlító táblázat

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | Django támogatás | Trigger kezelés | Rekurzív query | Linux támogatás | Ár | Teljesítmény |
| PostgreSQL | natív | 6.0-tól | 8.4-től | natív | ingyenes | Jól teljesít komplex lekérdezéseken és nagy adathalmazokon |
| MySQL | natív | 5.0-tól | 8.0-tól | natív | ingyenes, de az Enterprise verzió jobb teljesítményt kínál | gyors lekérdezéseken és rövid írásokon jól teljesít, de komplex feladatokban a PostgreSQL jobb |
| MariaDB | natív | 5.1-től | 10.2-től | natív | ingyenes | jó teljesítmény olvasás kritikus alkalmazásokban, könnyű replikációt biztosít |
| SQLite | natív | 3.6-tól | 3.8.3-tól | natív | ingyenes | gyors, egyszerű, egyszálas alkalmazás ami nem ideális több felhasználós alkalmazásokhoz |

Az összehasonlítás eredményeként a MariaDB és a PostgreSQL közötti választás maradt az utolsó lépésként.

5. táblázat. Végső összehasonlítás

|  |  |  |
| --- | --- | --- |
| **Jellemző** | **PostgreSQL** | **MariaDB** |
| **Teljesítmény** | Gyors komplex lekérdezések és analitika esetén | Gyors **olvasás-kritikus** és **egyszerű lekérdezések** esetén |
| **ACID kompatibilitás** | Teljes ACID-kompatibilitás | Teljes ACID-kompatibilitás, InnoDB használata |
| **Skálázhatóság** | Vertikális és horizontális skálázás, replikáció | Horizontális skálázás |
| **SQL támogatás** | Támogatja a **teljes SQL szabványokat**, fejlett lekérdezésekkel | MySQL konvenciókat követ, de fejlesztésekkel |
| **Telepítés és használat** | Bonyolultabb, több funkcióval rendelkezik | Könnyebb telepítés és beállítás a MySQL felhasználók számára |
| **Felhasználási esetek** | Vállalati alkalmazások | **Webalkalmazások**, **magas forgalmú oldalak** |

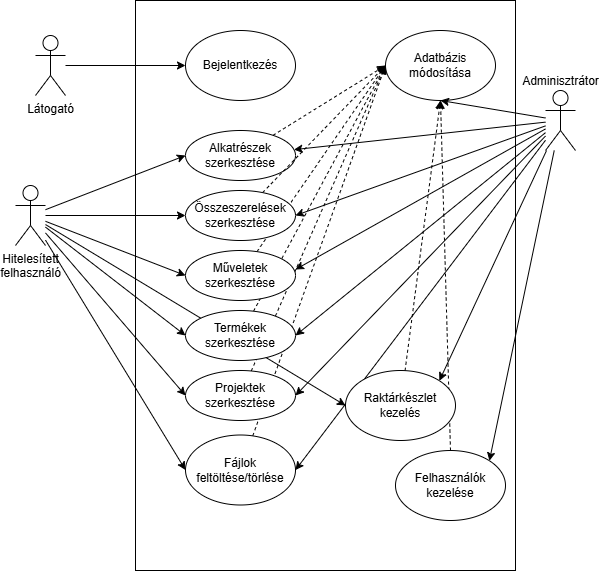
Ezen összehasonlítás eredményeképpen a használandó technológia a MariaDB. Ebben az alkalmazásban inkább olvasás kritikus lekérdezések lesznek használva. Fontos a könnyű telepítés és karbantartás, valamint gyakran ezt használják ehhez hasonló webalkalmazásokhoz.

* 1. Felhasználói felület tervezése

A felhasználói felület egy webszerver által szolgáltatott felületként érhető el bármely böngészőből. Alapvetően számítógépes/laptopos felhasználásra tervezendő, de alkalom adtán mobileszközről is elérhetőnek kell lennie. A felhasználói felületet három részre bonthatjuk felhasználói jogosultság szerint.

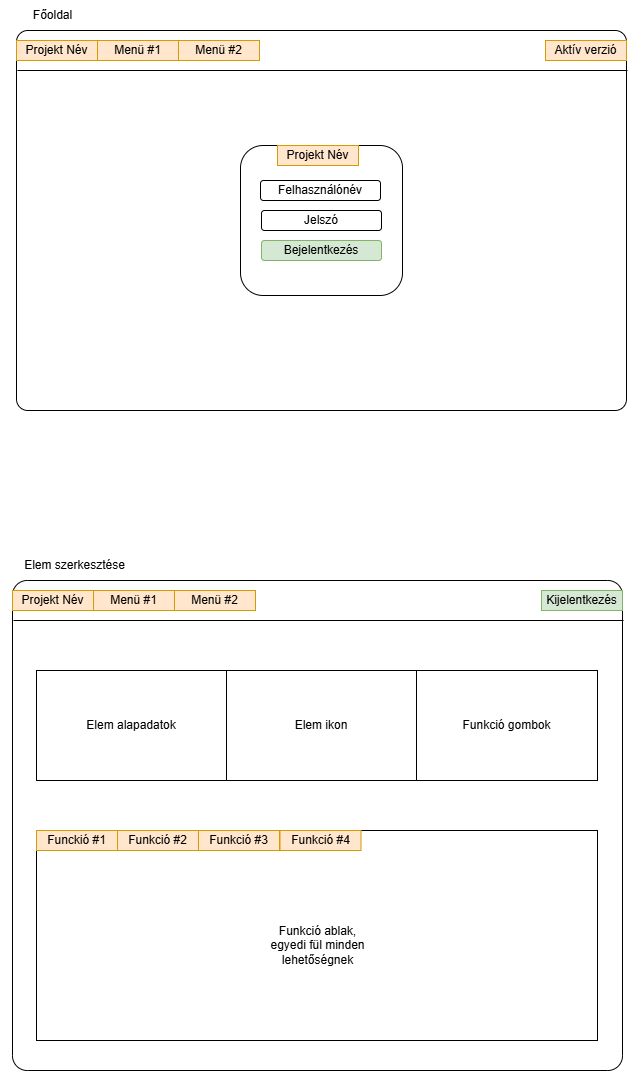
* Látogató – nem hitelesített felhasználó
* Hitelesített felhasználó
* Adminisztrátor

A látogatóknak korlátozott hozzáférése van. Csupán a bejelentkező képernyőt és egy rövid leírást találnak a projektről. A bejelentkező képernyő egy egyszerű form, mely középen két szövegbeviteli mezővel engedi hitelesíteni a felhasználót és a felső sorban található egy vízszintes menü, amely a projektismertető oldalra való átirányításra használható.



8. ábra – UML használati eset diagram

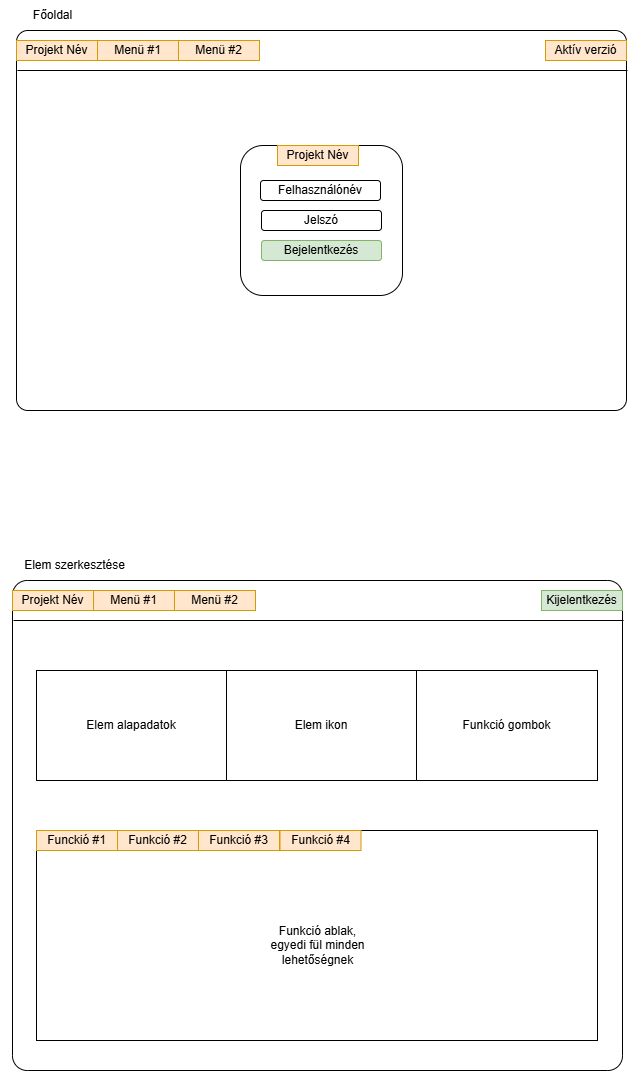
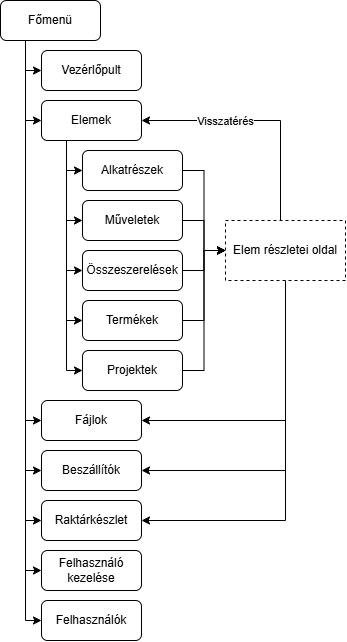
Egy bejelentkezett felhasználó képes böngészni az adatbázisban szereplő elemeket. A főbb funkcionalitást az „Elem részletei” oldalról éri el. Ez az oldal megjeleníti az elemhez tartozó összes tulajdonságot és szerkeszthető formában biztosítja a felhasználónak. Mind az öt különböző elemtípus (alkatrész, művelet, összeszerelés, termék, projekt) ugyan azon az oldalon jelenik meg, ám a funkció gombok eltérhetnek a típus függvényében. Ennek előnye, hogy egy sablon oldal használata elegendő, ami jelentős kódolást takarít meg.

Az „Elem részletei” oldalon a felhasználó képes kell legyen:

* Fájl hozzáadni az elemhez
* Ikont módosítani
* BOM hierarchiát megjeleníti/módosítani amennyiben ez lehetséges

9. ábra – Főoldal terve

* Egy nem alkatrész elemtípusnak költség és munkaerő igényt becsülni

Ehhez egy függőlegesen két részre osztott képernyőkép lenne ideális. Ennek felső része további három részre osztása ellátja az alap adatok megjelenítése, az ikon megjelenítése és a funkciógombok elhelyezése feladatokat. Az oldal alsó részén egy több füllel szétválasztott kezelőfelület helyezkedik el. Itt minden jelentős funkcionalitás egy dedikált fület kap, amelyen csak a saját kezelőelemeik jelennek meg.

10. ábra – Elem részletei oldal

11. ábra – Navigációs diagram

Ezen túl további oldalak létrehozása szükséges mellékes funkciók megoldására, mint a készlet bevételezés és kiadása. Ezen az oldalon van lehetőség megvásárolt alkatrészek adatbázisba történő felvitelére. Ezek alapján képes az adatbázis követni, hogy melyik alkatrészek állnak raktáron.

A felhasználók kezelésére a Django egy beépített adminisztrációs oldala áll rendelkezésre. Ezen van lehetőség mind a két felhasználói jogosultsággal rendelkező felhasználókat létrehozni. Mivel ez a funkcionalitás nem napi szintű használatban van, ezért ennek a menübe való megjelenítése nem javasolt.

* 1. OOP modellek tervezése

Az objektum orientált programozás [13] számos előnyét lehet felhasználni ebben a projektben. Megvalósítás szintjén egy ORM rendszerhez nagyon hasonló struktúra a kedvező megoldás. A szükséges objektumok önállóan kommunikálva az adatbázissal olvasnak, létrehoznak, törölnek és módosítanak rekordokat. Ez ebben a formában megvalósítja a C.R.U.D. alapelveket. [14] Azonban ezeknek a kibővítése szükséges az egyedi funkcionalitásokkal. Egy-egy objektum olvasásához akár több tábla összefűzése is szükséges lehet. Az olvasási művelet azonban önmagában nem okoz semmilyen formában verseny helyzetet az adatbázist aktívan használók körében. A teljes adatbázis struktúrát három osztály implementálásának köszönhetően tudjuk kezelni:

1. *Elemek* osztály – Ez az objektum kezel minden típusú elemet. Ez végzi az adatbázissal való kommunikációt és készíti a BOM listát is.
2. *Media* osztály – Ez az objektum önmagában egy, a szerveren lévő, fájlt reprezentál. Ennek a segítségével lehet fenntartani a fájlrendszer és az adatbázis közti konzisztenciát. Az objektum képes a fájlrendszerről törölni fájlokat.
3. *Beszállító* osztály – Egy beszállítót reprezentál. Ennek segítségével az elemek objektum listában tudja tartani a beszállítókat a könnyebb kezelés végett.

A megrendelő anyacégére való tekintettel a teljes forráskódban a változónevek és függvény/metódusnevek angol nyelven kerültek megfogalmazásra.

* + 1. Elemek osztály

Az *elemek* osztály tekinthető a kód alapvető elemének. Ez tartalmazza a másik két objektumot (vagy listákat belőlük). Egy ilyen objektum létrehozásakor egy adatbázis művelettel lekérdeződik néhány alapvető paraméter, ami szükséges az inicializáláshoz. Ezután az összes belső paraméter @property taggel lett ellátva, ami azt eredményezi, hogy valós időben a paraméter lekérdezésekor történik meg az adatbázisművelet. Ennek a megoldásnak előnye, hogy minden esetben valós idejű adatot szolgáltat a felhasználói felület felé. Hátránya a reakcióidő csökkenése. Több elem egyszerre történő kezelésekor kialakulhat versenyhelyzet az adatbázis oldalán, azonban ez inkonzisztenciához sosem tud vezetni, csupán a válaszidő növekedéséhez. Minden objektum külön adatbáziskapcsolattal rendelkezik. Ennek eredményeképpen egy olyan tranzakció nem valósítható meg ebben a struktúrában, ami több elemet egyszerre érint. Ez problémát okozott a rekurzív műveletek visszavonásakor hiba esetén, de probléma enélkül is megoldható volt.

Ez az osztály tartalmazza az SQL lekérdezéseket saját magára és a hierarchiára való tekintettel is. Minden lekérdezésnél figyelem lett fordítva arra, hogy az elem ne tudjon olyan adatokhoz hozzáférni, ami nem hozzá kapcsolódik. Ez helyenként többszörösen összefűzött WHERE záradékokat jelent, hogy el lehessen érni a kívánt eredményt. Az itt megjelenő függvények natív python listákat vagy szövegeket adnak eredményül, kivéve egyet.

A BOM hierarchia kirajzolásához használt kódrészlet megvalósításához elengedhetetlen volt, hogy a függvény kimenete HTML kód legyen, így ez egy kivételként rögtön HTML kódot állít elő.

Az adatbázis tervezésekor megvizsgált kétféle BOM hierarchia előállítási módok ebben az osztályban kerülnének implementálásra. Mivel a vizsgálat eredményeképpen az egyik túl sok felesleges hálózati forgalmat generál így annak a kódnak a végső verzióban helye nincs, csupán a teszt idejére és a teljesítményméréshez volt rá szükség. A teljesítmény teszt eredménye szerint a végső implementált algoritmus, mely tartalmazza a query hívást és visszatérésének értelmezését is, körülbelül 8-10 milliszekundum alatt képes felépíteni egy projekt teljes hálóját egyetlen hálózati üzenetváltásból.

Az összes elem egy listában történő lekérdezéséhez is implementálva lett egy függvény. Azonban ez az elemekhez csupán az elengedhetetlen tulajdonságokat tölti be, egy SQL lekérdezéssel az összeshez. Ezzel erőforrást takarít meg. Hátránya, hogy habár az objektum létrejött és látszólag tartalmaz minden adatot, az adatok valós időben fognak betöltődni az adatbázisból amikor arra a megjelenítéskor szükség van.

Az osztály belső változóit, eljárásait és függvényit az ehhez tartozó melléklet tartalmazza.

* + 1. Media osztály

A szoftver egyik alapvető funkcionalitása a fájlok tárolása és rendszerezése. Ehhez két nyilvánvaló megvalósítási út létezik.

* Fájlok tárolása adatbázisban BLOB mezőben
* Fájlok tárolása a fájlendszeren és a hivatkozás tárolása az adatbázisban

Abban az első lehetőségnek előnye, hogy egyszerű a megvalósítás. Nincsenek szinkronizációs problémák, mikor a rekordok és a valós fájlrendszer nem egyezik. Olvasni is könnyebb belőle, mert egy SQL lekérdezéssel megkapjuk a teljes fájl tartalmát. Ennek ellenére a hátrányai arra késztetik az iparági szereplőket, hogy a második megoldást részesítsék előnyben. Abban az esetben, ha BLOB mezőket használunk az adatbázisban akkor be kell vezetnie az adatbázismotornak a pointereket, melyek nagyban megnehezítik a keresést. Nem lehet majd előre becsülni egy-egy lekérdezés idejét, mert az nagyban függ a merevlemezen történő szektorkeresési időtől. A merevlemezen történő adatolvasás legszignifikánsabb ideje az, ameddig az olvasófej pozíciót vált és megkeresi az olvasandó szektorokat. Ez a probléma orvosolható lenne SSD meghajtók alkalmazásával, amiben kialakításából és működéséből adódóan nincs mozgó alkatrész és minden szektort logikai áramkörök segítségével ér el. Ebben az esetben sajnos számolni kell az SSD véges írási kapacitásával.

Az ebben a projektben résztvevő szerverek merevlemezzel rendelkeznek. Ezt figyelembe kellett venni a tervezéskor.

A fájlok adatbázisban történő tárolásának másik hátránya az adatbázis méretének folyamatos növekedése, amely idővel nehézkessé tudja tenni a kezelését, biztonsági mentését és visszaállítását. Abban az esetben, ha csak egy fájlnév referenciát tárolunk az adatbázisban akkor az elérési út elfér egy fix hosszúságú VARCHAR mezőben, amivel a motor könnyen számol, hogy hol a mező eleje és vége. A tárolt fájlokat pedig egy biztonsági mentés alkalmával egy zip fájlba mentve könnyen archiválhatjuk az adatbázis mentés mellé. Ebben az esetben fontos, hogy az adatbázis snapshot és a fájlok kimentése egyszerre kell, hogy történjen a szinkronizáció megtartása végett és a fájlok együttesen tesznek ki egy biztonsági mentést.

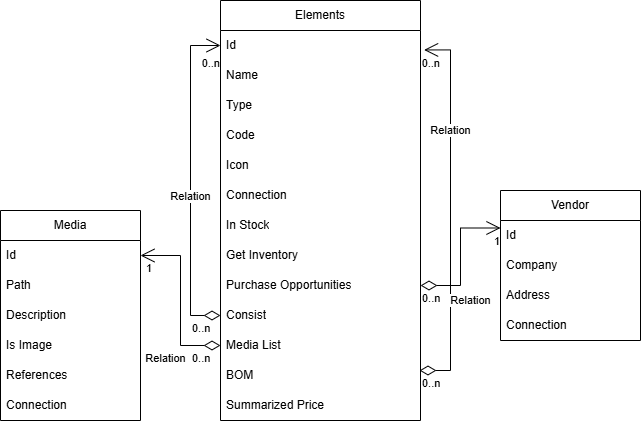
The blobs we store into the database are also generally inefficient because you are blowing up your database cache on raw data that is using up space. As time goes and your cache gets filled, it can create bottlenecks very quickly. Instead, we should delegate caching to a separate intermediate layer like Redis and cache the actual image and not just the blob. With this, we can avoid overloading the database, but the goal here is the abstraction since we won’t care what kind of persistence we implement with this new layer in-between. [15]

A *Media* osztály teremti meg a kapcsolatot a fájlrendszer és az adatbázis között. A benne lévő függvények és eljárások biztosítják a folyamatos szinkronizációt. Az osztály szerkezete rövid, mert csak néhány alapvető funkció kidolgozása volt szükséges, mint a fájlok elemekhez történő csatolása adatbázis szinten, vagy a fájlok törlése egyszerre adatbázis és fájlrendszer szinten.

A fájlok tárolása nem a forráskód számára fenntartott mappában történik. Ez egy globális változóban definiált helyen valósul meg. Ezt a változót a settings.py fájl tartalmazza, mint minden egyéb konfigurációs paramétert.

Az osztály belső változóit, eljárásait és függvényit az ehhez tartozó melléklet tartalmazza.

* + 1. Beszállító osztály

Minden megvásárolható terméknek több beszerzési lehetősége is lehet. Ezekhez különböző árak, pénznemek és beszállítói cégek tartozhatnak. Az adatbázis mikor árat becsül akkor ezek az árak közt próbálja a legoptimálisabb megoldást kínálni. Különböző pénznemek közt nem képes átváltani, azt mindig a rögzített pénznemben jeleníti meg. Ez a struktúra előrevetíti, hogy az elemekhez tartozó beszállítói lehetőség nem egy egyszerű változó vagy struktúra lesz, hanem egy lista, amely az összes lehetőséget tartalmazza. Ennek folyamodványaképpen lett létrehozva ez az osztály. Ez nagyon egyszerű műveleteket tartalmaz. Képes elemekhez beszállítói lehetőségeket adni, módosítani vagy törölni is. Extra funkcionalitásképpen ebben a fájlban kapott helyet egy segédfüggvény ami az összes beszállító adatait lekérdezi az adatbázisból és a felhasználó felületnek biztosítja megjeleníthető formátumban. Habár ez nem az osztály része, de logikailag ide tartozik.

12. ábra – Osztálydiagram

Az osztály belső változóit, eljárásait és függvényit az ehhez tartozó melléklet tartalmazza.

* 1. Tesztelés tervezése

„Testing can prove the presence of errors, but not their absence.” [16]

A szoftverfejlesztés lépéseinek fontos eleme a tesztelés elvégzése. Ezeket a tesztelési eljárásokat előre meg kell határozni, így a fejlesztés során idő takarítható meg. A tesztelési technikák két fő csoportba csoportosíthatók:

* Statikus elemzési módszerek
* Dinamikus tesztelés

A statikus elemzési módszer a forráskódra fókuszál és nem a futtatott programra. Ez két részre bontható az alapján, hogy ember vagy gép végzi. Programozás közben a fejlesztői környezet nagysegítségre van ennek a tesztelésnek az elvégzésében. A fejlesztői környezet által kínált elemzés az automatizált módszerek közé sorolható. Ez a szintaktikai és egyszerűbb szemantikai hibákat képes kiszűrni. Képes figyelmeztetni egy nem inicializált változóra, vagy egy létrehozott ám de sose használt kódrészletre. Azonban ennek a képessége határolt. Nem képes a logikát felismerni a kódban és értelmezni azt. Így a bonyolultabb hibák rejtve maradnak. Azok csak emberi átvizsgálással fedhetőek fel. Ezért javasolt időről időre egy független csapattal átvizsgáltatni az újonnan írt kódrészleteket, hogy elkerülhetővé váljanak a későbbiekben a teljes kódátírással történő hibajavítások (refaktorálások).

A dinamikus tesztelési elemzéseknél elengedhetetlenné válik az automatizálás. Általánosságban véve ezek a tesztelések két részre bonthatók az alapján, hogy ismert-e a forráskódja az alkalmazásnak. Ezek alapján lehet fekete vagy fehér dobozos tesztelés. Esetünkben a forráskód teljes mértékig ismert, így fehér dobozos tesztelésről beszélünk. [19] Ilyen esetben van lehetőségünk a forráskódban lévő minden kisebb elemre kiterjedő külön teszt eljárást létrehozni. Ideális esetben a tesztek megírása megelőzi a tényleges modul létrehozását és a teszt ellenőrzi a modul jó működését. Ezek a kis teszt eljárások gyakran lefuttathatók, hogy minél hamarabb kiderüljön, ha egy modulban olyan változtatás lett végrehajtva, ami egy előző funkcionalitást veszélyeztet. Ebben a projektben a tesztelés ezen fázisa is így lett elvégezve. A főbb funkcionalitások tesztjei minden egyes git feltöltés alkalmával lefutnak és emailben értesít amennyiben olyan változás lett alkalmazva, ami egy már működő algoritmus eredményeit valótlanná teszik.

A forráskód tárolásához a GitHub tárhelyszolgáltatása ad helyet. Ennek a szolgáltatásnak vannak extra használható előnyei is. Ilyen a GitHub Actions is. Ezzel a funkcióval képes a forráskódot egy előre meghatározott független futtatási környezetben tesztelni. Ehhez a mi esetünkben egy Docker konténer létrehozása és futtatása szükséges. A Dockerfájl a forráskód része. Ennek előnye, hogy a telepítési lépések abban azonnal dokumentálva vannak. [17]

A felhőben minden új feltöltött változás kivált egy trigger eseményt, ami elkészíti a független futtatókörnyezetet. Esetünkben ez magába foglalja a teljes python imterpretert, minden függőséget és az adatbázist is. Végleges futtatási környezetként nem lenne javasolt ennyi mindent egy konténerbe helyezni, de tesztelési funkciók miatt így sokkal gyorsabb a kezelése. A docker elindulása után feltelepíti a szükséges függőségeket, az adatbázist és a betölti a kezdő értékeket az adatbázisba.

Ezek után a Django beépített Unit test funkcionalitását segítségül véve a gyári test parancsot lefuttatva elindulnak a unit testek. Esetünkben a főbb funckiók tesztelése zajlik:

* Adatbázis kapcsolat ellenőrzése
* Adatbázisba történő írás és visszaolvasás
* Egy előre meghatározott projekt létrehozása a további eredmények ellenőrzéséhez

A teszt eljárást a test.py fájl tartalmazza a forráskód mappájában. A teszt hierarchia egy laptop gyártói tevékenységet szimulál. Ez magába foglalja a következő elemeket:

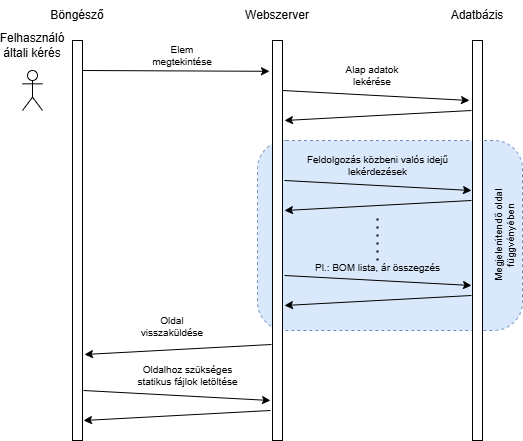
* 8db alkatrész hozzáadása
* 4db művelet hozzáadása
* 3db összeszerelés hozzáadása
* 1db termék hozzáadása
* 1db projekt hozzáadása

Ezen elemek hozzáadása után adott számú bejegyzésnek kell lennie az adatbázisban. Ha a végeredmény jó azzal biztosítottuk a teljes logika működését. Ebben az esetben képes az elemek osztály kommunikálni az adatbázissal, írni bele és létre is hozza a szükséges rekordokat. A létrehozás után pedig azok permanensek maradnak, mert egy utána történő olvasással összegezni tudjuk a rekordok számát. [18]

A szoftver tesztelését a fejlesztés befejezése után sem érdemes abbahagyni, mert frissebb külsős könyvtárak esetén az eddig működő kódrészeink eltérő eredményeket adhatnak. Ebben is nagy segítség az automatizált tesztelés.

1. Teljesítmény tesztelés

A tesztelés gyakorlati kivitelezése során a szoftver különböző komponenseinek független és önálló tesztelése elengedhetetlen. A szoftver felhasználói felületének terhelés tesztje szükséges, de önmagában nem elegendő eredménnyel szolgál. Annak az eredménynek a tudatában még nem vagyunk tisztában a teljes adatelérési- és megjelenítési lánc leggyengébb pontjával.



13. ábra. Tesztelés közben megvizsgálandó csomópontok

A 12. ábra mutatja egy tetszőleges elem megjelenítését a felhasználói oldalon. A felhasználó által indított GET kérés a webszerver felé egy, a felhasználó által nem észrevehető, adatbázis lekérdezési sort indít el. Ennek a sornak a hossza függhet az elem típusától, a hozzá kapcsolódó fájlok számától és a egy hierarchia közepén attól is, hogy hány elem tartozik az elem alá. Ennek az időigénye előre nem meghatározható, de becsülhető. Amint az adatbázis ellátta minden információval a webszervert, az visszaküldi a HTML választ a böngészőnek. Ekkor a megjelenítő motor elkezdi feldolgozni a kapott kódot és megkezdi a statikus fájlok beszerzését a merevlemezről, vagy a webszerverről. Ez a kialakítás azért előnyös ebben a formában, mert a statikus fájlok nem az adatbázisban vannak tárolva, így nem kell minden alkalommal letölteni őket. A webböngésző natív cache funkcióját kihasználva hálózati terhelést spórol meg a felhasználó, ami időnyereségként jelentkezik a felhasználói élményben.

A 12. ábrán jól látható, hogy a teljesítménytesztelés kivitelezését kettő, önállóan és függetlenül kivitelezett folyamatra bontható.

* 1. Böngésző – Webszerver kapcsolat

A böngésző által küldött üzeneteket a python által futtatott webszerver feldolgozza és válaszol rá. A tesztelés ezen pontján fontos, hogy olyan oldal renderelésére kerüljön sor, aminek eléréshez nincs szükség adatbázisra. Ezzel tudjuk tesztelni a hálózat és a webszerver teherbírását.

A python programozási nyelv számos előnye mellett egy nagyon fontos hátránya is megjelenik, ami a tényleges párhuzamos futtatás natív hiánya. Sajnos ebben a nyelvben egy párhuzamos szál nyitása és futtatása nem egy operációs rendszer szintű szál nyitását eredményezi. Ebben az esetben a python futtatókörnyezet, ami önmagában egy szálon fut, szimulálja a párhuzamos szálat. [20] Ez soha nem lehet olyan hatékony és gyors, mintha az operációs rendszer kezelné a párhuzamos műveleteket. Erre a python megoldása a multiprocessing bevezetése. A tényleges párhuzamosításhoz ebben az esetben külön folyamatként képes futtatni eljárásokat. Ez már tényleg gyors és hatékony megoldás, habár programozói oldalról extra munkát igényel az IPC megvalósítása.

Emiatt a Django keretrendszer is sajátosan kezeli az egyidőben érkezett több kérés feldolgozását. Alapvetően kínál egy könnyen és gyorsan indítható saját webszervert. Ez egyszálú és a konkurens kérések, habár feldolgozásra kerülnek és ki lesznek szolgálva, de valójában egy szekvencia mentén kerülnek feldolgozásra. Erre a megoldás egy külsős modul alkalmazása, amit Gunicorn-nak neveznek. Ez képes natív folyamatokra lebontani a párhuzamos kéréseket és gyorsabban kiszolgálni, mint egy natív WSGI alkalmazás. [21] A teszt során az egyszálú kiszolgálás mérése van előtérbe helyezve, mert ennek a rosszabb a teljesítménye. Amennyiben ez elegendő teljesítményt biztosít, úgy a valós környezetbe a gyorsabb megoldást helyezve biztos a zavartalan működés.

Egy weboldal terheléstesztelésére számtalan megoldás létezik. Még a Django keretrendszer is kínál saját megoldást. Ennek ellenére én egy független alkalmazását részesítek előnyben. A Locust nevű alkalmazás egy nyílt forráskódú terhelési platform. Előre definiált szkriptek futtatását teszi lehetővé weboldalakon és ezeknek a metrikáját egyhelyre gyűjtve egyszerű eredményt mutat az összesített teljesítményről. Ez az alkalmazás képes egyszerre több kliens szerepét is betölteni és egymás után több iteráció futtatására, hogy az eredmények átlaga pontosabb és valósabb képet mutasson.

Első lépésként egy felhasználót szimulálva és 10 egymást követő oldal lekérése azt mutatja a teszt környezeten, hogy egy oldal betöltése ~100 milliszekundumot vesz igénybe. Ebben az esetben a cache ki van kapcsolva teljesen, hogy a lehető legerőforrásigényesebb lehetőséget mérjük. Ezt az eredményt a kliens számítógépen egy egyszerű böngésző fejlesztőpaneljén validálva valós eredménynek tűnik. Ott annyival több információhoz jutunk, hogy ennek a felosztása 81-19 arányban az oldal megjelenítése és nem a hálózati átvitel. Ez fejlesztői oldalról megnyugtató, mert a megjelenítési idő csakis kizárólag a kliens számítógép teljesítményétől függ. Ez azt jelenti, hogy a webszerver wifin keresztül elérve a klienst 19ms alatt képes egy statikus weboldalt feldolgozni és elküldeni. Abban a pillanatban, hogy a felhasználók számát növeljük ez a feldolgozási idő növekszik. Erre a használt technika korlátozottsága ad választ. Az előzetes felmérés során megállapításra kerültek a kritikus időszakok. Aszerint a legkritikusabb felhasználás az adatbázis feltöltése vagy egy új termék bevitele. Ekkor legfeljebb 10 felhasználó használja a rendszert az átlagos 3-4 fővel szemben. A tesztkörnyezetet 10 kliens számítógépre állítva az oldal megjelenítési ideje 140 milliszekundumra emelkedett. Ez az érték teljesen elfogadható, tekintve, hogy a végső telepítési helyen egy másik technikát használva sokkal gyorsabb elérése lesz.

* 1. Webszerver – Adatbázis kapcsolat

A webszerver és az adatbáziskapcsolat tesztelése sokkal determinisztikusabb eredményekkel szolgál, mint a kliens számítógép-webszerver kapcsolat. Ebben az esetben a kapcsolat docker konténerek közti kapcsolat. Ez legrosszabb esetben is vezetékes hálózaton történik meg, ha a konténerek különböző fizikai szerveren vannak megvalósítva. Esetünkben a konténerek egy fizikai szerven helyezkednek el, így a kettejük közti hálózati kapcsolat késleltetésmentesnek tekinthető.

Egy adatbázis teljesítményét két fő összetevő határozza meg. A kapcsolat milyensége és az adatbázis olvasási/írási sebessége. Miután esetünkben a kapcsolat teljesítménye statikusnak tekinthető ezért azzal nem is érdemes számolni. Sokkal fontosabb tényező egy lekérdezés feldolgozási és megvalósítási ideje. Ennek tesztelésére egy, vagy akár több különböző SQL lekérdezés futtatására van szükség. Egy ilyen teljesítményre való hozzávetőleges eredményének becslésére a phpMyAdmin [25] felület is képes. Ez a felület volt használva a fejlesztés közben, így a tesztelésbe való bevonása kézenfekvő. Itt minden SQL query futtatási ideje megjelenik. Ez relatív arányszámként használható, hogy mennyire bonyolult egy lekérdezés. Fontos ebben az estben is, hogy a legteljesítmény-igényesebb kérésekre kell optimalizálni a rendszer áteresztőképességét. Esetünkben egy egyszerűbb lekérdezési ideje, ami a teljes *elements* tábla tartalmát lekéri, 0.001 másodperc. Ez valószínűsíthető, hogy ennél gyorsabb, csak a mérés pontossága ennek megjelenítését nem teszi lehetővé.

Egy sokkal bonyolultabb lekérdezés futtatása már ennél értelmezhetőbb eredményt ad. Az összes futtatható kérés közül a legbonyolultabb a BOM lista rekurzív lekérdezése. Habár ez a rekurzív mivolta miatt szintén nem tekinthető determinisztikusnak, egy átlagos projekt méretre vetítve számolhatunk a tesztelésben vele, mint arányszám. Ennek a kérésnek a nehézsége, hogy több tábla szorzatának többszöri összefűzésére van szükség a teljes BOM hierarchia kialakításához. Egy átlagos projektre történő futtatás eredményeképpen a futtatási idő 0.006 másodperc. Ez arányszámként azt jelenti, hogy legalább 6x annyi a teljesítményigénye van, mint egy sima egytáblás lekérdezésnek.

Az eddig elvégzett mérések mind egy felhasználóra vonatkoznak. A python szekvenciális feladatvégzése miatt egy felhasználó által indított kérések soha nem lesznek párhuzamosan feldolgozva az adatbázis által, azonban több felhasználó kérése már okozhat ilyen lehetőséget. Ennek mérésére egy másik segédprogram van segítségemre. A mysqlslap program egy adatbázis terhelésére szolgál. [22] Ennek segítségével egy adott lekérdezés futtatása lesz lehetséges akár több kliens számítógépet szimulálva.

14. ábra. Az adatbázis sebességteszt eredményei

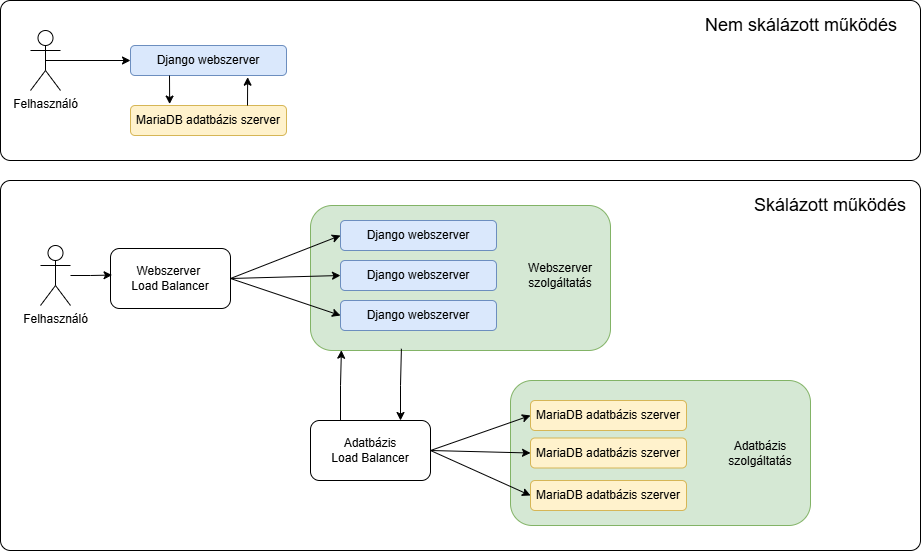
A 13. ábra tartalmazza az 1-10-20 és 100 kliensre számított adatbázis iteráció időt, adott lekérdezés esetén. Mivel ez teljesen lineáris ezért az adatbázis 100 kliens esetén is ugyan annyi időt tölt egy lekérdezés feldolgozásával, mint egy kliens esetén. Ez azt jelenti, hogy hatalmas tartalék van már a létező rendszerben is.

* 1. Teljes feldolgozási lánc tesztelése

A tesztelés utolsó lépéseként a teljes feldolgozási lánc tesztelése marad hátra. Ebben az esetben is a Locust alkalmazás lesz segítségemre. Habár, ebben az esetben nem egy statikus weboldal elérése a cél, hanem egy adatbázis eléréséhez kötött oldal megjelenítése. Ez a tesztelés méri a valós felhasználói élményt. Előzetes eredmények tudatában arra számítunk, hogy a webszerver lesz majd a legszűkebb keresztmetszet a rendszerben, mivel az adatbázis egy statikus késleltetésnek tekinthető, de a kliensek számának növelésével ez a statikus érték nem emelkedik.

Az eredmények nagyon hasonlítanak a 6.1-es fejezetben említettekhez. Itt az látszik, hogy a kliensek számának növelésével növekedik az elérési idő is és ez a növekedés nem lineáris. Szerencsére előre meg tudjuk határozni, hogy összesen hány felhasználó fogja használni a rendszert. A maximum értéket a megrendelő 10-ben határozta meg. Ekkora terhelés esetén a megjelenítési idő 200 milliszekundum alá tehető. Ez az érték teljesen folyamatos felhasználói élményt jelent és nem szükséges a további beavatkozás.

1. Esetleges áteresztőképesség növelése

Habár a jelenlegi rendszer is képes a felhasználói igények teljesítésére, szükséges előre gondolkodva egy fejlesztési terv kialakítása. Amennyiben a maximális felhasználószámot tovább emeljük, úgy lesz egyre lassabb a rendszer. Erre egy megoldás lehet a rendszerek horizontális skálázása. Ebben nagy előnyt jelent, hogy teljesen dockerizált alkalmazásról beszélünk. Egy teljesítmény elosztó (load balancer) illesztése a rendszerbe segít az áteresztőképesség növelésében anélkül, hogy a forráskódban bármi módosítást kellene végrehajtani. [23]

15. ábra. Skálázott és nem skálázott működési ábra

A skálázott működés közben a versenyhelyzet egy valós probléma lehet. Ennek a webszerver szolgáltatás esetén nagyon kicsi a valószínűsége, mivel ott az adatok inkább csak olvasva vannak. Az egyetlen írási lehetőség egy fájl feltöltése vagy törlése. Ezesetben az operációs rendszer fogja az egyik szervert visszatartani és feloldani a versenyhelyzetet. Az adatbázis esetében a MariaDB nagyon könnyen skálázható és előre fel van készítve az ilyen működésre. Ilyen esetben a load balancer feladata a szerverek közti szinkronizáció biztosítása. [24] Bizonyos esetekben a lekérdezési fa és az aktuálisan futtatott tranzakciók így is versenyhelyzetet okozhatnak, ám összességében a rendszer összesített áteresztőképessége növekedik.

1. Fejlesztés közben felismert és megoldott hibák
   1. Raktárkészlet kezelése

A szoftver egyik alapvető funkciója a raktárkészlet-kezelés. Ehhez a kezelőfelületen külön dedikált gomb van, ahol kézzel hozzá lehet adni vásárolt alkatrészeket. Az alkatrészek hozzáadása könnyű feladat, mert azok vannak a hierarchia legalján, ők nem függenek semmitől. Azonban, ha a hierarchia-szinten feljebb lévő típusú elemeket adnánk hozzá akkor egyesével meg kell vizsgálni annak minden függőségét. Ha egy termék három összeszerelésből és 1 műveletből áll akkor ennek összesen négy függősége van. Csak abban az esetben lehet a raktárkészlet-nyilvántartásban növelni az értékét, ha mind a négy függőségéből van elegendő darab raktáron. Ezzel egyidőben az összes függőségének a raktárkészletét csökkenteni kell az adatbázisban. Ezzel valósítva meg az alkatrészek fizikai vándorlását a hierarchia szinten felfelé. Ennek a megvalósítása két különböző elven is történhet.

Ha a négy függőségen egyesével végig menve ellenőrizzük a raktárkészletét, és ha van elegendő raktáron akkor azonnal csökkentjük az értéküket. Ekkor minden elemen egyszer elég végig menni, azonban, ha a harmadik elemnél járva kiderül, hogy nincs elegendő raktáron akkor az előző módosításokat vissza kell vonni.

Egy termék egy objektumként jelenik meg a python kódban, amely tartalmaz egy listát az összes őt alkotó elemről így kézenfekvő lenne az adatbázisban nyitni egy tranzakciót és egy ciklussal végig járva a listát csökkenteni minden alkotóelem raktárkészletét. Ha valamelyiknél ez nem lehetséges, akkor egy a tranzakciót rollback utasítással visszacsévélve az eredeti állapotba meg oldottuk a problémát. Ez elméleti szinten egy egyszerű megvalósítás, azonban mivel a listában minden elem egy független objektum, minden elem önálló adatbáziskapcsolattal rendelkezik. Ez sajnos nem teszi lehetővé, hogy egy rollback utasítással visszatérjünk az eredeti állapotba, mert külön kapcsolatokból érkező utasításokat nem tud egy rollback parancs visszavonni.

Másik lehetőségként a python oldalon kellett megoldást keresni. Mielőtt bármilyen módosítást végrehajtana az adatbázisban, ellenőrzi, hogy van-e gátló tényező. Amennyiben nincs akkor kiadja az összes utasítást. Ha talál gátló tényezőt, akkor nem is folytatja a keresést, csak informálja a felhasználót, hogy nem lehetséges a módosítás. Ez a megoldás biztonságos, kevéssel több kódolást igényel, azonban sajnos több adatbázis műveletet. Minden elemhez kétszer kell lekérdezést kezdeményezni. Egyszer lekérdezni és egyszer módostani. Végül ez a megoldás került megvalósításra a stabil működése miatt.

* 1. Tesztkörnyezet implementálása kódba

A tesztelési folyamatok akkor a leghatékonyabbak, ha a forráskódon semmilyen módosítás sincs az éles és a teszt verzió között. A való életben a tesztkörnyezet néhány apróbb pontban eltér a rényleges futtatókörnyezethez. A cél az, hogy a tényleges algoritmusok kódja teljesen transzparensen tudjon működni mind a két környezetben. Ennek elérésére az összes osztály úgy lett kialakítva, hogy a konstruktorában egy globális függvény meghívásával (*self.conn = get\_database\_connection()*) kér egy adatbáziskapcsolatot. A létrehozott objektum nem tudja, hogy teszt vagy valós környezetben fut. Az adatbáziskapcsolatot létrehozó függvény ellenőrzi a környezeti változókat és azok alapján hozza létre a connection sztringet.

Ezzel elérve, hogy az osztályokban futó kód nem különbözik a különböző futtatókörnyezetek között. Továbbá elérve, hogy ha valamelyik környezetben változtatni kellene, akkor azt egy helyen kell megtenni, a kapcsolatot létrehozó függvény törzsében. Ez a függvény a Django ajánlásait követve a *settings.py* fájlban található.

* 1. Böngészők cache eljárásainak támogatása

A felhasználói élmény egyik legegyszerűbb javítása, ha sikerül feleslegesen végrehajtott műveletektől megszabadulni. Fel kellett ismerni, hogy nem csak a css és js fájlok statikus fájlok, amik cache-elhetőek a böngésző által. Ugyan ilyen statikus fájlok az elemek ikonjai, képei. A fejlesztés elején erre nem fordítottam megfelelő figyelmet és a webszerver minden alkalommal behelyettesítette az elküldendő kódba a képek base64 formátumban tárolt verzióját. Ezzel többszörösen meglassítva az átvitelt. Először is a python kódnak ezt kódolnia kell, majd egy sokkal nagyobb fájlt kell átvinni a hálózaton.

Erre megoldásként egy új elérési URL-t hoztam létre. A media/<fájlnév> URL arra lett létrehozva, hogy a felhasználó által feltöltött fájlokat lehessen vele megnyitni. Ezt is csak bejelentkezett felhasználók érik el. Ezzel elérve, hogy a HTML kódba csupán hivatkozni kell egy minden elemnél a kép címére ezzel az URL eléréssel. Ekkor a böngésző eldönti, hogy megvan e neki a rövidtávú tárjában és ha igen akkor betölti onnan, ezzel nem terhelve feleslegesen a hálózatot és a szervereket. Így extra időt nyerve a back-end oldalon a base64 kódolás megspórolásával. Ez extra lehetőséget nyújtana extra paraméterek megadására, amely akár képes lenne egy eltárolt kép különböző méretekben történő visszaadására. Ez a teljesítménytesztek eredményét látva nem került implementálásra, mert van még elérhető tartalék a rendszerben.

1. Felhasználói útmutató
   1. Telepítés

A szoftver telepítésére két lehetőség van. A forráskódot közvetlenül futtatva is képes a működésre és egy telepítőfájl használata is engedélyezett.

A forráskód mappájában található *makeInstaller.sh* szkript egy önmagában futtatható állományt készít egy .deb formátumban. A telepítő elkészítéséhez szükséges, hogy a számítógépen minden szükséges függőség telepítve legyen, de a telepítő ezeket magába csomagolva a telepítőfájlban tartalmazni fogja. A telepítő az alkalmazást a „/opt/productManager” elérési útra másolja és létrehozza a „/var/productManager” mappát a feltöltött fájloknak.

A forráskód önmagában való futtatásához szükséges a függőségek kézi telepítése. A python 3.12-es verzió támogatott a rendszer által. A python csomag függőségek egy requirements.txt fájlban találhatóak, hogy a pip csomagkezelő egy utasítás kiadása után az összeset automatikusan telepíteni tudja. Ebben az esetben kézzel kell létrehozni a „/var/productManager” elérési utat a feltöltött fájlok tárolására.

* 1. Adatbázis beállítás

A rendszer igényel egy futó MariaDB adatbázist. Ennek futtatása ugyan azon a fizikai szerveren javallott, ahol a webszerver is fut. A könnyebb kezelhetőség kedvéért phpMyAdmin használata javasolt, az adatok könnyebb bevitele érdekében.

Kettő darab adatbázis létrehozására van szükség. Az egyik a Django keretrendszer futásához szükséges. A könnyen beazonosíthatóság miatt ezt „django” néven célszerű létrehozni. A benne található táblákat a forráskód mappájában egy *databases* mappán belül találja. A django.sql létrehozza a szükséges táblákat és rekordokat az adatbázisban.

A második adatbázis tárolja a termékek hierarchiáját. Ezt product\_manager néven hozzuk létre. A hozzá tartozó sql fájl az előzőleg már említett elérési úton található hozzá.

Az adatbázisok nevei, felhasználói adatai és elérési útjai az ajánlásoktól eltérhet. Minden esetben javallott ellenőrizni a gyári beállításokat, hogy a szoftver hozzáférhessen az adatbázishoz. Az ehhez szükséges kódrészlet a forráskód mappájában a productManager/settings.py fájlon belül a „*get\_database\_connection()*” függvényben található.

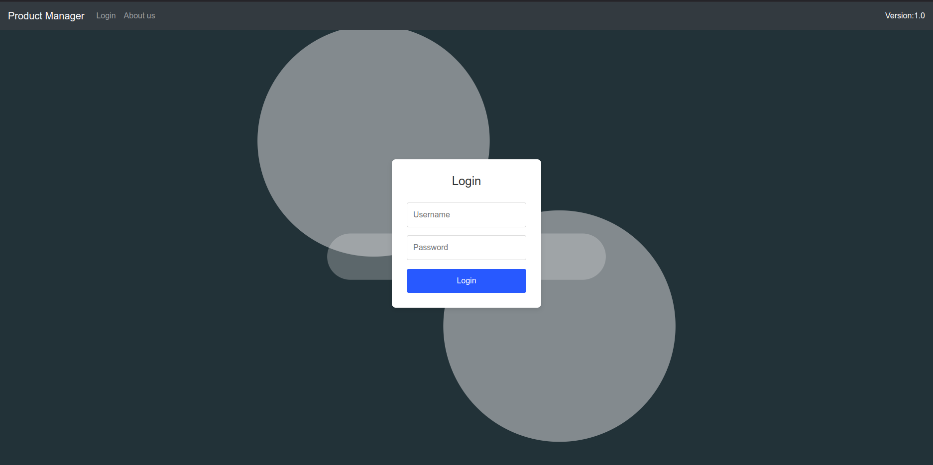
* 1. Futtatás

Az alkalmazást a manage.py fájl segítségével lehet a legkönnyebben elindítani. Az ehhez használt utasítás

*python3 manage.py runserver*

Ezután a szerver az alapértelmezett 8000 porton válaszon egy böngészőn belül.

* 1. Kezdőképernyő

A böngészőben megnyitva egy bejelentkező képernyő jelenik meg. Az ehhez használható adminisztrátori fiók előre definiálva van az adatbázisban.

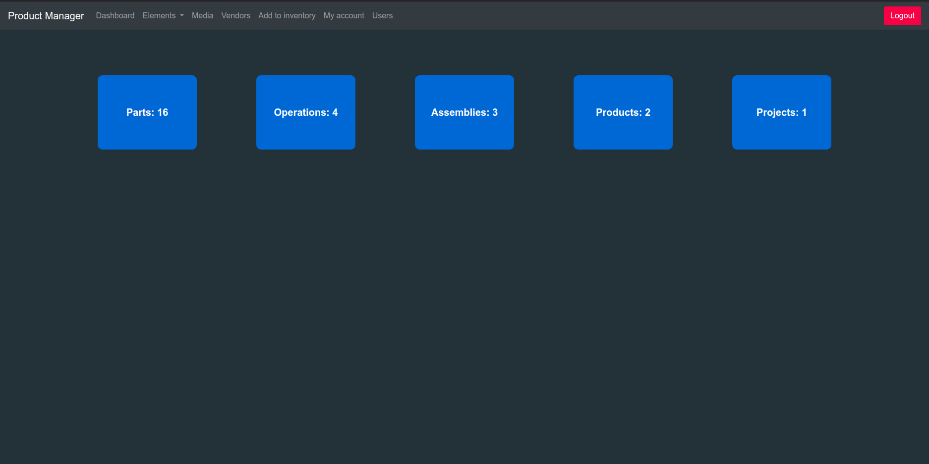
16. ábra. Kezdőképernyő

Felhasználónév: admin

Jelszó: adminadmin

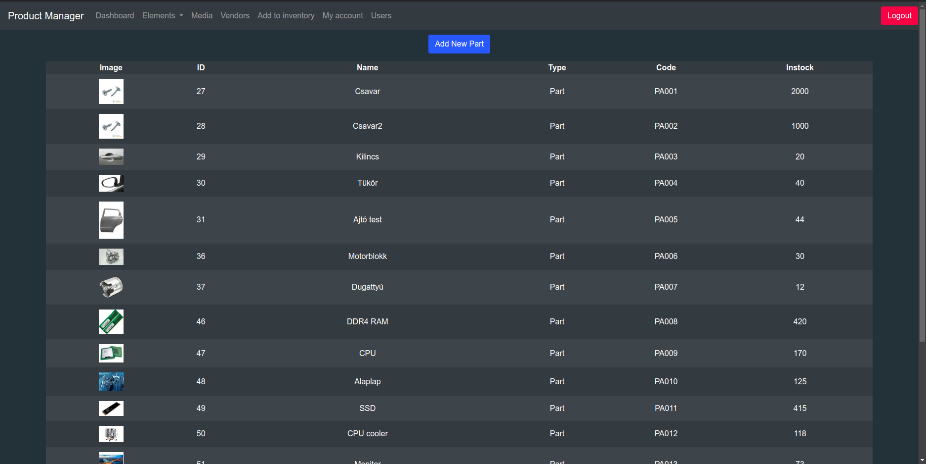
A felhasználók hozzáadásához a Django beépített felületét lehet alkalmazni. Ezesetben a hozzátartozó URL: <http://localhost:8000/admin>. Ez az URL eltérhet a személyre szabott beállításoktól függően. Ezen a felületen létre lehet hozni adminisztrátor és normál jogokkal rendelkező felhasználókat is.

* 1. Vezérlőpult képernyő

Bejelentkezést követően egy átfogó adatokkal szolgáló oldalra irányít a rendszer, ahol az adatbázisban jelenleg szereplő elemek számát jeleníti meg. Ez statisztikai adatokkal szolgál a felhasználó számára. A menürendszer vízszintes elrendezésű és a felső sorban található. A hozzá tartozó navigációs ábra a 10. ábrán látható. A kijelentkezésre a jobb felső sarokban található gomb ad lehetőséget.

17. ábra. Vezérlőpult

* 1. Termékek megjelenítése és hozzáadása

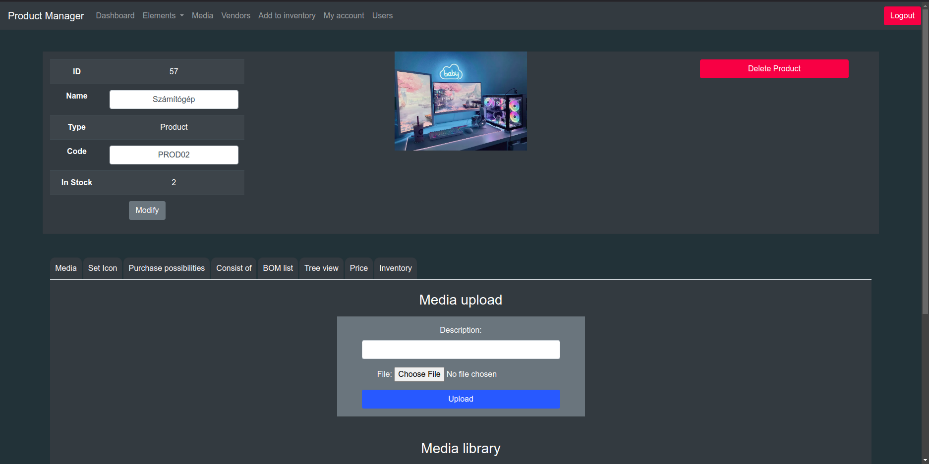
A felső menüsorból az *elements* fület választva egy legördülő menüből lehet választani a különböző hierarchia szintek közül. Egy szintet kiválasztva és rákattintva egy új oldal jelenik meg, ahol az összes, ebbe a típusba tartozó elemet megjeleníti a rendszer egy listában. A megjelenő listában található mezők:

18. ábra Elemek listázása

* Kép
* ID – az adatbázisban eltárolt egyedi azonosító
* Név – egy olvasható és könnyen beazonosítható beszédes név
* Típus – az elem hierarchiában elfoglalt típusa
* Kód – a cég belső kódja az alkatrészre vonatkozóan
* Raktárkészlet – jelenlegi raktárkészlet állapot

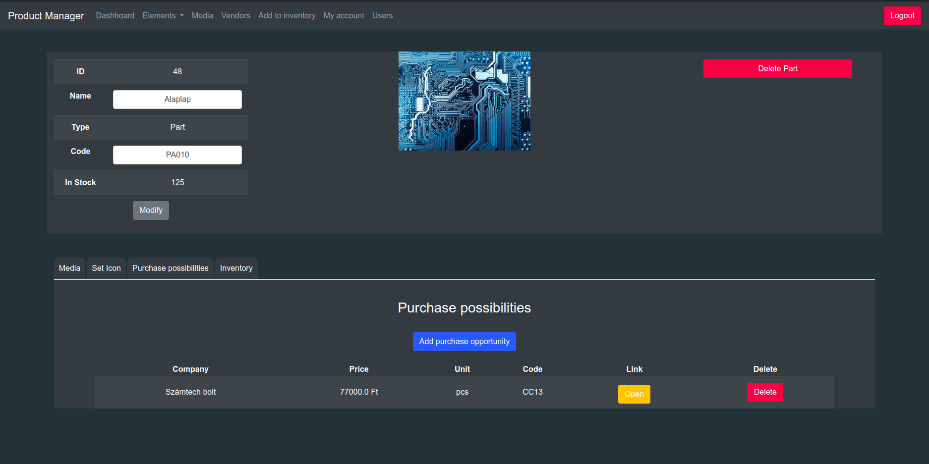
A felső sorban az új elem/művelet/összeszerelés/termék/projekt hozzáadás gombra kattintva egy új elemet rögzíthetünk fel az adatbázisban. Ehhez két mező kitöltése szükséges, a Név és Kód. Fontos, hogy a kód egy meghatározó mező, nem szerepelhet kétszer ugyan az az érték az adatbázisban. Egy új elemet létrehozva az megjelenik a listában és rákattintva lehetőség nyílik a részletei megjelenítésére és szerkesztésére.

* 1. Elem megtekintése

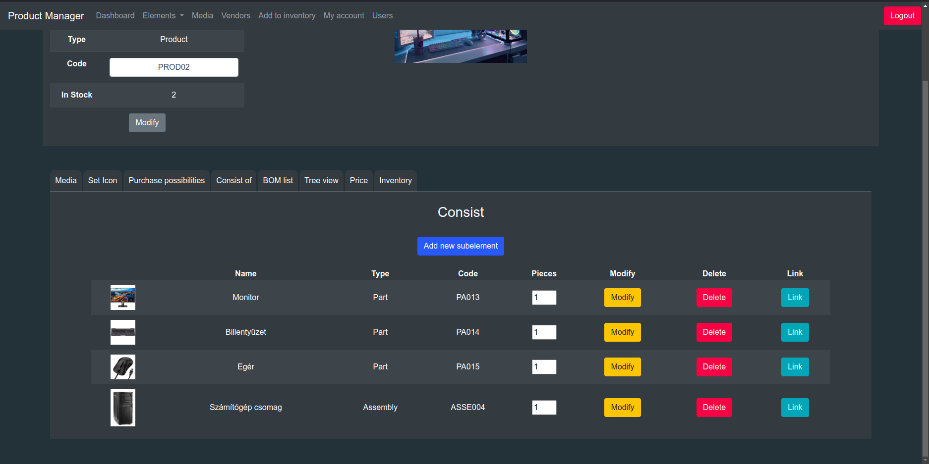
A 9.6-os pontban említett listában egy elemre kattintva megtekinthető annak részletei. Ez egy függőlegesen kettő részre vágott oldalon jelenik meg, ahol a felső oldal az általános információkra utalnak az alsó részben pedig füleken különböző műveleteket hajthatunk végre. A felső oldalon, ahol az alap adatok tekinthetőek meg ott van lehetőség az elem nevének és kódjának megváltoztatására is. Itt is fontos, hogy a kód változtatásakor nem lehet egyszerre két elemnek ugyan az a kódja a rendszerben.

19. ábra. Elem részletező

Az első oldalon fájlokat tölthetünk fel, amelyek kifejezetten ehhez az elemhez lesznek kapcsolva. Több fájl is feltölthető. A több feltöltött fájlból a képek közül lehet választani ikont az elemhez.

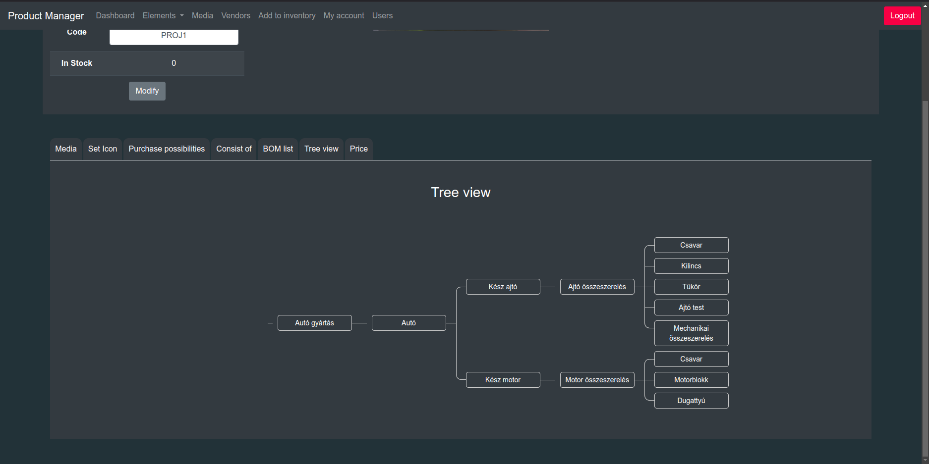
Az ikonválasztás sajátossága, hogy globális listából is választható az Ikon fülön. Ezzel elérve, hogy több elem osztozzon ugyan azon az elemen. Ha a kép törlésre kerül, akkor egy alapértelmezett kép kerül behelyettesítésre az érintett elemeknél. A harmadik oldalon vásárlási lehetőségeket lehet rögzíteni az elemekhez. Ehhez előzetesen a beszállítók menüpontban hozzá kell adni a beszállítókat. Egy termékhez több beszállítói lehetőség is rendelhető. Ez esetben válik valóssá a költségbecslés pontban a minimum és maximum ár megjelenítése egy projektre vonatkozóan.

20. ábra. Beszerzési lehetőség

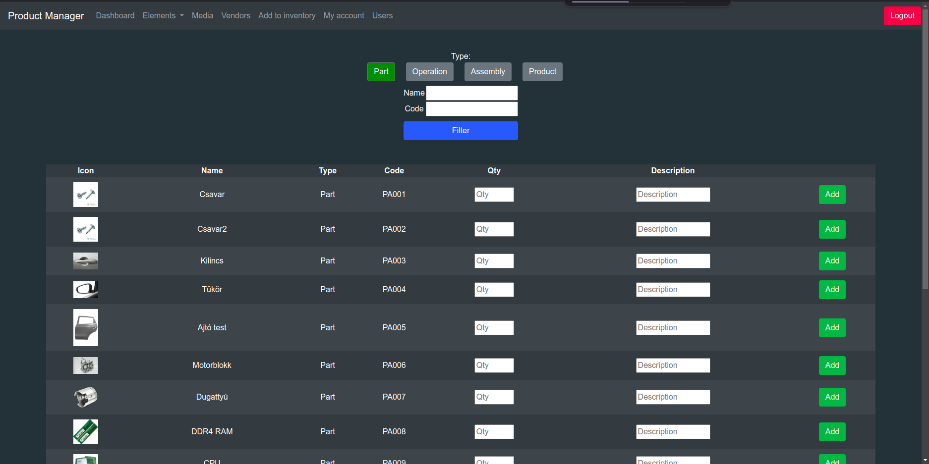
A tartalmaz menüpontban lehet kialakítani a hierarchia szintet. Fontos, hogy egy elemnél kizárólag az alatta lévő szintek megjelenítése történik. Elemek hozzáadása az erre kijelölt gombbal történik és a hozzáadást egy szűrő is segíti.

21. ábra. Tartalmaz menüpont

A BOM lista fülön a rekurzívan kibontott részletes BOM lista tekinthető meg. Mivel ez egy számított lista, ezért ez helyben közvetlenül nem szerkeszthető, csak a tartalmaz lista elérhető.

A BOM lista szemléltetésére egy fa lista nézet is elérhető az erre kijelölt menüpontban. Itt is csak a hierarchiában alatta lévő elemek megjelenítése történik.

22. ábra. Fa nézet

A raktárkészlet menüpontban lehetőség adódik kézzel bevételezni megvásárolt termékeket. Itt minden termékhez egy darabszámot és egy tetszőleges megjegyzést lehet adni a későbbi könnyebb beazonosítás miatt. Ez gyakorlatban a számla sorszáma vagy egy dátum lehet.

23. ábra. Raktárkészlet

A termékek kivételével elérhető egy Ár menüpont, ahol az elem ára és előállításához szükséges emberi munkaerő jelenik meg. Az ár minimum és maximum értékként, a munkaerő pedig órában kifejezve. Ez valósítja meg a projektekre való összeg és munkaóra becslés kritériumot.

1. Jövőbeni fejlesztési lehetőségek

Egy alkalmazás állapotára sosem lehet kijelenteni, hogy készen lenne. Minden esetben marad benne javítani vagy módosítani való. Ez az alkalmazás sem kivétel. A szoftver második fejlesztési körében több megvalósítandó fejlesztés is részt fog venni.

Egy projekthez történő automatikus megrendelés. Az adatbázis rendelkezik a jelenlegi raktárkészlet információval és a beszerzéshez szükséges cég adataival. Ettől a ponttól egy automatikus email küldése a beszállítónak egy megrendelésről egy egyszerű és hasznos lépés. Ez a kérés az ügyféllel is megbeszélés tárgya lett és kérték a fejlesztést a következő ciklusban.

Munkafolyamatok emberekhez rendelése. Ha a művelet típusként létrehozott elemek alkalmazottakhoz lennének rendelhetőek akkor mérni lehetne az aktuális munkaterhelését a munkavállalóknak. Ehhez némi átalakítás szükséges az adatbázison, de megvalósítható kérés.

Kereső mező az elemek oldalon. Az elemek oldalon jelenleg egy listaként jelenik meg az összes adatbázisban létező rekord az adott típushoz vonatkozóan. Ebben a keresési lehetőség jelenleg nincs megoldva a rendszer által, csak a böngészőbe épített keresés funkcióval. Egy beépített kereső hasznos lehet, ha már több 1000 bejegyzéssel rendelkezik majd az adatbázis.

Raktárkészlet figyelmeztetés. Ha az adatbázisban lévő elemek raktárkészlete egy érték alá csökken, akkor figyelmeztetést küldhetne az illetékeseknek, hogy beszerzést kell indítani.

1. Bevezetés utáni tapasztalatok

A bevezetési folyamatot megelőzte a telepítési időszak. Ekkor az ügyfél informatikai csapata megkapta a szükséges dokumentációt, amely alapján el tudták készíteni a futtatókörnyezetet. Ez a gyakorlatban konténerizált környezetben valósult meg.

A már meglévő adatokat, amelyek excel táblázatok formájában voltak elérhetőek, egyedi szkriptek segítségével lehetett átmozgatni az új rendszerbe. Ennek segítségével az emberi erőforrást minimalizálni lehetett és összesen néhány napnyi ellenőrzést követően egy árnyék rendszerként üzembeállt a rendszer. Ebben az időszakban több erőforrást igényel a rendszer, mert egyszerre kell kezelni a régi és az új rendszereket is. Ennek ellenére ez a legbiztonságosabb módja, hogy minden algoritmus és adat migrálási probléma véglegesen validálva legyen az ügyfél által is. Ebben az időablakban volt a legkézenfekvőbb az új felhasználók számára az oktatást is megvalósítani. Minden új bevezetett eszközhöz szükséges szakszerű oktatást is tartani, mert a teljes dokumentáció átadása nem egyenértékű a gyakorlati oktatással. Mivel ebben az időszakban egyébként is szélsőséges eseteket vettünk számba, itt volt a legnagyobb esélye az algoritmusoknak hibázni. Szerencsére nagy, azonnal nem javítható hiba nem derült ki. Ez az időszak 1-2 hetet vett igénybe, mely után ez a rendszer lett a fő rendszerük.

Már rövidtávon is jelentkezik az igazi előnye az új megvalósításnak. Sokkal gyorsabban lehet benne dolgozni, mint az előző rendszerben. Egy új irányvonalat is bevezettek, amire a készítésnél nem is esett szó: az értékesítők is külön belépést kaptak a rendszerhez. Sokkal gyorsabban tudnak árajánlatot adni és gyártási időt becsülni, mintha a teljes gyártási láncon végig futna a kérés és a hozzá tartozó válasz. Ez fejlesztői oldalról semmi újat nem jelent, csupán egy új felhasználót kellett létrehozni.

Az első fázist végleges verzióként használva egyre inkább érződik az igény a dolgozók részéről a második fázis elkezdésére, ahol az előző fejezetekben említett extra funkciók lesznek elérhetőek, mint például az automatikus anyag rendelés.

1. Summary

Small and medium sized companies often struggle with the problem of finding a software that matches with their existing problem and budget in the same time. The above-mentioned company also has this dilemma.

The original task was to find a solution that handles their production management more efficiently than their existing solution with excel tables. They were satisfied with a web application that stores the BOM hierarchy in a relational database. The key point here was to get a cross-platform solution that is centralized and can be used by multi people in the same time.

The leadership was aware the fact that the market already has solution for the same scenario, but those products are costly compared to a small and efficient application. As a result, they agreed to start the custom development of the new application.

The purpose of this development is clearly defined and it makes the scope of work quite rigid. The management had to declare their processes and existing data structure that can be transformed into the new data structure. This involved the 5 level of hierarchy (part, operation, assembly, product, project). Although the 5 level is predefined, the source code left room for further expansion.

The architectural testing had been performed in a way that the application can be easily deployed on the customer’s existing server park or even in the cloud as well. According to the latest trends, most of the companies will transfer at least a part of their infrastructure into the cloud, because of efficiency and safety reasons.

Applying the well-known testing methods eliminated lots of testing during the development and the first phase of the deployment as well. Using a docker container for testing purpose gave an extra benefit: the Dockerfile in the source directory is an actual recipe to make a running environment from scratch. This gave extra strength to the documentation.

Transferring the existing excel tables into the new database system is semi-automatized. This requires some extra human resources from the customer side. As soon as this process finished and the new system started working the efficiency of the employees improved right away.

Recording all the operations that is required for a specific product gives extra information to the management, because it was not defined clearly yet. They got a view about how much work is involved in a product and actually how much does it cost. It makes a better estimation is the sales as well. Before this solution the production time was estimated by the previous experiences. It cannot be accurate at all.

In the future a new module is planned to be implemented that links all the operations to employees. It makes a better view about how busy are the workers and which department needs extra work resource.

Since the application had not been in long term use, the return of investment cannot be determined yet. However, the management already agreed that the next phase of the development is required and the new modules will give extra benefit for the company. According to them, the already existing system already made it clear, that it did not just make the processes simpler and quicker, it gave extra documentation for the processes without any effort from their side.

1. Mellékletek
   1. Osztály specifikációk
      1. Elements osztály

Egy osztály, amelyik egy elemet reprezentál a BOM listában. A hierarchia bármely szintjére alkalmazható, minden elem típust képes megvalósítani. Egy elem létrehozásához két lehetőség van:

* Üres elem létrehozása a konstruktor kitöltésével
* Elem betöltése az adatbázisból a load\_parameters\_from\_database függvénnyel

Belső változói:

* self.id -Az adatbázisban lévő egyedi azonosító
* self.name – Az adatbázisban lévő rövid elnevezés
* self.type – Az adatbázisban lévő elem típus
* self.code – Az adatbázisban lévő egyedi kód
* self.icon – Az adatbázisban lévő ikon referencia
* self.conn – Az adatbáziskapcsolat a táblaműveletekhez

6. Táblázat Eljárások és függvények

|  |  |
| --- | --- |
| **def \_\_init\_\_(self, id=0, name="", type="", code="", icon="")** | |
| Leírás | Az osztály konstruktora. Üres elem létrehozásakor használatos.  Itt készül egy új adatbáziskapcsolat ami kizárólag erre az elemre használatos. |
| Bementi paraméterek | id:<int> - Az adatbázisban lévő egyedi azonosítója (default:0)  name:<str> - Az elem neve (default:"")  type:<str> - Az elem típusa szövegesen kifejezve (default:"")  code:<str> - Az elem egyedi kódja (default:"")  icon:<int> - Referencia az elem kódjának rekorjára a fájlok táblában (default:"") |
| Kimeneti paraméterek | Egy objektum az elem reprezentálására |
| **def load\_parameters\_from\_database(self, part\_id)** | |
| Leírás | Az objektum belső változóinak feltöltése az adatbázisból letöltött adatokkal. A függvény lekérdezi az adatbázisból a kért elemet és feltölti az objektum következő változóit a letöltött értékekkel: id,name,type,code,icon |
| Bemeneti paraméterek | part\_id:<int> - Az adatbázisban lévő elem egyedi azonosítója |
| Kimeneti paraméterek | - |
| **def instock(self)** | |
| Leírás | Visszaadja egy elem raktárkészletét az adatbázisból. Visszatérési értékként egy összesített számot ad, amely az adattáblában lévő beszerzések értékének összege. |
| Bemeneti paraméterek | - |
| Kimeneti paraméterek | raktárkészlet:<int> |
| **def get\_inventory(self)** | |
| Leírás | @property  Visszaadja egy elem beszerzési történetét az adatbázis alapján. Ennek a függvénynek a pieces mezőinek összesítése adja az instock() függvény eredményét. Kimentként egy lista várandó, amit tartalmazza az összes rekordot a beszerzések táblából, a szükséges mezők: pieces, description, date |
| Bemeneti paraméterek | - |
| Kimeneti paraméterek | beszerzési\_lista:<list> - {"pieces", "description", "date"} |
| **def purchaseOpportunities(self)** | |
| Leírás | @property  Visszaadja egy elem beszerzési lehetőségeit az adatbázisból. |
| Bemeneti paraméterek | - |
| Kimeneti paraméterek | beszerzési\_lista:<list> - {"company", "price", "priceunit", "link", "unit", "code"} |
| **def consist\_short\_list(self):** | |
| Leírás | @property  Visszaadja, hogy az adott elem milyen más elemeket tartalmaz. A hierachia szint lefelé történő kibontása egy listába. A kibontás gyökérelemét minden esetben az objektum ID-ja határozza meg, ami nem egy bemenő paraméter. Visszatérési értékként egy listát ad amely az elemek objektumait tartalmazza. |
| Bemeneti paraméterek | - |
| Kimeneti paraméterek | tartalmaz\_lista:<list> |
| **def get\_tree\_diagram\_piece(self)** | |
| Leírás | Az elem és az ő által tartalmazott más elemek fa diagramjának kirajzolása HTML kódban. |
| Bemeneti paraméterek | - |
| Kimeneti paraméterek | kód:<str> -> A HTML kódba közvetlenül beilleszthető kódrészlet amely a fa struktúra kirajzolását valósítja meg. |

|  |  |
| --- | --- |
| **def mediaList(self)** | |
| Leírás | @property  Visszaadja, hogy az adott elemhez milyen fájlok vannak feltöltve az adatbázisban. Visszatérési értékként egy listát ad amely a feltöltött fájlok objektumait tartalmazza. |
| Bemeneti paraméterek | - |
| Kimeneti paraméterek | fájl\_lista:<list> |
| **def bom(self)** | |
| Leírás | @property  Visszaadja az elem rekurzívan kidolgozott BOM listáját. A gyökér elem minden esetben az adott elem. A rekurzív feldolgozást az adatbázis valósítja meg. |
| Bemeneti paraméterek | - |
| Kimeneti paraméterek | fájl\_lista:<list> - {"ContainerID", "ElementID", "Pieces", "ChildName", "Type", "Icon"} |
| **def sum\_price (self)** | |
| Leírás | @property  Visszaadja az elem BOM listájának az adatbázisban rögzített árainak összesítését. Ebben a listában szerepel az összes elem, amit a szülő tartalmaz és azoknak az árai. Az árak, mint egységár és mint darabszámmal felszorozott összesített ár is megjelennek. |
| Bemeneti paraméterek | - |
| Kimeneti paraméterek | fájl\_lista:<list> -> {"ChildName", "Description", "min\_price", "max\_price", "UnitType", "pieces", "unit"} |
| **def change\_icon(self, icon\_path)** | |
| Leírás | Megváltoztatja az adatbázisban az elemhez rendelt ikon referenciát. |
| Bemeneti paraméterek | icon\_path:<int> -> A referencia a fájlok listához |
| Kimeneti paraméterek | success:<boolean> |
| **def createInDatabase(self)** | |
| Leírás | Egy üres elem létrehozása, majd annak adatokkal történő feltöltése után ez a függvény hozza létre az elemet az adatbázisban. A konzisztencia megtartása végett az objektum ezután felülírja magát az adatbázisban tárolt információkkal. |
| Bemeneti paraméterek | - |
| Kimeneti paraméterek | success:<boolean> |
| **def delete(self)** | |
| Leírás | Ez az eljárás kitörli az elemet az adatbázisból. Az objektum konstruktora nem hívódik meg automatikusan ebből. |
| Bemeneti paraméterek | - |
| Kimeneti paraméterek | success:<boolean> |

|  |  |
| --- | --- |
| **def add\_purchase\_opportunity(self, vendorCode, priceUnit, price, unit, code, link)** | |
| Leírás | Hozzáad az elemhez egy beszerzési lehetőséget az adatbázisban. |
| Bemeneti paraméterek | vendorCode:<str>  priceUnit:<int> - az adatbázisban tárolt értékek közüli ID  price:<int>  unit:<str>  code:<str>  link:<str> |
| Kimeneti paraméterek | success:<boolean> |
| **def add\_consist\_element(self, childID, pieces)** | |
| Leírás | Egy elem hozzárendelése az adott objektumhoz. Ezzel a hierarchiában alá-fölé rendeltségi viszonyban lesznek. |
| Bemeneti paraméterek | childID:<int> - a hozzáadandó elem ID-je  pieces:<int> - hozzáadandó darabszám |
| Kimeneti paraméterek | success:<boolean> |
| **def modify\_consist\_element(self, childID, pieces)** | |
| Leírás | Egy korábban már hozzáadott elem darabszámának módosítása a hierarchia fában. |
| Bemeneti paraméterek | childID:<int> - a módosítandó elem ID-je  pieces:<int> - darabszám |
| Kimeneti paraméterek | success:<boolean> |
| **def delete\_consist\_element(self, childID)** | |
| Leírás | Egy korábban már hozzáadott elem törlése a listából, így kikerül az alá-fölé rendeltségi hierarchiából. |
| Bemeneti paraméterek | childID:<int> - a törlendő elem ID-je |
| Kimeneti paraméterek | success:<boolean> |
| **def delete\_purchase\_opportunity(self, orderCode)** | |
| Leírás | Egy korábban már hozzáadott beszerzési lehetőség törlése az adatbázisból. |
| Bemeneti paraméterek | orderCode:<int> -> a törlendő elem ID-je |
| Kimeneti paraméterek | success:<boolean> |
| **def add\_to\_inventory(self, qty, description)** | |
| Leírás | Egy beszerzés regisztrálása az adatbázisban. Ezzel a függvénnyel növelhető (vagy akár csökkenthető) a raktárkészlete az adott terméknek. |
| Bemeneti paraméterek | qty:<int> - a darabszám  description:<str> - leírás a beszerzéshez (pl számla sorszám) |
| Kimeneti paraméterek | success:<boolean> |

|  |  |
| --- | --- |
| **def modify\_name(self, new\_name)** | |
| Leírás | Módosítja az elem nevét az adatbázisban. |
| Bemeneti paraméterek | new\_name:<str> - az új név |
| Kimeneti paraméterek | success:<boolean> |
| **def modify\_code(self, new\_code)** | |
| Leírás | Módosítja az elem kódját az adatbázisban. |
| Bemeneti paraméterek | new\_code:<str> - az új kód |
| Kimeneti paraméterek | success:<boolean> |

* + 1. Vendor osztály

Egy osztály, amelyik egy létező beszállítót reprezentál. Egy elem létrehozásához két lehetőség van:

* Üres elem létrehozása a konstruktor kitöltésével
* Elem betöltése az adatbázisból a load\_parameters\_from\_database függvénnyel

Belső változói:

* self.id - Az adatbázisban lévő egyedi azonosító
* self.company – A beszállító cégneve
* self.address – A cég telephelye
* self.conn - Az adatbáziskapcsolat a táblaműveletekhez

7. Táblázat Eljárások és függvények

|  |  |
| --- | --- |
| **def \_\_init\_\_(self, id, company, address)** | |
| Leírás | Az osztály konstriktora. Üres elem létrehozásakor használatos. Itt készül egy új adatbáziskapcsolat ami kizárólag erre az elemre használatos. |
| Bemeneti paraméterek | id:<int> -> A fájl elérési útja (kötelező mező)  company:<str> > A cég elnevezése (kötelező mező)  address:<str> - A cég telephelyének címe (kötelező mező) |
| Kimeneti paraméterek | Egy objektum az elem reprezentálására |
| **def load\_parameters\_from\_database(self, id):** | |
| Leírás | Az objektum belső változóinak feltöltése az adatbázisból letöltött adatokkal. A függvény lekérdezi az adatbázisból a kért elemet és feltölti az objektum következő változóit a letöltött értékekkel: id, company, address |
| Bemeneti paraméterek | id:<int> -> Az adatbázisban lévő elem egyedi azonosítója |
| Kimeneti paraméterek | - |

|  |  |
| --- | --- |
| **def createInDatabase(self)** | |
| Leírás | Egy üres elem létrehozása, majd annak adatokkal történő feltöltése után ez a függvény hozza létre az elemet az adatbázisban. |
| Bemeneti paraméterek | - |
| Kimeneti paraméterek | success:<boolean> |
| **def delete(self)** | |
| Leírás | A bejegyzés törlése az adatbázisból. A destruktor nem hívódik meg automatikusan az eljárás végén. |
| Bemeneti paraméterek | - |
| Kimeneti paraméterek | success:<boolean> |

* + 1. Media osztály

Egy osztály, amelyik egy feltöltött fájlt reprezentál.

Belső változói:

* self.id - Az adatbázisban lévő egyedi azonosító
* self.path – A fájl elérési útja a fájlrendszerben
* self.description – A fájlhoz tartozó leírás
* self.is\_image – A fájl kép típusának tárolása
* self.conn - Az adatbáziskapcsolat a táblaműveletekhez

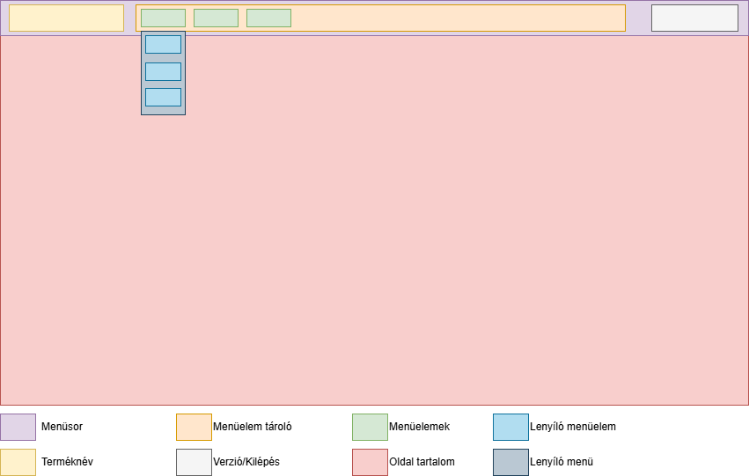
|  |  |
| --- | --- |
| **def \_\_init\_\_(self, path, Description, refs=0, id=-1)** | |
| Leírás | Az osztály konstriktora. Üres elem létrehozásakor használatos. Itt készül egy új adatbáziskapcsolat ami kizárólag erre az elemre használatos. |
| Bemeneti paraméterek | path:<str> -> A fájl elérési útja (kötelező mező)  Description:<str> -> A fájl leírása (kötelező mező)  id:<int> -> Az adatbáizban lévő egyedi ID (default: -1, nincs az adatbázisban) |
| Kimeneti paraméterek | Egy objektum az elem reprezentálására |
| **def load\_parameters\_from\_database(self, id):** | |
| Leírás | Az objektum belső változóinak feltöltése az adatbázisból letöltött adatokkal. A függvény lekérdezi az adatbázisból a kért elemet és feltölti az objektum következő változóit a letöltött értékekkel: id, path, description |
| Bemeneti paraméterek | id:<int> -> Az adatbázisban lévő elem egyedi azonosítója |
| Kimeneti paraméterek | - |

|  |  |
| --- | --- |
| **def createInDatabase(self)** | |
| Leírás | Egy üres elem létrehozása, majd annak adatokkal történő feltöltése után ez a függvény hozza létre az elemet az adatbázisban. |
| Bemeneti paraméterek | - |
| Kimeneti paraméterek | success:<boolean> |
| **def delete(self)** | |
| Leírás | A fájl törlése az adatbázisból és a fájlrendszerről is. A destruktor nem hívódik meg automatikusan az eljárás végén. |
| Bemeneti paraméterek | - |
| Kimeneti paraméterek | success:<boolean> |
| **def attachFileToElement(self, id)** | |
| Leírás | A fájl hozzárendelése egy elemhez az adatbázisban. Itt alá-fölé rendeltségi viszony alakul ki az elem és a fájl között. |
| Bemeneti paraméterek | - |
| Kimeneti paraméterek | success:<boolean> |

* 1. Képernyőképek tervei

Általános szabályként a használt bootstrap css keretrendszer osztályainak használata az előnyös, így extra css kód megspórolása érhető el. Minden olyan osztály, amely nincs kifejtve részletesen, abban a keretrendszerben található meg.

* + 1. Általános menü

Az általános képernyőkép része egy felül megjelenő vízszintes menüsor. Ez minden oldalon megjelenik, de a tartalma az autentikáció függvényében változik.

**Menüsor:** Ez a konténer tárolja a teljes menüsor tartalmát.

HTML tag: div

24. ábra Felső menüsor

Hozzá tartozó CSS osztályok: navbar navbar-expand-lg navbar-dark bg-dark

Terméknév: Egy szöveg a termék nevével.

CSS osztály: navbar-brand

**Menüelem tároló:** Ebben a tárolóban helyezkednek el a menüelemek egymás utáni sorban az autentikáció függvényében. Mobilos nézetben ez a menüsor függőleges és összezárható formátumúra vált át magától.

HTML tag: div; CSS osztályok: collapse navbar-collapse

**Menüelemek:** Linkek az egyes menük eléréséhez. Egyes esetekben lenyíló menü lehet, mely kattintásra nyílik le. A menüelem tárolón belül egy számozatlan listaként jeleníti meg a menüelemeket, amelyek egy linket tartalmaznak a különböző oldalakhoz.

HTML tag: a; CSS osztályok a számozatlan listához:navbar-nav mr-auto

CSS osztályok a lista elemekhez: nav-item

CSS osztályok az oldalakra mutató linkekhez a listaelemen belül: nav-link

**Lenyíló menü:** Kattintásra egy almenü lesz látható. A lenyílás lehetőségét apró nyíl jelzi az elem jobb oldalán.

HTML tag: div; CSS osztályok: dropdown-menu bg-dark show

**Lenyíló menüelem:** Az almenük megjelenítése linkek formájában.

HTML tag: a; CSS osztályok: dropdown-item

Oldal tartalom: Ezen a helyen jelennek majd meg a különböző oldalak a menüsávtól függetlenül.

**Verzió/Kilépés:**

Bejelentkezett felhasználó esetében itt egy kijelentkezés gomb jelenik meg:

HTML tag: button; CSS osztályok: btn btn-danger my-2 my-sm-0 ml-3

Nem bejelentkezett felhasználó esetén a szoftver verziószáma jelenik meg:

HMTL tag: div; CSS osztályok: my-2 my-sm-0 ml-3

* + 1. Bejelentkező képernyő

A fő képernyő bejelentkezésre alkalmas. Egy középre igazított ablakban megjelenő form amelynek két eleme van: felhasználónév, jelszó. Ezen túl egy submit gomb post üzenet formájában továbbítja az adatokat a szerver felé a “/” oldalra. Az oldal hátterében egy SVG fájl megjelenítése biztosítja a modern megjelenést.

**Bejelentkező ablak:**

HTML tag: div

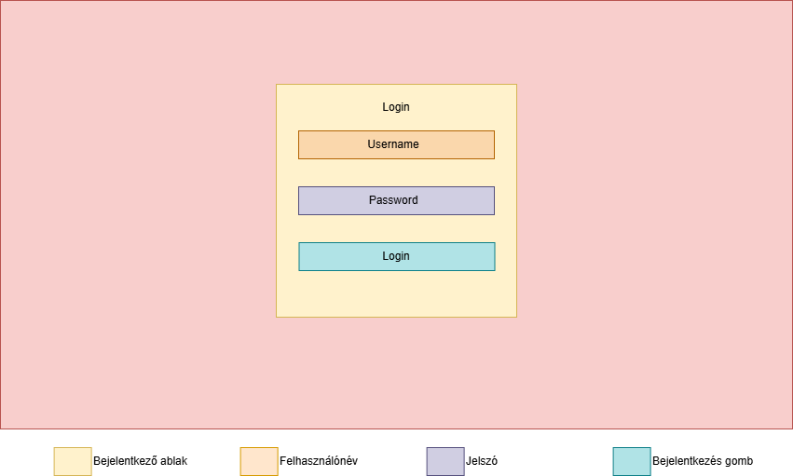
CSS:

background-color: #ffffff;

border-radius: 8px;

box-shadow: 0 4px 10px rgba(0, 0, 0, 0.1);

padding: 30px;

 text-align: center;

width: 300px;

**Felhasználónév:**

HTML tag: input>text

Form name: name

Megjelenítendő szöveg: Username

**Jelszó:**

HTML tag: input>password

Form name: password

Megjelenítendő szöveg: Password

25. ábra Bejelentkező képernyő

**Bejelentkezés gomb:**

HTML tag: button>submit

CSS osztály: submit-btn

Megjelenítendő szöveg: Login

* + 1. Cégbemutató oldal

Az oldal a cég bemutatására szolgál. A szöveg két részből áll, a felső részben az cégbemutató szöveg található figyelemfelhívó formátumban. Az alsó részben a cég által biztosított bemutatkozó szöveg.

**Cégbemutató szöveg:**

HTML tag: h2

CSS: pl-5 pr-5

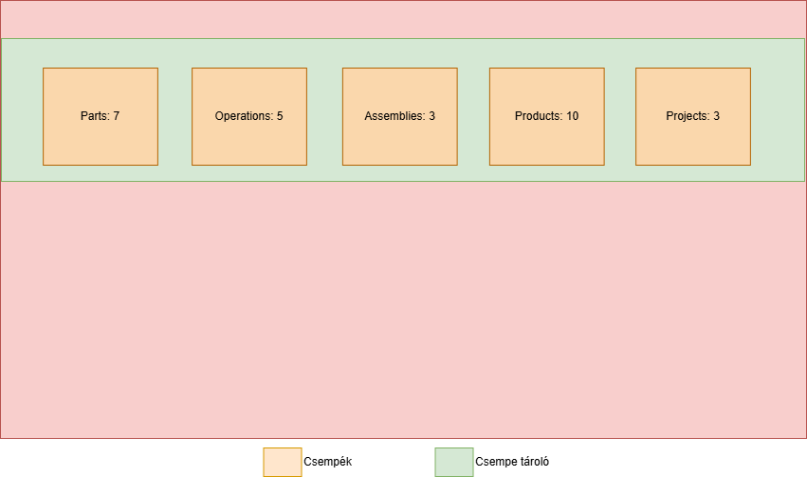
**Szövegtörzs:**

HTML tag: p

CSS: pl-5 pr-5

26. ábra Cégbemutató oldal

* + 1. Vezérlőpult oldal

Az oldalon 5db csempe ikon található az 5 hierarchia típus megjelenítésére.

**Csempe tároló:**

A csempe elemek összefogására és középen tartására szolgál.

HTML tag: div

CSS: pt-5 pr-5 pl-5

display: flex;

flex-wrap: wrap;

gap: 20px;

justify-content: center;

**Csempék:**

27. ábra Vezérlőpult oldal

Minden elem típus esetén külön megjelenítendő csempe. A megjelenítendő szöveg a típus neve és az adatbázisban található elemszáma.

HTML: a

CSS:

background-color: #0078d4;

color: white;

width: 200px;

height: 150px;

display: flex;

align-items: center;

justify-content: center;

font-size: 20px;

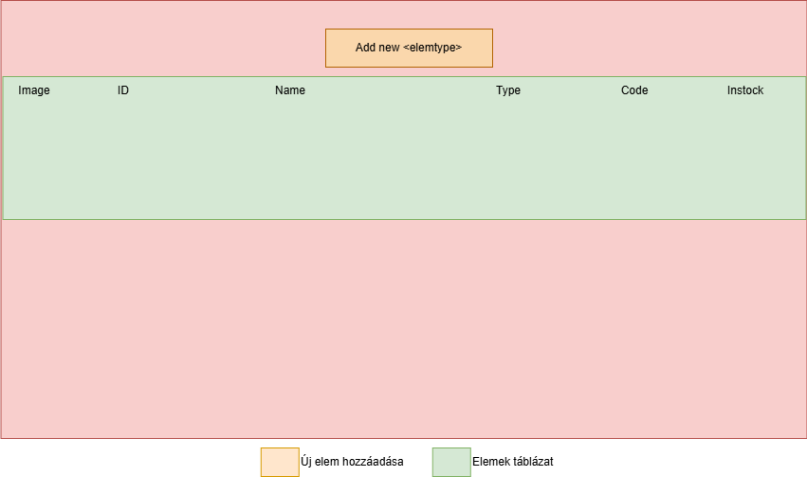
font-weight: bold;

border-radius: 10px;

transition: transform 0.3s ease;

margin: 50px;

* + 1. Elemek listája oldal

Egy oldal képes megjeleníteni mind az 5 elemtípust. Ez az oldal meghívásától függ, hogy melyik elemek kerülnek megjelenítésre. A megjelenítési oldalon csak a felső gombban megjelenített szövegben különbözik. A háttérfolyamatban pedig az SQL-ben elküldött WHERE záradékban.

28. ábra Elemek listája oldal

**Új elem hozzáadása:**

Egy gomb, amelynek szövege a megjelenítendő elemektől függ. Középre helyezett. Kattintásra egy felugró ablak jelenik meg, amelyben hozzá lehet adni új elemet. A felugró ablak egy bootsrap modal elem.

HTML tag: button

CSS: btn btn-primary mt-3 mb-3

Modal ID: #AddNewElementModal

**Elemek táblázat:**

Minden megjelenítendő elem külön sorban jelenik meg. A sorra való kattintáskor az “elem részletei” oldal jelenik meg a kiválasztott elemre vonatkozóan.

HTML tag: table

CSS: table-striped table-dark element\_table

* + 1. Új elem hozzáadása felugró ablak

Egy új elem hozzáadására szolgál. Egy formot tartalmaz 2 mezővel és egy küldő gombbal. A modal elem középre van igazítva minden irányból és a hátteret elsötétíti. A form a “/parts/add” oldalt hívja meg a mentendő elemekkel POST metódussal.

**Modal:**

HTML tag: div

CSS:

position: relative;

display: -ms-flexbox;

display: flex;

-ms-flex-direction: column;

flex-direction: column;

width: 100%;

pointer-events: auto;

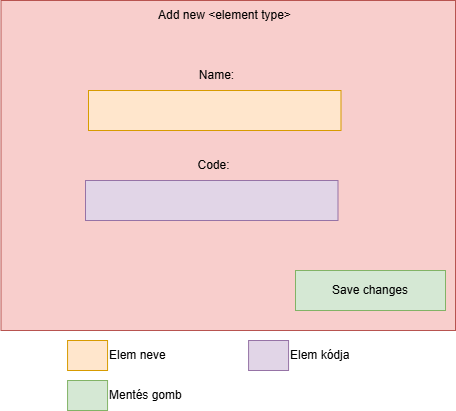
background-color: #fff;

background-clip: padding-box;

border: 1px solid rgba(0, 0, 0, .2);

border-radius: .3rem;

outline: 0;

****

29. ábra Új elem hozzáadása felugró ablak

**Elem neve:**

kötelező mező

HTML tag: input>text

CSS: form-control

Name: name

**Elem kódja:**

kötelező mező

HTML tag: input>text

CSS: form-control

Name: code

**Mentés gomb:**

HTML tag: input>submit

CSS: btn btn-primary

* + 1. Elem részletei oldal

Az oldal minden típusú elem megjelenítésére alkalmas. Az oldal függőlegesen két részre van bontva. A felső részen az elem alaptulajdonságai láthatóak és egy akciógomb menü.

Az oldal alsó részén az elemmel végezhető műveletek láthatóak több füles elrendezésben. A fülek láthatósága az elem típusától függ.

Az felső rész első oszlopában egy form található, amellyel az elem 2 tulajdonsága módosítható. Ez egy POST kérést küldd a “#” oldalra. Ehhez rejtett form mező használata szükséges (name=formType, value=modifyFundamentals). A megjelenítés táblázatos formátumban történik, váltott sorosan sötétebb színnel.

A felső rész középső elemében a kép ikonja jelenik meg.

A felső rész jobb oldalán egy akció menü jelenik meg, amelyben alapműveletek érhetőek el gombnyomásra.

**Alap tulajdonságok táblázat:**

HTML tag: div

CSS osztály: mt-5CSS:

display: flex;

flex-wrap: wrap;

margin-right: -15px;

margin-left: -15px;

**Minden oszlopa:**

HTML tag: div:

CSS: col-3

**Név mező:**

HTML tag: input>text

CSS: form-control me-sm-2 text-center

Name: element\_name

**Kód mező:**

HTML tag: input>text

CSS: form-control me-sm-2 text-center

Name: element\_code

**Módosít gomb:**

HTML tag: input>submit

CSS: btn btn-secondary

Name: Modify

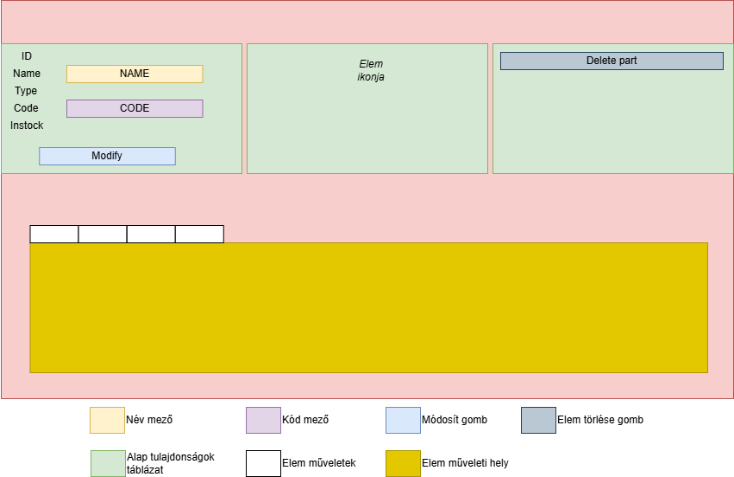
Value: Modfiy

**Elem törlése gomb:**

HTML tag: button

CSS: btn btn-danger mt-3

Megnyomásra egy felugró ablak megkérdezi, hogy biztosan törölni kívánja-e. Pozitív válasz esetén az “elements/delete/<id>” oldalra navigál a törlés elvégézéséhez POST metódussal.

Az alsó rész az elemmel kapcsolatos műveletekhez köthető. Egy sorban látható az elemek neve és a nevekre történő kattintás az alatta lévő rész tartalmát változtatva egy “füles” elrendezést valósít meg.

30. ábra Elem részletei oldal

**Elem műveletek:**

Egy számozatlan listában jelennek meg.

HTML tag: ul

CSS: nav nav-tabs mt-5

Minden egyes elem:

HTML tag: li

CSS:

padding: 10px;

background: #343a40;

border-radius: 10px 10px 0px 0px;

margin-right: 3px;

Az alatta lévő helyen egy gyüjtő konténerben jelennek meg a panelek, melyből egyszerre mindig csak egy látható.

**A gyűjtő konténer adatai:**

HTML tag: div

**Minden egyes panel adata:**

HTML tag: div

CSS: tab-pane fade pt-4 pb-4 rounded <show>

A show osztályt abban az esetben kell aktiválni, ha arra kattintott a felhasználó

Minden fül tartalma alább lett részletezve.

* + 1. Elemek részletei oldal / Media oldal

Ez a fülön lehet az elemekhez különböző fájlokat csatolni. Itt van lehetőség a múltban csatolt fájlok letöltésére is.

Az oldal felső részén egy középre helyezett form jelenik meg amiben 2 mező található. A leírás mező és a fájl csatolásához szükséges mező. Ez alatt a feltöltés gomb jelenik meg.

**Feltöltés form:**

A form a “#”-ra küldi a fájl POST metódussal.

HTML tag: div

CSS: modal-body bg-secondary ml-auto mr-auto mt-3 mb-1

**Leírás mező:**

kötelező mező

HTML tag: input>text

Name: description

CSS: form-control ml-auto mr-auto mb-3

width:400px;**Fájl mező:**

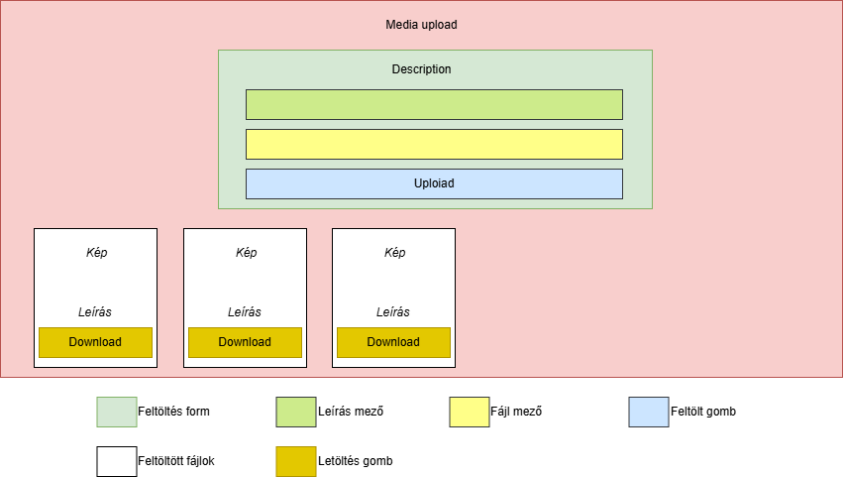
kötelező mező

HTML tag: input>file

Name: file

CSS: ml-auto mr-auto mb-3

Az oldal alsó részén a feltöltött fájlok jelennek meg flex-box formában balra igazítva.

**Feltöltött fájlok:**

31. ábra Media aloldal

Egy ilyen fájl kinézetét valósítja meg. Minden fájl külön ilyen konténert kap.

HTML tag: div

CSS: card bg-light text-dark m-2 rounded

**Ezen belül a felső kép adatai:**

HTML tag: img

CSS: card-img-top

height:150px;

**Ez alatt egy div helyezkedik el a következő adatokkal:**

HTML tag: div

CSS: card-body

Tartalomként a leírás szövege egy h4-es tagben “card-title” class-ba foglalva.

**Letöltés gomb:**

HTML tag: a

CSS: btn btn-primary

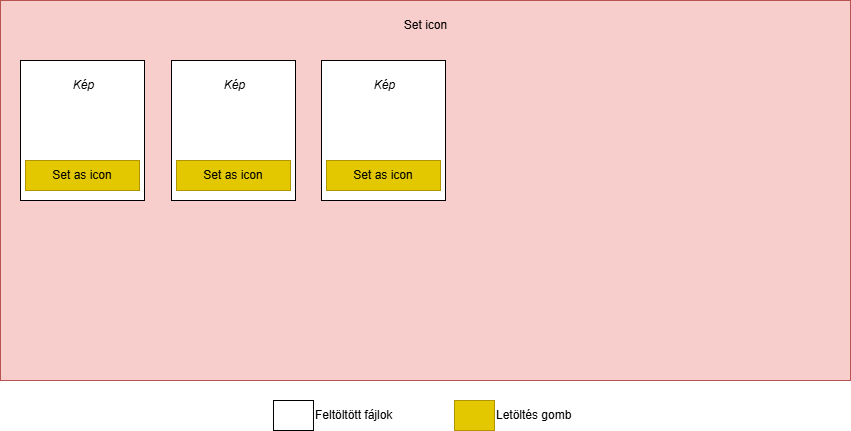
Megjelenítendő szöveg: Download

A link a feltöltött fájl referanciájára mutat.

* + 1. Elemek részletei oldal / Ikon beállítása

Itt a már korábban feltöltött fájlokat lehet beállítani egy kattintással, mint ikon az adott elemhez. Ehhez egy listában felsorolódik az összes feltöltött kép egy balra rendezett flex-box konténerben.

**Egy benne elhelyezkedő elem leírása:**

CSS: card bg-light text-dark m-2 rounded

HTML tag: div

**Ezen belül a felső kép adatai:**

HTML tag: img

CSS: card-img-top

height:150px;

**Beállítás gomb:**

HTML tag: a

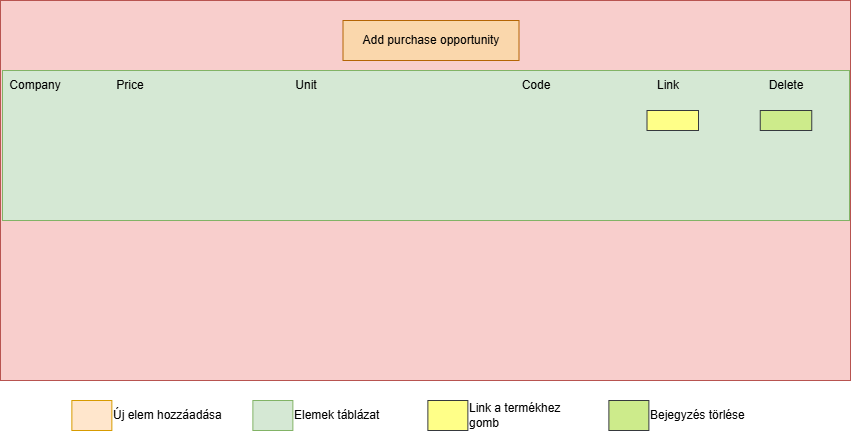
CSS: btn btn-primary

Megjelenítendő szöveg: Set as icon

32. ábra Ikon beállítása aloldal

* + 1. Elemek részletei oldal / Beszerzési lehetőségek aloldal

Ezen az oldalon beszállítói lehetőségek regisztrálására van lehetőség. Az új lehetőség gomb megnyomásakor egy felugró ablakban van lehetőség egy új elem létrehozására.

A már meglévő elemek az alsó részen egy táblázatban jelennek meg.

33. ábra Beszerzési lehetőségek aloldal

**Új elem hozzáadása gomb:**

HTML tag: div

CSS: btn btn-primary m-4

Kattintásra a “.add-vendor-form” nevű modal aktiválódik.

**Elemek táblázat:**

HTML tag: table

CSS: table-striped table-dark

width: 90%;

margin-left: auto;

margin-right: auto;

Az utolsó két oszlopban egy-egy gomb jelennek meg amelyek a termék beszállítói linkjét tartalmazzák, illetve az bejegyzés törlését teszik lehetővé.

**Link gomb:**

HTML tag: a

CSS: btn btn-warning mt-2

Megjelenítendő szöveg: Link

**Törlés gomb:**

Kattintásra egy felugró ablak megkérdezi, hogy biztosan törli-e.

HTML tag: a

CSS: btn btn-danger

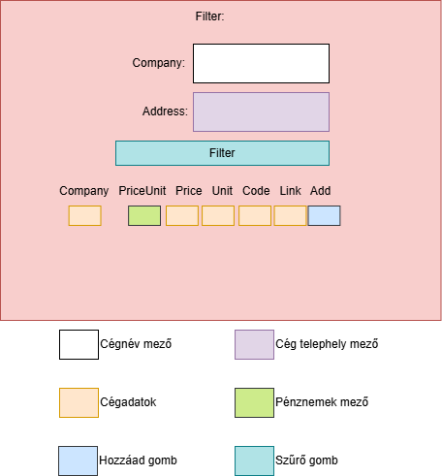
Megjelenítendő szöveg: Delete

* + 1. Elemek részletei oldal / Beszerzési lehetőségek aloldal / Felugró ablak

A felugró modal panel az erre vonatkozó gomb megnyomása után. A felső részében egy szűrő található, amely az alatta megjelenített lista elemeit szűri.

A felső form két mezőt tartalmaz középre igazítva:

**Cégnév mező:**

****kötelező mező

34. ábra Beszerzési lehetőségek felugró ablaka

HTML tag: input>text

Name: company

**Address mező:**

HTML tag: input>text

Name: address

**Filter gomb:**

A form elküldése a POST metódussal a “#” oldalra.

HTML tag: input>submit

CSS: btn btn-primary mt-2 w-100

Megjelenítendő szöveg: Filter

**Az alsó táblázat adatai:**

Az első oszlopa nem szerkeszthető, a cégnév jelenik meg benne.

A második oszlop egy legördülő menü, amiben a mentett pénznemek jelennek meg.

Ezt követően az összes oszlop text mező az oszlop fejlécének megfelelő névvel, kisbetűsen írva form hivatkozásba.

**A hozzáadás gomb adatai:**

HTML tag: input>submit

CSS:btn btn-success

Megjelenítendő szöveg: Add

A formhoz két hidden mező is szükséges:

name: formtype value=addVendor

name: vendorID value=<adatbázisból vett érték>

name: elementID value=<megjelenített elem id>

A form összes input mezője kötelező érték.

* + 1. Elemek részletei oldal / Tartalmaz lista

Ez az oldal a hierarchia felépítésére szolgál. A felső sorban egy gomb jelenik meg amire kattintva egy felugró ablakban új elem hozzáadása lehetséges.

Alatta a már felvett elemek jelennek meg és lehetőség van a darabszám módosítására is. A darabszám módosítására egy form ad lehetőséget. A Törlés gombra kattintva egy felugró ablak megjelenítése után törli az elemet a listából. A link gomb a termék részletes lapját jeleníti meg.

**Új elem hozzáadása:**

HTML tag:button

CSS: btn btn-primary m-4

Megjelenítendő modal: .add-consist-form

**Elemek táblázat:**

HTML tag: table

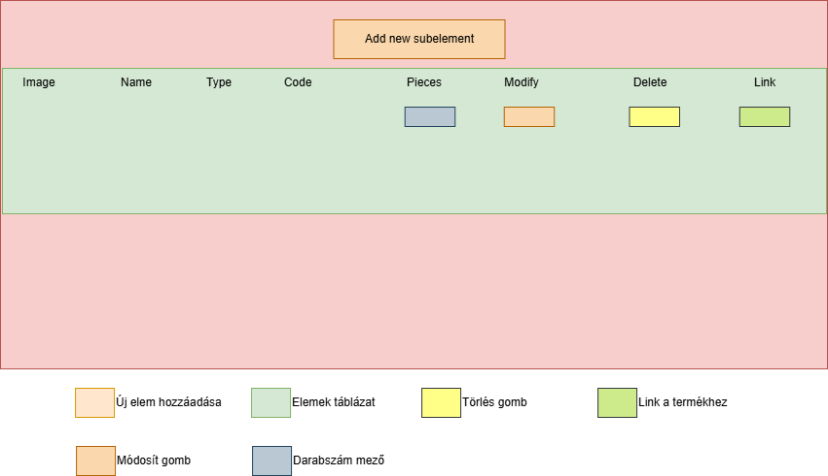
CSS: table-striped table-dark

width: 90%;

margin-left: auto;

margin-right: auto;

**Törlés gomb:**

HTML tag: button

CSS: btn btn-danger

Megjelenítendő szöveg: Delete

**Link gomb:**

HTML tag: a

CSS: btn btn-info

Megjelenítendő szöveg: Link

**Módosít gomb:**

Módosítja egy rekord darabszámát.

HTML tag: input>submit

CSS: btn btn-warning

Megjelenítendő szöveg: Modify

**Pieces mező:**

Egy form eleme, melynek a submit gombja a módosít gomb.

HTML tag: input>number

CSS: width:50px;

35. ábra Tartalmaz lista aloldal

**A módosít form rejtett mezői:**

name=formType; value=consistModify

name=childID; value=<id>

* + 1. Elemek részletei oldal / Tartalmaz lista / Új elem hozzáadása

A felső sorban egy szűrő form található meg. Ennek legelső eleme egy 5 elemű választó gomb az 5 különböző típusú elem típusra. Másik két mezője az elemnév és elemkód. A Filter gombra kattintva a szűrő aktiválódik.

Alatta egy listában a megjelenítendő elemek foglalnak helyet egy táblázatban. Az utolsó két eleme a táblázatnak egy mező, ami a mennyiséget tartalmazza és a hozzáadás gombot.

**Elemnév mező:**

kötelező mező

HTML tag: input>text

**Elemkód mező:**

kötelező mező

HTML tag: input>text

**Típus választó gombok:**

kötelező mező

HTML tag: input>radio

CSS: btn-check

Az értékek az 5 mezőhöz: {part, operation, assembly, product, project}

**Szűrő gomb:**

HTML tag: form>submit

CSS: btn btn-primary mt-2 w-100

A form POST metódost használ a “#” végcélra.

Az alsó táblázat egy középre igazított elem. Minden sor egy külön form a két benne foglalt mezővel. Az első oszlopa egy miniatűr képet jelenít meg az elemről

**Táblázat adatai:**

HTML tag: table

CSS: table-striped table-dark mb-3

width: 90%;

margin-left: auto;

margin-right: auto;

**Mennyiség mező:**

kötelező mező

HTML tag: input>number

**Hozzáad gomb:**

HTML tag: input>submit

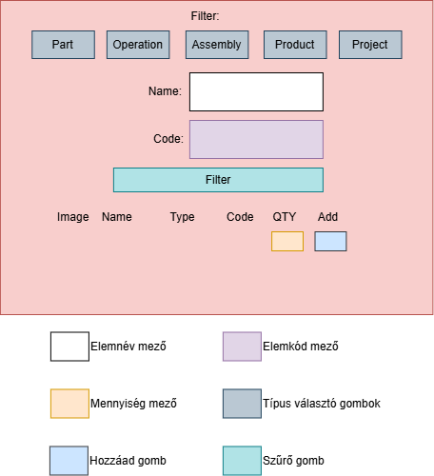
CSS: btn btn-success

**A hozzáadás form rejtett mezői:**

name=formType value=addConsist

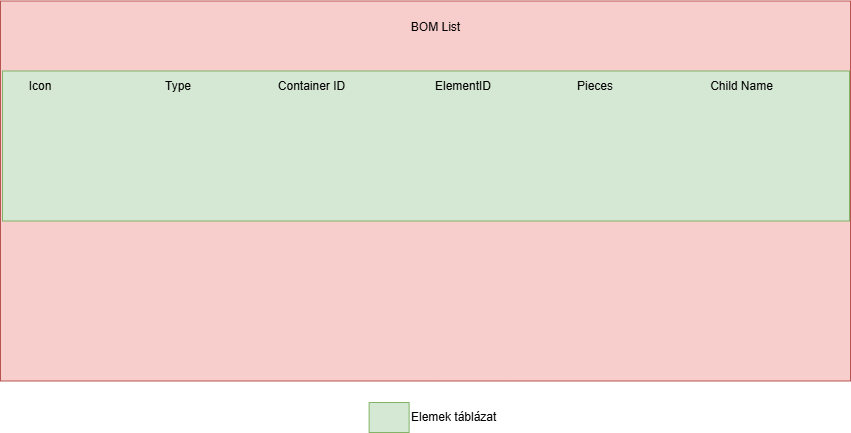
name=parentID value=id

name=childID value=id

* + 1. Elemek részletei oldal / BOM lista aloldal

36. ábra Új elem hozzáadása felugró ablak

Egy táblázatos formában megjeleníti a kibontott BOM listát. Módosításra itt nincs lehetőség, csak a tartalmaz oldalon.

**Elemek táblázat:**

HTML tag: table

CSS:

table-striped table-dark

width: 90%;

margin-left: auto;

margin-right: auto;

Az első oszlopban egy miniatűr kép jelenik meg. A többi oszlopban az adatbázis ide vonatkozó mezője.

37. ábra BOM lista aloldal

* + 1. Elemek részletei oldal / Fa nézet aloldal

****A hierarchia vizuális megjelenítésére szolgál. Egy eleme van, amibe a python kód készíti el a szükséges HTML kódot, így a front-end csak megjeleníti a kész kódot.

**Fa nézet:**

HTML tag:div

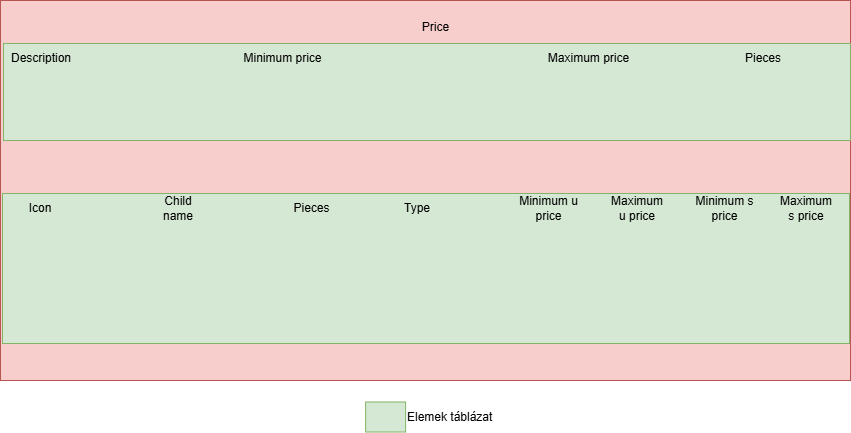
38. ábra Fa nézet aloldal

Div name: tree

* + 1. Elemek részletei oldal / Ár aloldal

Ez az oldal az elemhez köthető árazási információkat mutatja meg.

Két táblázatra bontható. A felső egy összesített táblázat, amelyben a szükséges munkaórák és a szükséges anyagi javak jelennek meg.

Az alsó táblázatban tételesen felsorolásra kerül az összes tétel, amiből az összesítés készült.

39. ábra Ár aloldal

Mind a két táblázat ugyan olyan stílussal rendelkezik.

**Elemek táblázat:**

HTML tag: table

CSS:

table-striped table-dark

width: 90%;

margin-left: auto;

margin-right: auto;

* + 1. Média oldal

Ezen az oldalon az összes már feltöltött fájl megtekinthető, letölthető és törölhető.

Egy balra rendezett flex-boxban helyezkedik el minden elem.

**Egy ilyen elem leírása:**

Egy ilyen fájl kinézetét valósítja meg. Minden fájl külön ilyen konténert kap.

HTML tag: div

CSS: card bg-light text-dark m-2 rounded

**Ezen belül a felső kép adatai:**

HTML tag: img

CSS: card-img-top

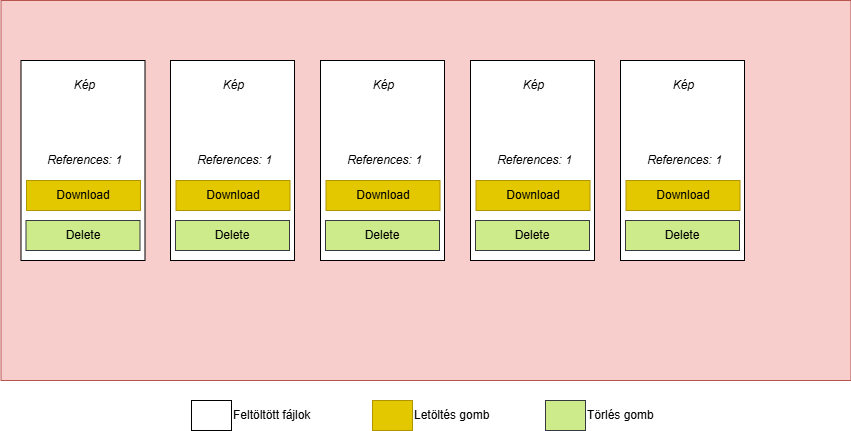
height:150px;

**Ez alatt egy div helyezkedik el a következő adatokkal:**

HTML tag: div

CSS: card-body

Tartalomként a leírás szövege egy h4-es tagben “card-title” class-ba foglalva.

Ez alatt a references felirat dőlt típusú és középre igazítva jeleníti meg az adatbázisból számított értéket.

40. ábra Média oldal

**Letöltés gomb:**

HTML tag: a

CSS: btn btn-primary

Megjelenítendő szöveg: Download

A link a feltöltött fájl referanciájára mutat.

**Törlés gomb:**

HTML tag: a

CSS: btn btn-danger

Megjelenítendő szöveg: Delete

A link a feltöltött fájl referanciájára mutat és egy felugró ablak után törli az elemet a rendszerből.

* + 1. Beszállítói oldal

A felső sorban egy gomb jelenik meg, melynek kattintására egy felugró ablak jeleik meg. A felugró ablakban lehetőség van új beszállítók hozzáadására.

Az oldal alsó részén egy táblázat jelenik meg az adatbázisban rögzített beszállítókról. Módosításra nincs lehetőség, csupán az utolsó oszlopban lévő törlés gomb aktív.

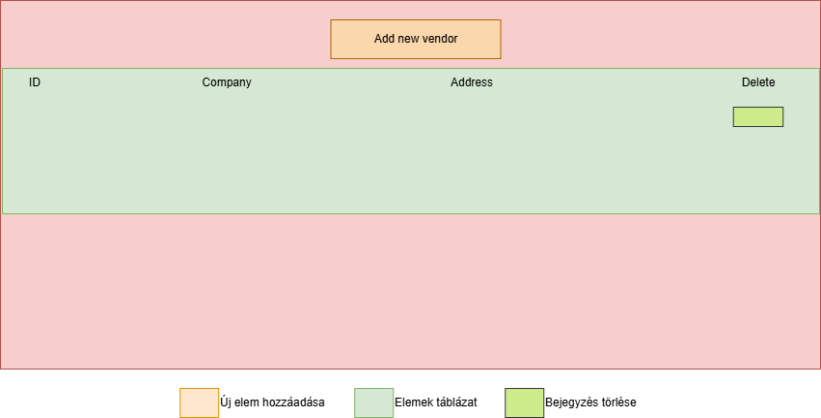
**Új elem hozzáadása:**

Középre helyezett. Kattintásra egy felugró ablak jelenik meg, amelyben hozzá lehet adni új elemet. A felugró ablak egy bootsrap modal elem. A felugró ablak elfogadásával az elem törlődik. Ehhez a következő URL meghívása szükséges: “/vendors/delete/<id>”

HTML tag: button

CSS: btn btn-primary mt-3 mb-3

Modal ID: #AddNewElementModal

**Elemek táblázat:**

41. ábra Beszállítói oldal

HTML tag: table

CSS:

table-striped table-dark

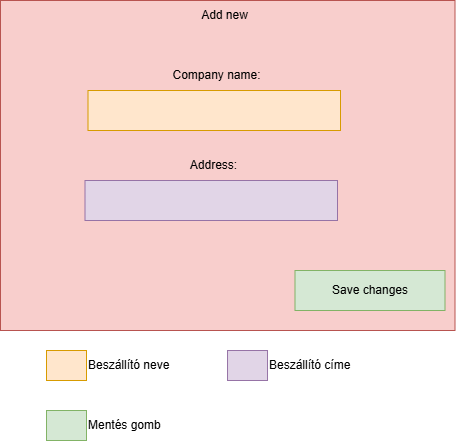
width: 90%;

margin-left: auto;

margin-right: auto;

* + 1. Beszállítói oldal / Új beszállító hozzáadása felugró ablak

A felugró ablak egy modal megjelenítésében jön létre. A form POST metódussal küldi az adatokat a szervernek a következő címre: “/vendors/add”

**Modal:**

42. ábra Új beszállító hozzáadása felugró ablak

HTML tag: div

CSS:

position: relative;

display: -ms-flexbox;

display: flex;

-ms-flex-direction: column;

flex-direction: column;

width: 100%;

pointer-events: auto;

background-color: #fff;

background-clip: padding-box;

border: 1px solid rgba(0, 0, 0, .2);

border-radius: .3rem;

outline: 0;

**Beszállító neve:**

kötelező mező

HTML tag: input>text

CSS: form-control

Name: name

**Mentés gomb:**

HTML tag: input>submit

CSS: btn btn-primary

**Beszállító címe:**

kötelező mező

HTML tag: input>text

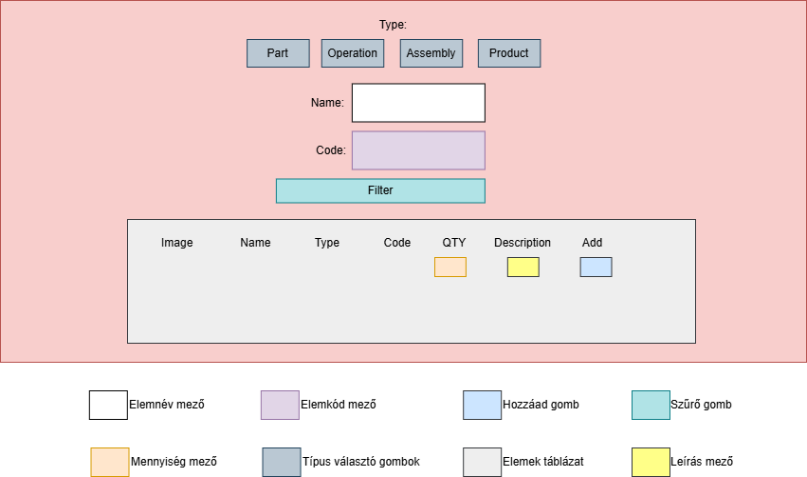
CSS: form-control

Name: address

* + 1. Raktárkészlet oldal

A felső sorban egy szűrő form található meg. Ennek legelső eleme egy négy elemű választó gomb a négy különböző típusú elem típusra. Másik két mezője az elemnév és elemkód. A Filter gombra kattintva a szűrő aktiválódik.

Alatta egy listában a megjelenítendő elemek foglalnak helyet egy táblázatban. Az utolsó három eleme a táblázatnak egy mező, ami a mennyiséget tartalmazza, a leírást mezőt és a hozzáadás gombot.



**Elemnév mező:**

kötelező mező

HTML tag: input>text

Name: name

Elemkód mező:

kötelező mező

HTML tag: input>text

Name: code

43. ábra Raktárkészlet oldal

**Típus választó gombok:**

kötelező mező

HTML tag: input>radio

CSS: btn-check

Name: type

Az értékek az 5 mezőhöz: {part, operation, assembly, product, project}

**Szűrő gomb:**

HTML tag: form>submit

CSS: btn btn-primary mt-2 w-100

A form POST metódost használ a “#” végcélra.

Az alsó táblázat egy középre igazított elem. Minden sor egy külön form a két benne foglalt mezővel. Az első oszlopa egy miniatűr képet jelenít meg az elemről

**Táblázat adatai:**

HTML tag: table

CSS:

table-striped table-dark mb-3

width: 90%;

margin-left: auto;

margin-right: auto;

**Hozzáad gomb:**

HTML tag: input>submit

CSS: btn btn-success

**Mennyiség mező:**

kötelező mező

HTML tag: input>number

Name: qty

**Leírás mező:**

kötelező mező

HTML tag: input>text

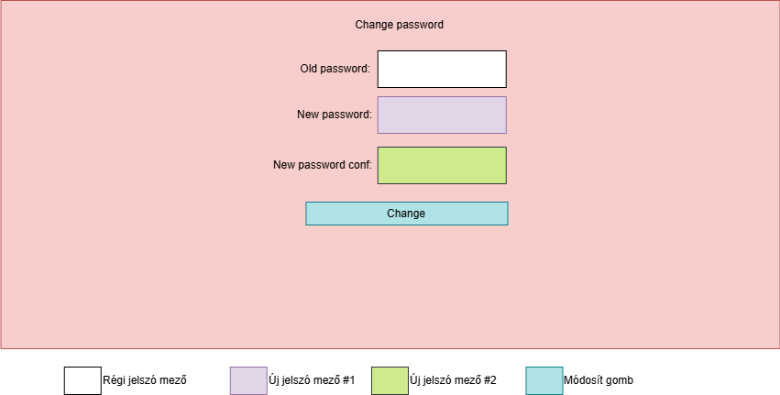
name:description

**A hozzáadás form rejtett mezői:**

name=formType value=addToInventory

name=elementID value=id

* + 1. Jelszókezelő oldal

Ezen az oldalon tud a felhasználó jelszót változtatni. Ehhez egy form megjelenítésére van szükség, amely 3 mezőt tartalmaz. A form POST metódussal küldi az adatokat a “#” célra.

44. ábra Jelszókezelő oldal

**Régi jelszó mező:**

kötelező mező

HTML tag: input>password

Name: old\_password

**Új jelszó mező #1:**

kötelező mező

HTML tag: input>password

Name: new\_password

**Új jelszó mező #2:**

kötelező mező

HTML tag: input>password

Name: new\_password2

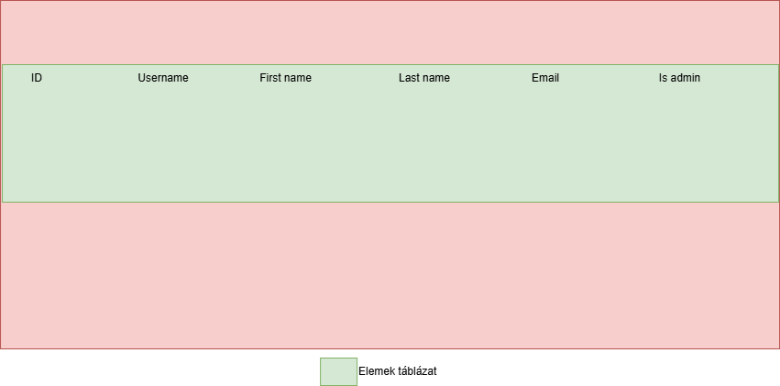
**Módosít gomb:**

HTML tag: input>submit

Megjelenítendő szöveg: Save changes

* + 1. Felhasználók oldal

Ezen az oldalon az összes felhasználó adatai megtekinthetőek, de nem szerkeszthetőek. . A megjelenítés egy táblázatos formában történik.

**Elemek táblázat:**

45. ábra Felhasználók oldal

HTML tag:table

CSS:

table-striped table-dark

width: 90%;

margin-left: auto;

margin-right: auto;

1. Szakirodalom
2. KSH: 20.8.1.11. Teljes munkaidőben alkalmazásban állók bruttó átlagkeresete foglalkozások szerint [Forrás](https://www.ksh.hu/stadat_files/mun/hu/mun0059.html)
3. SAP (2025.02.08.) Multilevel BOM. [Forrás](https://community.sap.com/t5/enterprise-resource-planning-blogs-by-sap/multilevel-bill-of-material-bom/ba-p/13515956)
4. Autodesk (2025. Február) Introducing Bill of Materials (BOM) for Autodesk Fusion. [Forrás](https://www.autodesk.com/products/fusion-360/blog/introducing-bill-of-materials-bom-autodesk-fusion/)
5. MRPeasy (2025) MRPeasy - Integrated BOM solutions. [Forrás](https://www.mrpeasy.com/)
6. Katana (2025) MRP. [Forrás](https://katanamrp.com/features/production-management/)
7. Sweigart, A. (2020) Automate the boring stuff with python [Forrás](https://automatetheboringstuff.com/)
8. Martin Lindström (2022), Containers & Virtual machines, [Forrás](https://www.diva-portal.org/smash/get/diva2:1665606/FULLTEXT01.pdf)
9. Portainer Official Documentation (2025) [Forrás](https://docs.portainer.io/)
10. Wikipédia – DevOps, (2025) [Forrás](https://en.wikipedia.org/wiki/DevOps)
11. Atlassian (2025.02.23.) Atlassian – DevOps **[Hiba! A hiperhivatkozás érvénytelen.](https://www.atlassian.com/devops)**[Forrás](https://www.atlassian.com/devops)
12. Adrian Holovaty, J. K. (2006). The Definitive Guide to Django: Web Development Done Right. [Forrás](http://slav0nic.org.ua/static/books/python/The%20Definitive%20Guide%20to%20Django%20-%20Apress.pdf)
13. SZTE. (2025.) Adatbázis gyakorlat, 1. NF. [Forrás](https://www.inf.u-szeged.hu/~gnemeth/adatbgyak/exe/Normalizalas_web/1_normlforma_1nf.html)
14. RealPython (2024. December 15.) Object-Oriented Programming (OOP) in Python. Forrás: Real Python: [Forrás/](https://realpython.com/python3-object-oriented-programming)
15. Wikipédia – CRUD [Forrás](https://en.wikipedia.org/wiki/Create,_read,_update_and_delete)
16. Medium. (2021. Március 29.) You shouldn’t store your images/videos in a database. [Forrás](https://medium.com/ensias-it/you-shouldnt-store-your-images-videos-in-a-database-6a78ffa277b2)
17. Johnson, P. C. (2025. Március). Dynamic and Static Testing. [Forrás](https://www.dcs.gla.ac.uk/~johnson/teaching/safety/powerpoint/15_Testing.pdf)
18. GitHub Official Documentation [Forrás](https://docs.github.com/en)
19. Pezze, M., & Michal, Y. (2008). Software Testing and Analysis: Process, Principles, and Techniques. [Forrás](https://ix.cs.uoregon.edu/~michal/book/Samples/book.pdf)
20. Rustam Sabirov (2023) A guide to black box vs. white box testing [Forrás](https://qase.io/blog/black-box-vs-white-box-testing/)
21. Multithreading in Python: The Ultimate Guide (2022) [Forrás](https://www.dataquest.io/blog/multithreading-in-python/)
22. Deploying Gunicorn (2025) [Forrás](https://docs.gunicorn.org/en/latest/deploy.html)
23. mysqlslap — A Load Emulation Client [Forrás](https://dev.mysql.com/doc/refman/8.4/en/mysqlslap.html)
24. AWS - Elastic Load Balancer [Forrás](https://docs.aws.amazon.com/pdfs/elasticloadbalancing/latest/userguide/elb-ug.pdf)
25. MariaDB Official Documentation [Forrás](https://mariadb.com/database-topics/scalability/)
26. phpMyAdmin Official Documentation [Forrás](https://docs.phpmyadmin.net/en/latest/)

# 