



ZÁRÓDOLGOZAT

Készítették:

Bihari Csaba – Gaál Gergely Benjámin 2/14E

Konzulens:

Farkas Zoltán

Miskolc

2025.

Miskolci SZC Kandó Kálmán Informatikai Technikum

Miskolci Szakképzési Centrum

SZOFTVERFEJLESZTŐ- ÉS TESZTELŐ SZAK

Dokumentáció

Warehouse webapplikáció

Bihari Csaba – Gaál Gergely Benjámin

Konzulens: Farkas Zoltán

2024-2025

Tartalomjegyzék

Tartalom

1.	Témaválasztás indoklás	1
2.	A felhasznált technológiák bemutatása	2
2.1.	Adatbázis – MySQL.....	2
2.2.	Backend – ASP.net core 8, Entity Framework.....	2
2.3.	Frontend – React, React-Router-Dom, Vanta, JWT	3
2.4.	Verziókezelés – Git/GitHub.....	4
3.	Felhasznált fejlesztői környezetek bemutatása.....	4
3.1.	Adatbázis - XAMPP (adatbázis host-olás)	4
3.2.	Backend – Visual studio	6
3.3.	Frontend – Visual studio Code.....	7
4.	Felhasznált projektmanagement eszközök	8
4.1.	Trello	8
4.2.	Discord	9
5.	A szoftver bemutatása.....	9
5.1.	Frontend, webes felhasználói felület működése	9
5.2.	Backend.....	18
5.3.	Adatbázis.....	23
6.	Irodalomjegyzék	26
7.	Ábra jegyzék.....	26
8.	Mellékletek	27
8.1.	Github link.....	27
8.2.	Trello link.....	27

1. Témaválasztás indoklás

Személyes munkatapasztalataink alapján ezt a témát találtuk a legideálisabbnak a projektünk elkészítéséhez, amihez a mindennapokban előforduló észrevételek és hiányosságok inspiráltak minket. A modern vállalkozások működésében a készletgazdálkodás és az erőforrások hatékony kezelése kiemelkedő szerepet játszik. A raktárak, mint logisztikai központok, kulcsszereplői a termelés és az értékesítés közötti kapcsolatnak. A digitális megoldások, mint a raktárkezelő rendszerek (Warehouse Management Systems – WMS), lehetővé teszik a készletek nyilvántartását, mozgásainak követését és a leltározási folyamatok automatizálását. A raktárkezelés célja, hogy minimalizálja a készletek kezelése során fellépő hibákat, biztosítsa a termékek gyors és pontos elérhetőségét, valamint optimalizálja a tárolási helyek kihasználtságát.

A projektünk célja egy webes alapú raktárkezelő rendszer létrehozása, amely segíti a raktározási folyamatok hatékonyabb kezelését, átláthatóságát és automatizálását. A rendszer képes különböző termékek nyilvántartására, azok állapotának követésére, valamint felhasználói jogosultságkezelésre is.

A fejlesztéshez modern technológiákat alkalmaztunk:

- **Frontend:** React (modern, komponens alapú felhasználói felület)
- **Backend:** ASP.NET Core Web API (megbízható és skálázható REST API)
- **Adatbázis:** MySQL (strukturált adattárolás és relációk támogatása)
- **Authentikáció:** JWT tokenekkel támogatott szerep alapú jogosultságkezelés
- **ORM (Object-Relational Mapping):** Entity Framework Core

A rendszer három felhasználó jogosultsági szinttel dolgozik:

- **User:** új anyagot rögzíthet, de nem törölhet és módosíthat
- **Super User:** az anyagokat módosíthatja és törölheti
- **Admin:** minden funkcióhoz teljes hozzáféréssel rendelkezik + felhasználókat módosíthat és törölhet

2. A felhasznált technológiák bemutatása

2.1. Adatbázis – MySQL

A MySQL egy széles körben használt nyílt forráskódú relációs adatbázis-kezelő rendszer (RDBMS), amely jól illeszkedik a készletkezelő rendszerhez, mivel képes hatékonyan tárolni a strukturált adatokat és kezelni a relációkat. Az adatok biztosítják a rendszer számára a szükséges információkat, például a termékek nyilvántartását, a felhasználói adatokat és a raktárak helyszíneit. Először 1995-ben vezették be, és azóta a világ egyik legnépszerűbb adatbázisrendszerévé vált.



2.2. Backend – ASP.net core 8, Entity Framework

Az ASP.net keretrendszert a .Net fejlesztőkörnyezet része, amely lehetővé teszi különféle webes alkalmazások és szolgáltatások gyors és hatékony fejlesztését. A keretrendszer nyílt forráskódú, multiplatform (fut Windows, Linux és Mac platformokon), különféle eszközöket, nyelveket (C#, F#) és függvény könyvtárakat tartalmaz. Kiadási ciklusa éves, egy hosszabb és egy rövidebb támogatási periódus váltja egymást. Jelenleg a 9-es kiadás a legfrissebb, ez egy rövidebb támogatási periódussal rendelkező kiadás. A rendszer moduláris, a modulok megfelelő alkalmazásának kombinációjával az alábbi applikáció feladatokat képes ellátni többek között:

Applikáció típus	Feladat(hozzávaló eszköz)
Web applikáció	új, server oldali web UI fejlesztés (Razor)
Web applikáció	MVC architektúra alapú web applikáció fejlesztés
Web applikáció	kliens oldali UI fejlesztés (Blazor)
WEB API	RESTful HTTP szolgáltatások (web API, minimal API)
gRPC applikáció	contact-first szolgáltatások Protocol Bufferek használatával
valós idejű applikáció	kétirányú kommunikáció megvalósítása server és kliens között

Tartalmaz továbbá például biztonsági eszközöket autentikációra, autorizációra, CORS-ra {Cross origin resource sharing}, valamint tartalmaz még egyéb, a korszerű web alkalmazások fejlesztése során felmerülő feladatok elvégzésére szolgáló eszközöket.

Bár nem az ASP.NET részre, az adatok adatbázisból történő elérését Entity Framework-kön keresztül képes kezelni, amely egy ORM szoftver és szintént a .NET környezet része.

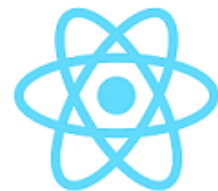
A feladat során felhasznált elemei az ASP.NET keretrendszernek:

- WEB API applikáció template : RESTful HTTP végpontok kialakítása és szolgáltatások kezelése kontrollerek segítségével.
- ASP.NET core Identity rendszert, ami az authozizációs és autentikáció feladatokat látja el.
- Swagger - ami lehetővé teszi a végpontok tesztelését

Entity framework: kapcsolat létrehozása az adatbázis és web api alkalmazás között

2.3. Frontend – React, React-Router-Dom, Vanta, JWT

A React egy modern JavaScript könyvtár, amelyet a felhasználói felületek fejlesztésére használnak. A komponens alapú megközelítés lehetővé teszi, hogy a felhasználói felület elemei újrafelhasználhatók és könnyen karbantarthatók legyenek. A React legnagyobb előnye a gyors renderelés, amely a virtuális DOM-nak köszönhetően gyors választ időt biztosít, különösen akkor, ha az alkalmazás komplex adatokat kezel. Emellett a React közösségi támogatása és ökoszisztémája (pl. React Router) segít a fejlesztés felgyorsításában.



A **React-Router-DOM** egy népszerű könyvtár a React alkalmazásokban, amely lehetővé teszi az ügyféloldali útvonalkezelést és a különböző komponensek közötti navigációt. Ezáltal egyoldalas alkalmazásokat (SPA) hozhatunk létre, ahol a felhasználók zökkenőmentesen navigálhatnak az oldalak között anélkül, hogy az egész oldal újratöltődne.

A **Vanta** egy JavaScript könyvtár, amely lenyűgöző 3D animált háttérhatások hozzáadását teszi lehetővé weboldalakhoz. React alkalmazásokban is könnyedén integrálható, így interaktív és látványos háttereket hozhatunk létre.

A **JWT (JSON Web Token)** egy nyílt szabvány (RFC 7519), amely lehetővé teszi a biztonságos adatátvitelt egy kliens és egy szerver között egy tömörített és aláírt token formájában. Webalkalmazásokban jellemzően **felhasználói hitelesítésre és jogosultság kezelésre** használják.

2.4. Verziókezelés – Git/GitHub

A Git egy nyílt forráskódú verziókezelő rendszer, amelyet Linus Torvalds fejlesztett ki a Linux kernel fejlesztésének támogatására. Lehetővé teszi a fájlok különböző verzióinak nyomon követését, a módosítások dokumentálását és a csapatmunkát, akár offline környezetben is. A Git gyors és megbízható, használható helyileg a saját gépeden vagy egy távoli szerveren keresztül, parancssoros és grafikus felületeken egyaránt.

A GitHub egy webalapú platform, amely a Git verziókezelő rendszerre épül. Lehetővé teszi a Git-adattárak (repositoryk) tárolását, megosztását és kezelését az interneten keresztül. A GitHub segítségével könnyedén együttműködhetsz más fejlesztőkkel, nyomon követheted a projektek előrehaladását, és hozzájárulhatsz nyílt forráskódú projektekhez.

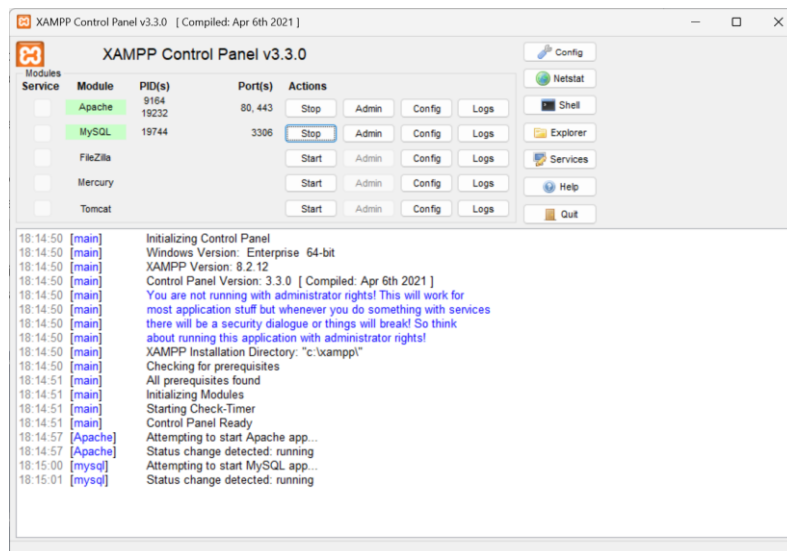
Fontos megjegyezni, hogy míg a Git egy verziókezelő szoftver, addig a GitHub egy szolgáltatás, amely a Gitet használja a kódok tárolására és kezelésére. Tehát a Git használatához nem feltétlenül szükséges a GitHub, de a GitHub a Git funkcionalitására építve kínál további szolgáltatásokat, mint például a kódmegosztás és az együttműködés támogatása.



3. Felhasznált fejlesztői környezetek bemutatása

3.1. Adatbázis - XAMPP (adatbázis host-olás)

A XAMPP egy olyan ingyenes, multiplatform keretrendszer, amely több szintén ingyenes webfejlesztéshez használt szoftvert képes futtatni létrehozva egy olyan környezetet amiben egyaránt megtalálható Apache web server, MySQL phpMyAdmin felülettel, FileZilla FTP szerver, stb , külön szolgáltatásonként indítva őket.

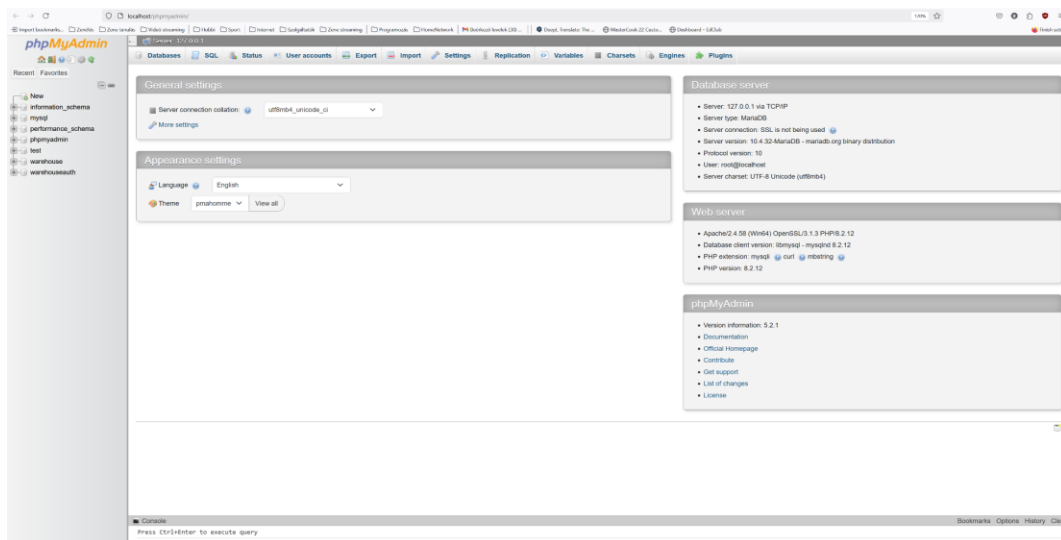


1. kép XAMPP indítókép

Elérhető szolgáltatások:

- Apache webservert - HTTP szerver szoftver
- MySQL adatbázis szerver phpMyAdmin kezelőfelülettel: Alapvetően a MySQL csak egy konzolos kezelőfelülettel rendelkezik és különféle, saját parancsokon keresztül lehet használni. A phpMyAdmin ebben segít, egy webes böngészős felületen keresztül lehetővé teszi, hogy kényelmesen bonyolítsuk az adatbázis kezelés támasztotta feladatokat.(adatbázisok, táblák, kapcsolatok, triggerek létrehozása, manipulálása, exportálása, importálása, stb)
- FileZilla - FTP fájlserver
- Mercury - levelező szerver
- Tomcat - szintén HTTP szerver, de teljesen Java alapú web applikációnak biztosít saját környezetet.

A feladat megoldása során csak a Apache webserver és a MySQL+phpMyAdmin volt igénybe véve a számos szolgáltatás közül.



2. kép phpMyadmin fő felülete

3.2. Backend – Visual studio

A Microsoft által fejlesztett és karbantartott Visual Studio integrált fejlesztői környezet (IDE) 1997-es megjelenése óta van velünk, több jelenleg használatos programnyelvet támogat, szorosan kötődik a Microsoft .Net fejlesztői környezethez, melynek számos tagjához tartalmaz specifikus eszközöket.

Jelenlegi aktuális verziója a Visual Studio 2022, 3 különféle kiadásban érhető el, amely verzióként egyre szélesedő eszközkészletével lefedi a közösségi programozók igényeitől egészen a kis -közepes és nagyvállalatokkal bezáróan.

A 3 kiadás:

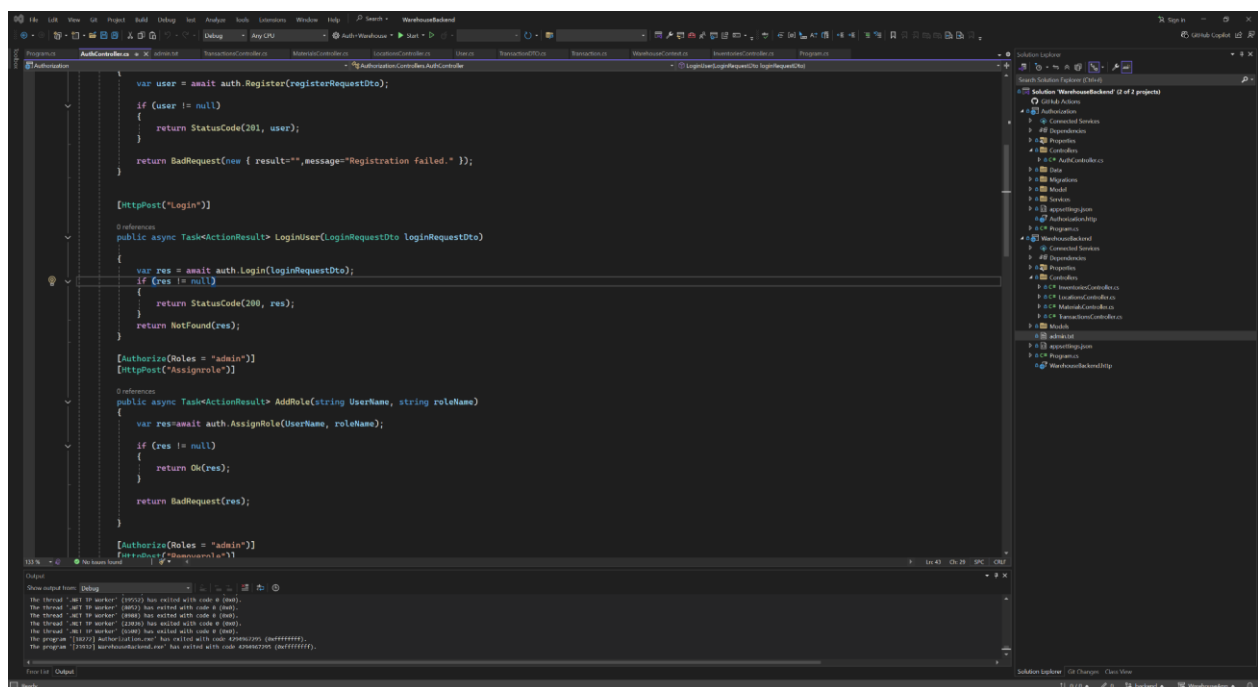
- Community edition: ingyenes verzió, nem kereskedelmi forgalomba szánt szoftverek fejlesztéséhez, mint pl nyílt forráskódú, vagy oktatási, kutatási céllal készült szoftverekhez.
- Professional edition: igényli licenc vásárlását, kereskedelmi forgalomba kerülő szoftverek fejlesztését teszi lehetővé. Tudása is több mint a community edition-nek, például MSDN támogatást tartalmaz vásárolt license- től függően.
- Enterprise edition: Ez a verzió még további eszközöket tartalmaz a korábbi verziókhoz képest. Szintén kereskedelmi forgalomba kerülő szoftverek fejlesztését teszi lehetővé.

Főbb részei:

- kódszerkesztő: az alap, máshol is megszokott tartalmaz nyelvfüggő intellisense megoldást (kifejezések egyfajta automatikus kiegészítése kódolás közben az adott nyelvi szintaktikáknak megfelelően) , és

újabbán AI képességekkel lett kiegészítve amely tovább gyorsítja a kód írását azáltal hogy javaslatot tesz egész kód részek beírására.

- Debugger
- Tervezők (designer-ek): pl MS Form és WPF applikációk UI-jának tervezésre alkalmas, de tartalmaz ezeken felül több tervezőt is.
- Tesztelő eszközök
- egyéb eszközök: például tartalmaz NuGet csomagkezelőt, amelynek segítségével különböző, .Net keretrendszerhez készült szoftverkomponensekkel kiegészíthetjük programunk képességeit, így azokat már saját magunknak nem szükséges kifejleszteni. .



3. kép Visual Studio felülete

3.3. Frontend – Visual studio Code

A **Visual Studio Code (VS Code)** egy ingyenes, nyílt forráskódú, platformfüggetlen forráskód-szerkesztő, amelyet a Microsoft fejlesztett. Elérhető Windows, macOS és Linux operációs rendszereken, és célja, hogy egyesítse a kódszerkesztők egyszerűségét a fejlett fejlesztői eszközök funkcionalitásával. A Visual Studio Code célja, hogy egy könnyű, mégis erőteljes eszközt biztosítson a fejlesztők számára, amely támogatja a modern webes és felhő alapú alkalmazások fejlesztését.

Főbb jellemzők:

- **IntelliSense:** Fejlett kódkiegészítési funkció, amely szintaktikai kiemelést és automatikus kódkiegészítést kínál, segítve a fejlesztőket a gyorsabb és pontosabb kódírásban.
- **Beépített Git integráció:** Lehetővé teszi a verziókezelést közvetlenül az editorból, megkönnyítve a kódváltozások nyomon követését és kezelését.
- **Bővíthetőség:** Számos kiterjesztés érhető el a Visual Studio Code Marketplace-en keresztül, amelyek további funkciókkal bővítik az editort, például új programozási nyelvek támogatásával vagy egyedi eszközökkel.
- **Beépített terminál:** Az editoron belül közvetlen hozzáférést biztosít a parancssorhoz, lehetővé téve a fejlesztők számára, hogy anélkül futtassanak parancsokat, hogy elhagynák az editort.
- **Debugging támogatás:** Lehetővé teszi a kód hibakeresését közvetlenül az editorból, támogatva a töréspontok beállítását, a változók figyelését és a kód lépésenkénti végrehajtását.

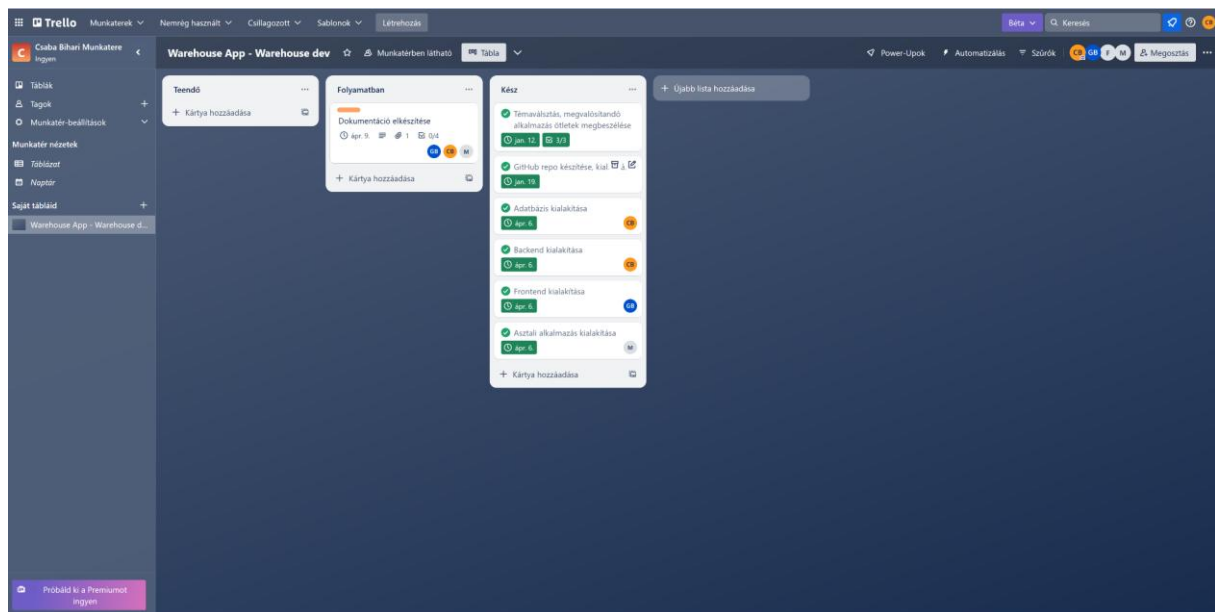


4. Felhasznált projektmanagement eszközök

4.1. Trello

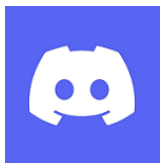
A Trello egy web alapú projektmanagement eszköz amelyen elősegíti mindenféle több résztvevős projektfeladat (nem csak szoftverfejlesztés) tervezését és lebonyolításának követését. Jelen világunkban rendkívüli módon felértékelődött a csapatmunka, mert a megoldandó feladatok egyre komplexebbekké váltak, valamint a megvalósítására rendelkezésre álló idő is egyre rövidebb.

Mivel webes alkalmazás lehetővé teszi, hogy a csapat tagjai földrajzilag különálló helyeken legyenek. A platformnak létezik ingyenes, korlátozott funkcionalitású változata (ezt használtuk jelen esetben) ami Kanban rendszer használatát teszi lehetővé egy tábla használatával, de kanban táblánkat több emberrel is megoszthatjuk, így kis projektek esetében ez is elég lehet. Az előfizetős verzió további, összetettebb projektmanagement eszközöket is tartalmaz, mint pl: több kártya, adatexportálás, stb.



4. kép Trello felülete

4.2. Discord



A Discord egy ingyenes, online kommunikációs platform, amely lehetővé teszi, hogy könnyedén tartsd a kapcsolatot barátaiddal, családoddal vagy akár kollégáiddal. Hang-, videó- és szöveges üzenetek révén bármikor beszélgethatsz, oszthatsz meg tartalmakat, vagy éppen közösen játszhattok online. Eredetileg videojátékosok számára készült, hogy egyszerűbben kommunikálhassanak játék közben, de mára sokkal szélesebb körben használják. Különböző közösségek, baráti társaságok, tanulócsoporthok és vállalkozások is találják hasznosnak.

5. A szoftver bemutatása

5.1. Frontend, webes felhasználói felület működése

A **StorageDevs Warehouse Management System** kezdőoldala a felhasználót egy vizuálisan megkapó, de informatív üdvözlő képernyővel fogadja. Az oldal közepén egy áttetsző, kékes panelen helyezkedik el a bemutatkozó szöveg, amely összefoglalja az alkalmazás alapvető funkcióit és használati lehetőségeit. A háttérben egy logisztikai raktárról készített videó helyezkedik el (hang nélkül). A platform célja, hogy lehetővé tegye a felhasználók számára a készlet egyszerű és hatékony nyomon követését, szerkesztését és kezelését.

A kezdőoldal szövege tájékoztat arról is, hogy a rendszer különböző jogosultsági szinteket támogat:

- A **Materials** és **Locations** menüpontok lehetővé teszik az anyagok és tárolóhelyek hozzáadását, módosítását vagy törlését.

- Az **Inventory** menüpont a jelenlegi készletek megtekintésére szolgál.
- A **Transaction** menüpont pedig minden anyagmozgást dokumentál.

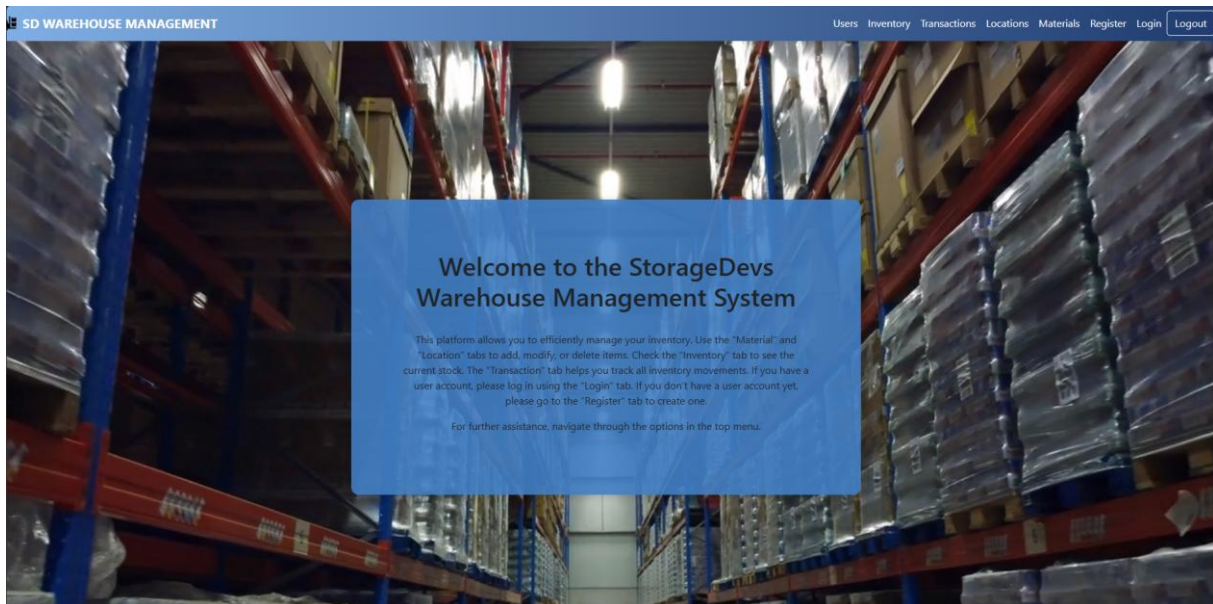
Amennyiben a felhasználó **Admin** jogosultsággal rendelkezik, elérhető számára a **Users** fül is, ahol új felhasználókat lehet regisztrálni vagy meglévőket módosítani és törölni.

Ez a bejelentkező rendszer lehetővé teszi a több szintű hozzáférés szabályozását: például a *User* nem törölhet anyagokat, míg a *Super User* már igen. Az *Admin* pedig a teljes rendszer felett rendelkezik, beleértve a felhasználók kezelését is.

A kezdőképernyő dizájnya egyszerre modern és funkcionális. A fejléc sávbán (Navbar) bal oldalon a rendszer logója és neve található, míg a jobb oldalon egy klasszikus menüsor sorakozik:

- **Inventory** - Összes készlet listázása
- **Transactions** - Tranzakciók
- **Locations** - Tárhelyek
- **Materials** - Anyagok
- **Register** - Regisztrációs felület
- **Login** - Bejelentkezési felület
- **Logout** - Kijelentkezés

A fejléc hátterét és kialakítását Bootstrap technológia segítségével készítettük el.

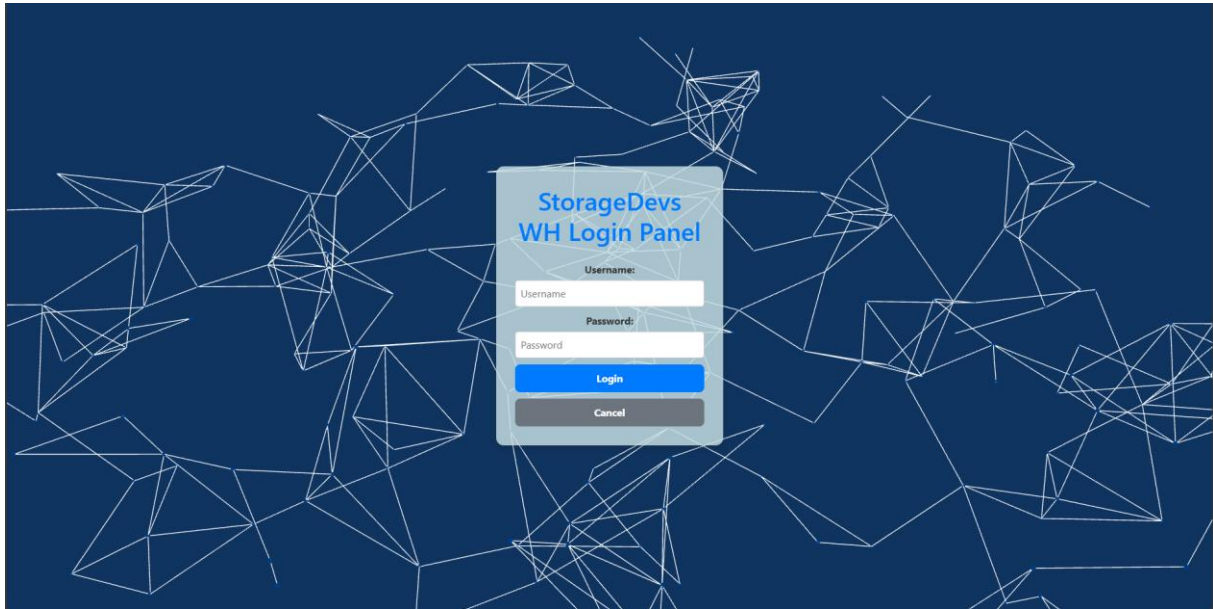


5. kép applikáció kezdőoldal

Ez az elrendezés gyors és átlátható navigációt biztosít a felhasználók számára. A felhasználó belépés után egyből láthatja, hogy mely funkciók elérhetők számára. Ez kulcsfontosságú a jogosultság-alapú működés szempontjából. A háttérként szolgáló videó vagy kép nem csupán díszítőelem: erőteljesen utal a rendszer valós alkalmazási területére, a logisztikai raktárak világára.

Bejelentkezés és kijelentkezés:

A belépési felület egy letisztult, modern megjelenésű login panelnek készült, amely a rendszer biztonságos használatának elsődleges kapujaként szolgál. A dizájn célja, hogy egyszerű és intuitív hozzáférést biztosítson a különböző felhasználók számára, miközben vizuálisan is illeszkedik a rendszer egészéhez.



6. kép bejelentkező oldal

A háttérben egy dinamikus, absztrakt animáció látható, amely vonalhálós pontkapcsolatokkal dolgozik. Ezt a Vanta könyvtár dinamikus 3D-s háttérének segítségével hoztuk létre. Ez egy modern webes esztétikát tükröz, miközben technológiai érzetet közvetít a felhasználó felé. A középen elhelyezkedő login panel áttetsző, világos háttérrel és kék árnyalatú gombbal rendelkezik.

A "**StorageDevs WH Login Panel**" elnevezésben a „WH” rövidítés a „Warehouse”, vagyis a raktár rövidítést takarja.

A rendszer validálja a felhasználónév és jelszó mezők tartalmát. Amennyiben ezek nem egyeznek egy érvényes felhasználói fiókkal, a felhasználót egy **hibajelzés** figyelmezteti:

“Wrong username or password!”.

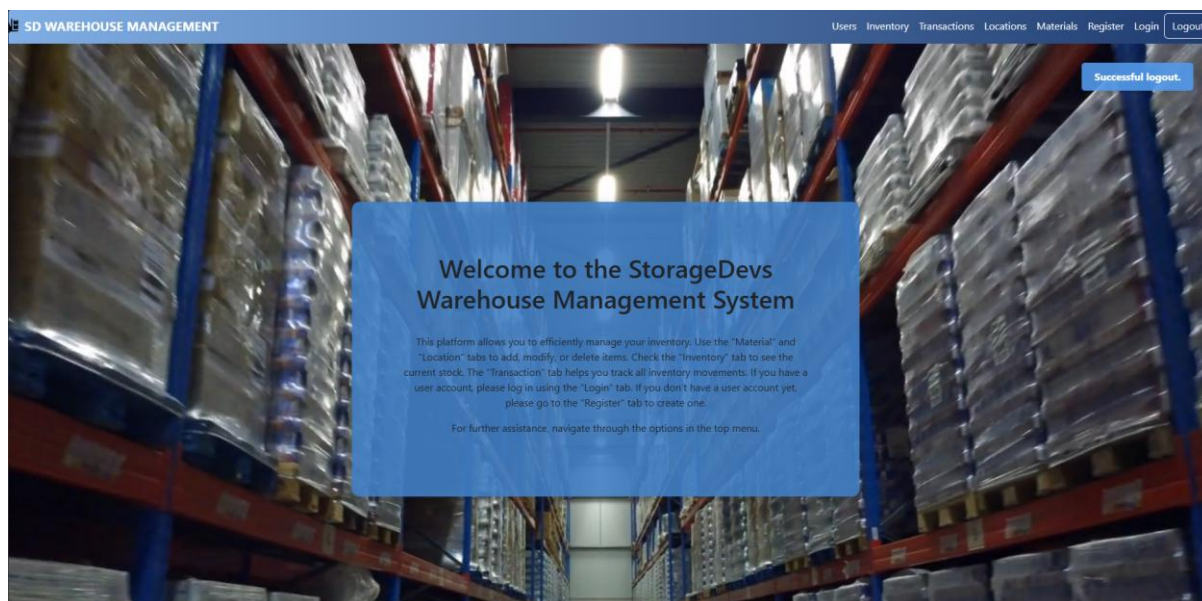
Ez a hibaüzenet **piros színnel** jelenik meg közvetlenül az űrlapmezők felett, jól látható módon. A hibakezelés célja, hogy azonnali visszajelzést adjon a felhasználónak, és lehetőséget biztosítson az adatok javítására. A bejelentkezési adatok ellenőrzése egy backend oldali API hívás során történik. A megadott hitelesítési adatok alapján a szerver ellenőrzi az adatbázisban szereplő felhasználókat, és amennyiben sikeres a belépés, egy **JWT (JSON Web Token)** kerül visszaküldésre, amely a felhasználói munkamenethez szükséges. Ez a token lehetővé teszi, hogy a felhasználó a rendszer többi részét elérje az engedélyei függvényében: pl. **User**,

SuperUser vagy **Admin** szint szerint. A szintek részletes leírása a felhasználói felületi részen kerülnek bemutatásra.

A felhasználó a felső menüsávban található **Logout** gomb segítségével jelentkezhet ki a rendszerből. A sikeres kijelentkezést követően a képernyő közepén egy kis visszajelző mező jelenik meg, amely a következő szöveget tartalmazza:

„Successful logout.”

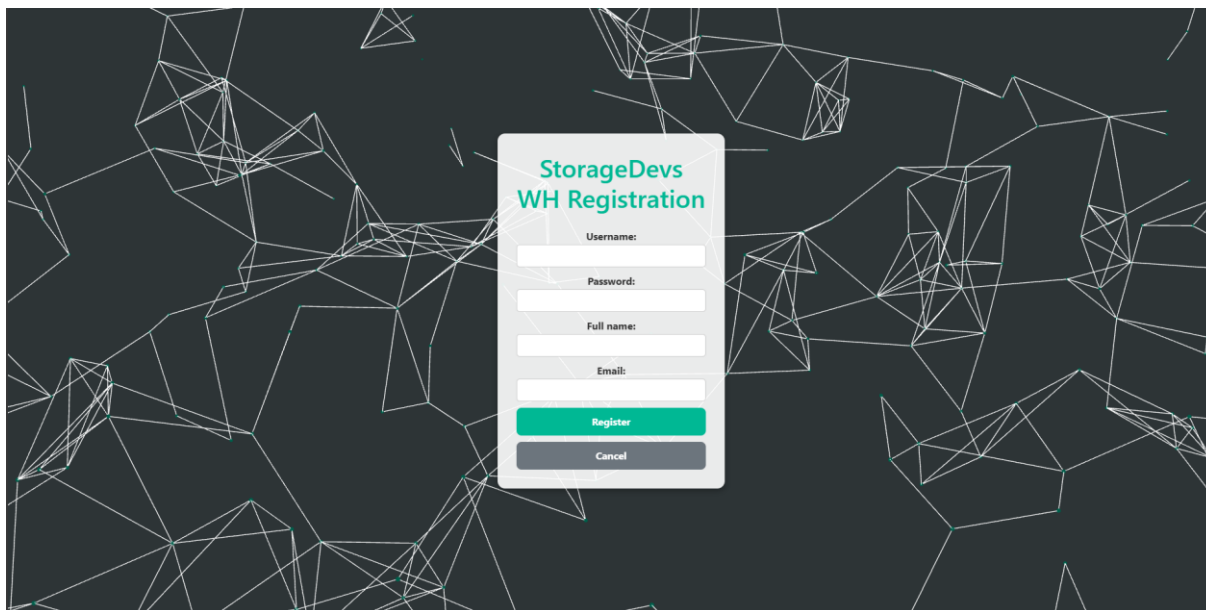
Ez az üzenet **3 másodperc** után automatikusan eltűnik, ezzel is biztosítva, hogy a felhasználó megerősítést kapjon a művelet sikerességéről, anélkül hogy bármilyen külön interakcióra lenne szükség.



7. kép kijelentkezés

Regisztráció:

A rendszer regisztrációs felülete szintén egy átlátható, korszerű webes űrlap hozzáadott Vanta könyvtárral, mint ahogy a login panelnél is láthatjuk. A cím türkiz-zöld árnyalatú betűtípussal jelenik meg, amely vizuálisan elkülönül a mezők szürkés típusától. A felület három mezőt tartalmaz a fiók létrehozásához: „**Username**”, „**Password**”, „**Full name**” és „**Email**”. Az űrlap alján található egy türkiz-zöld „**Register**” gomb, amely az adatok beküldését végzi.



8. kép regisztrációs oldal

A „**Cancel**” gombra kattintva vissza jutunk a kezdő oldalunkra a bejelentkezés és a regisztráció során is.

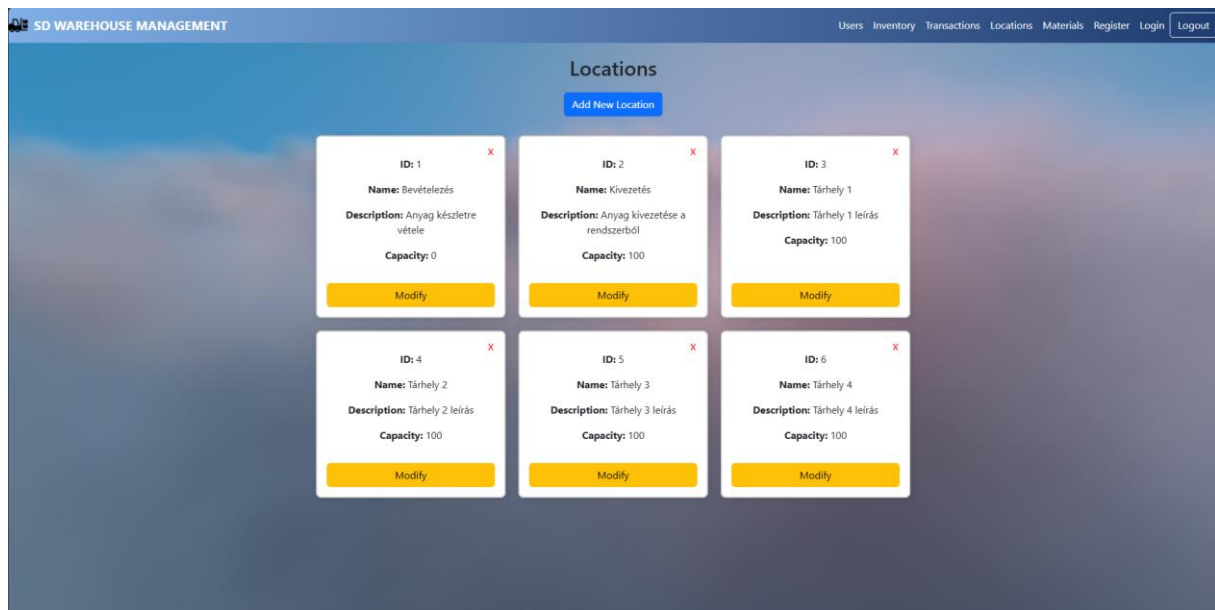
A regisztráció során többféle hiba is ellenőrzésre kerül:

- Üres mezők esetén a rendszer figyelmeztet a mező kitöltésére.
- Ha a jelszó és a jelszó megerősítése nem egyezik, a következő angol nyelvű hibaüzenet jelenik meg a mezők fölött **piros színnel**:
"Passwords do not match"
- Ha a felhasználónév már foglalt, akkor szintén piros hibaüzenet látható:
"Username already taken"
- Ha minden egyezik, akkor a rendszer ezzel a szöveggel fog 2 másodperc múlva elnavigálni minket a login oldalunkra:
"Registration successful! Redirecting to login..."
- Ha egyéb probléma merül fel, akkor a rendszer az alábbi hibát fogja kiírni:
"Registration failed. Username might be taken."
- A regisztráció csak akkor lép életbe, ha az e-mail címben szerepel a “@” karakter.

Felhasználói felület:

Locations:

A *Locations* oldal az alkalmazás azon felülete, amely a raktárban található különböző helyszínek/tárhelyek – például tárolók, be- és kivezetési pontok kezelését szolgálja. Az oldal célja, hogy a felhasználó átlátható módon megtekintse, módosítsa, törölje vagy újonnan felvegye a lokációs adatokat a rendszerbe.



9. kép elérhető lokáció listája

Az "Admin" és a "Superuser" felhasználók kétféle műveletet hajthatnak végre az egyes kártyákon:

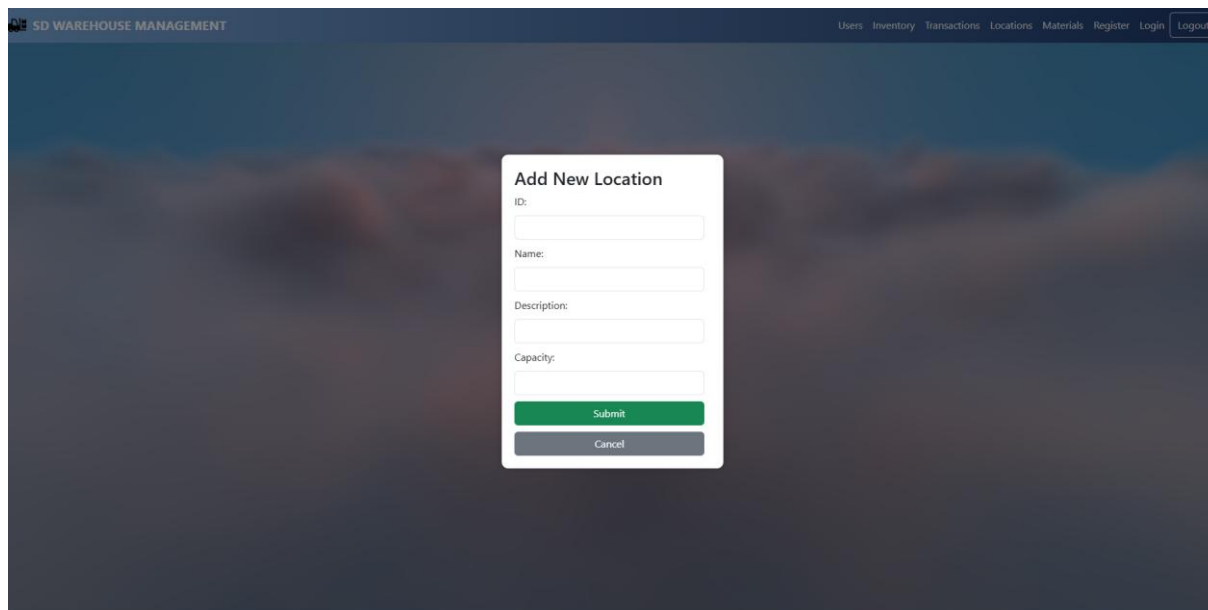
- **Modify gomb:** A kártya alján elhelyezett sárga színű gomb megnyomásával a kiválasztott helyszín adatai szerkeszthetők. Ez ugyan úgy, ahogy a hozzáadásnál is - egy formában történik, amely lehetővé teszi az értékek módosítását.
- **Törlés ikon (X):** A jobb felső sarokban található piros "X" ikon segítségével a kiválasztott lokáció véglegesen törölhető a rendszerből.

A felhasználók szerepköre befolyásolja az oldalak funkcionalitását:

- **User:** új anyagot rögzíthet, de nem törölhet és módosíthat
- **Super User:** az anyagokat módosíthatja és törölheti
- **Admin:** minden funkcióhoz teljes hozzáféréssel rendelkezik + felhasználókat módosíthat és törölhet

A "Locations" oldal cím közepén jelenik meg, alatta egy jól látható, "Add New Location" feliratú kék gomb található, amely új lokáció hozzáadására szolgál. Ennek megnyomásával egy felugró űrlap jelenik meg, ahol az új helyszín részletei (név, leírás, kapacitás stb.) adhatók meg. A

háttér ezen esetben elsötétedik, hogy a felhasználó könnyen tudjon koncentrálni a beviteli mezőkre.

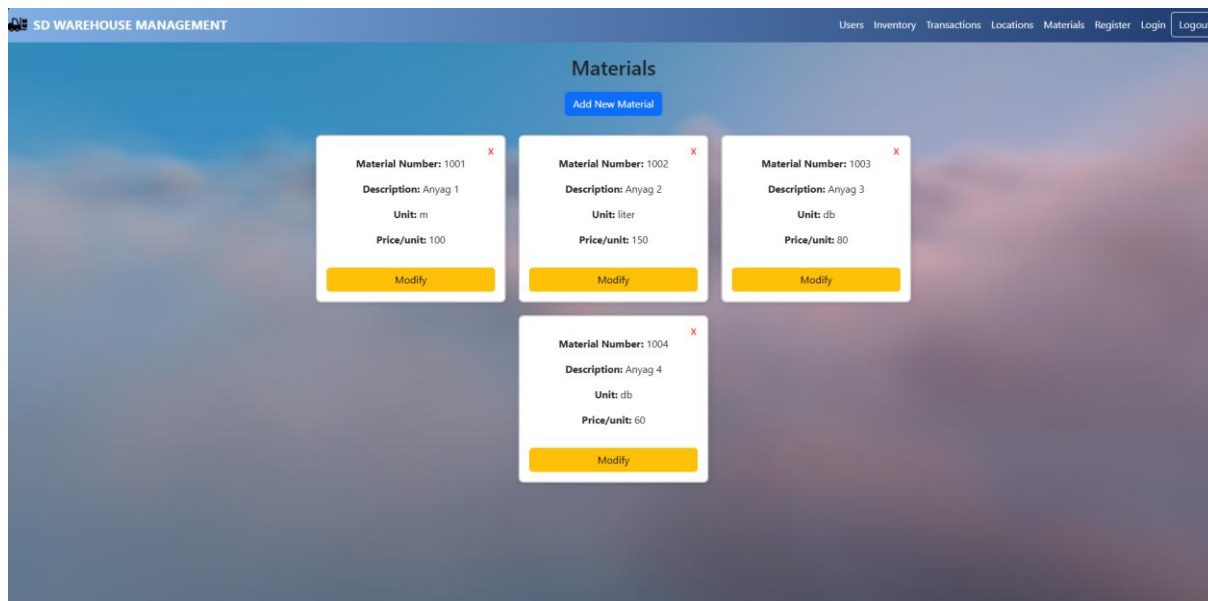


The screenshot shows the 'Add New Location' modal form. It has a dark background with a blurred image. The form is white and contains the following fields: ID, Name, Description, and Capacity. Each field has a corresponding input box. Below the fields are two buttons: 'Submit' (green) and 'Cancel' (grey). The top navigation bar includes 'Users', 'Inventory', 'Transactions', 'Locations', 'Materials', 'Register', 'Login', and 'Logout'.

10. kép lokáció hozzáadása felület

Materials:

A *Materials* komponens hasonló logikát követ, mint a *Locations* oldal. Az anyagok kezelése során a kártyák az adott anyag nevét, leírását, mennyiségét, mértékegységét és a hozzájuk rendelt tároló helyet jelenítik meg. Az interakciós elemek és a megjelenítési elvek megegyeznek, így a felhasználói élmény egységes marad az alkalmazás különböző részein.

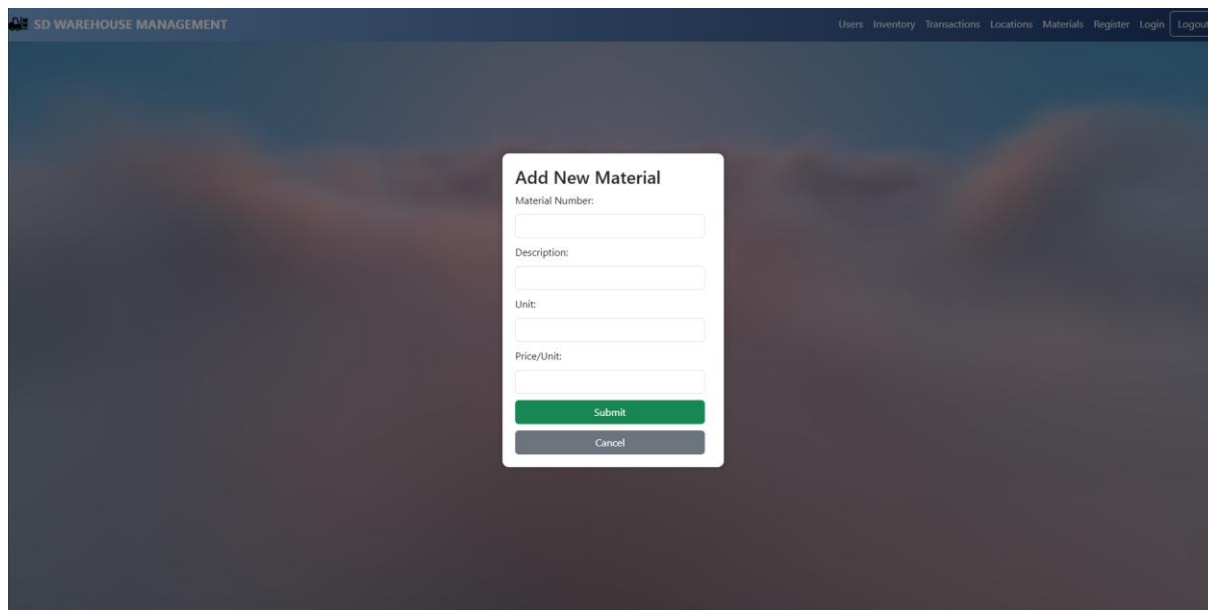


The screenshot shows the 'Materials' page. It has a dark background with a blurred image. At the top, there is a blue button labeled 'Add New Material'. Below this, there are four material cards. Each card displays the following information: Material Number, Description, Unit, and Price/unit. Each card also has a yellow 'Modify' button at the bottom. The top navigation bar includes 'Users', 'Inventory', 'Transactions', 'Locations', 'Materials', 'Register', 'Login', and 'Logout'.

11. kép elérhető anyagok listája

A *Materials* oldal az alkalmazás egyik központi felülete, amely az aktuálisan nyilvántartott anyagok kezelésére szolgál. Ez az oldal lehetőséget biztosít új anyagok rögzítésére, meglévők

szerkesztésére vagy törlésére, továbbá jól strukturált módon jeleníti meg a nyilvántartásban szereplő bejegyzéseket. A hozzáadás a következőképpen mutatkozik meg az anyagoknál:



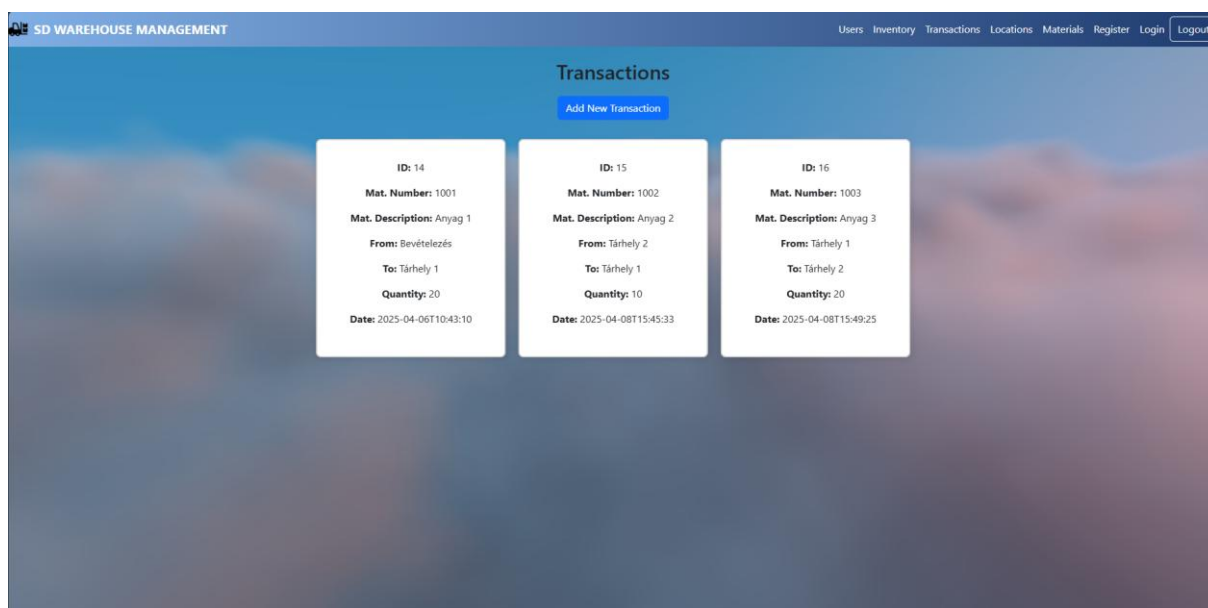
The screenshot shows the 'Add New Material' form in the SD Warehouse Management system. The form is centered on a dark background. It contains the following fields and buttons:

- Material Number:** A text input field.
- Description:** A text input field.
- Unit:** A text input field.
- Price/Unit:** A text input field.
- Submit:** A green button.
- Cancel:** A grey button.

12. kép új anyag hozzáadása felület

Transactions:

A *Transactions* komponens a raktárkezelő rendszer egyik kulcsfontosságú része, amely az anyagmozgások naplózására szolgál. A felhasználók ezen az oldalon keresztül tudják rögzíteni, hogy mely anyag mikor, honnan hová és milyen mennyiségben került átmozgatásra. Ez biztosítja az átláthatóságot és a készletmozgások pontos nyomon követését.

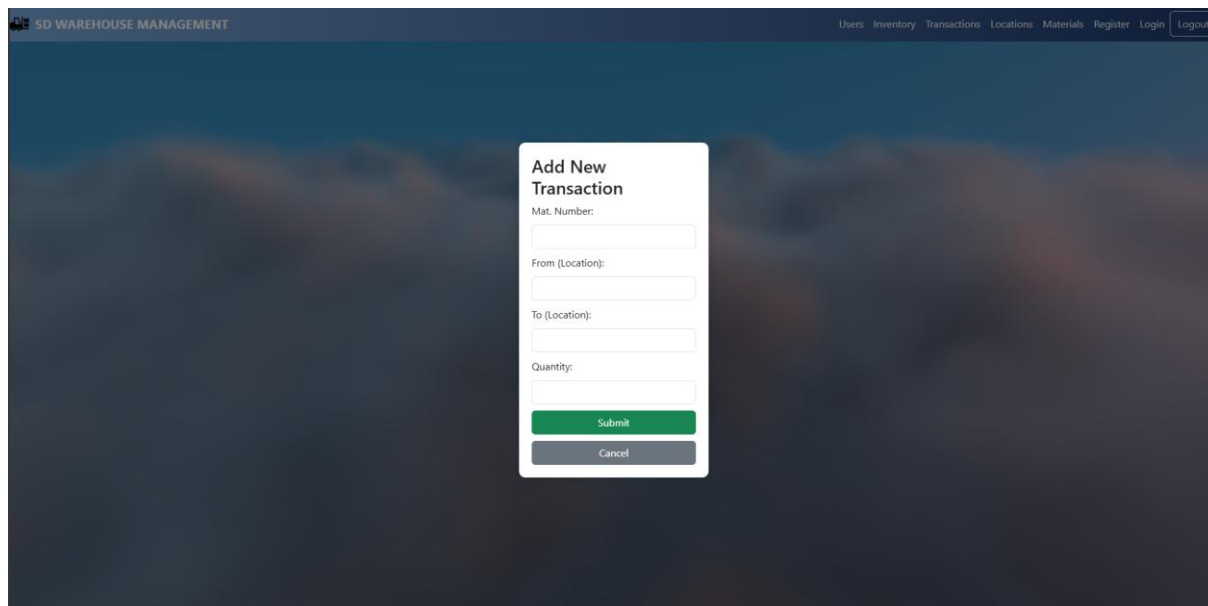


The screenshot shows the 'Transactions' page in the SD Warehouse Management system. The page has a blue header with the title 'Transactions' and a button 'Add New Transaction'. Below the header, there are three white cards displaying transaction details:

ID	Mat. Number	Mat. Description	From	To	Quantity	Date
14	1001	Anyag 1	Bevételezés	Tárhely 1	20	2025-04-06T10:43:10
15	1002	Anyag 2	Tárhely 2	Tárhely 1	10	2025-04-08T15:45:33
16	1003	Anyag 3	Tárhely 1	Tárhely 2	20	2025-04-08T15:49:25

13. kép tranzakciók listája

Az oldalon a felhasználók csak **új tranzakciókat tudnak hozzáadni**, meglévőket **nem lehet módosítani vagy törölni**. Ez a megkötés tudatos döntés, amely biztosítja az események naplózásának sérthetetlenségét, és megelőzi a visszamenőleges adatmanipuláció lehetőségét. Mivel a módosítás és törlés nem engedélyezett, így minden felhasználó, beleértve a Usereket is, **egyenlő hozzáféréssel rendelkeznek**.



14. kép új tranzakció hozzáadása felület

Az **"Add New Transaction"** gombra kattintva egy űrlap jelenik meg, amelyben megadható a tranzakcióhoz szükséges minden adat, beleértve az anyag nevét vagy számát, honnan és hová megy az anyag, valamint az tárolni kívánt mennyiséget.

A hozzáadásnál nem adjuk meg közvetlen az azonosítót és a dátumot, mivel automatikusan hozzáadja az adatbázis, így elkerülhetőek a hibák és biztosított az egységes adatkezelés.

A teljes felület reszponzív kialakítású, a kártyák rácsszerű elrendezésben jelennek meg, amelyek különböző képernyőméret esetén is optimálisan igazodnak a megjelenítő eszközhöz. A dizájn letisztult, modern hatást kelt, vizuálisan jól elkülönítve mutatja be az egyes elemeket, ezáltal könnyen kezelhető és átlátható marad nagy adatmennyiség esetén is.

A felhasználói élmény vizuális fokozása érdekében az alkalmazás összes oldalán **Vanta**-t használunk dinamikus háttérként. A login és register oldalon valamint a kezdőlapon kívül a **Vanta.Clouds** effekt van beépítve az összes további oldalon, amely egy valós időben animált felhőmozgást jelenít meg, reagálva az egérmozgásra és a képernyőméretre.

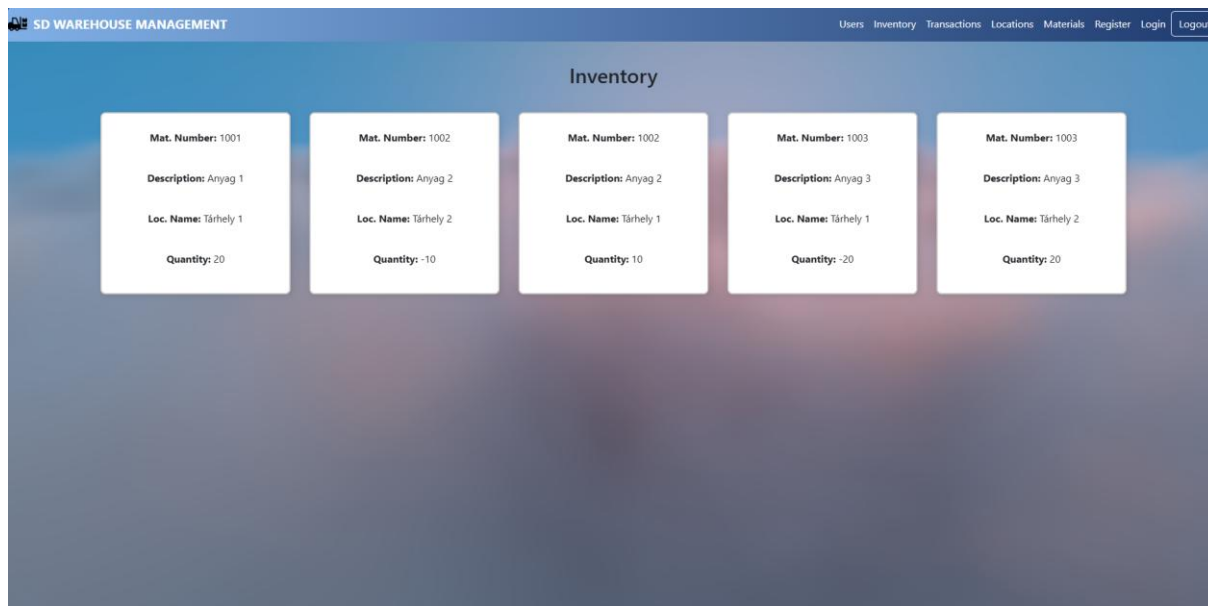
A háttér beépítése **useEffect** hook-on keresztül történik komponens szinten.

A **THREE.js** könyvtárra épül, így szükséges a **three** csomag külön importálása is.

Inventory:

Az Inventory oldal célja az aktuálisan elérhető anyagkészletek vizuális, jól átlátható megjelenítése az egyes raktár helyeken. Ezen az oldalon a program kizárólag csak kiírja az

adatokat és nem lehet további műveletet végrehajtani (hozzáadás, törlés, módosítás), mint a többi lapon.



15. kép készlet lista

Az oldal az alábbi adatokat jeleníti meg kártyák formájában:

- **Mat. Number** – az anyag egyedi azonosítója
- **Description** – az anyag leírása
- **Loc. Name** – a tárolási hely neve
- **Quantity** – az adott anyag mennyisége az adott tárhelyen

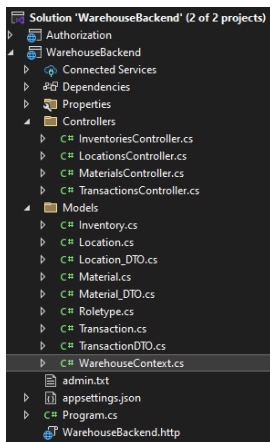
Egy anyag több tárhelyen is tárolható, így akár ugyanaz az azonosító is megjelenhet többször, eltérő helyszínekkel.

5.2. Backend

A szoftver backend része az ASP.NET 8-as verziójában készült Visual Studio fejlesztőkörnyezet segítségével. A WarehouseBackend "solution" két projektet tartalmaz: Authorization és

WarehouseBackend. Mindkettő WEB API template alapú, ami alapvetően meghatározza, hogy vannak felépítve.

Warehouse backend projekt: adatbázissal való összekapcsolódást és annak menedzselését az Entity Framework segítségével végzi, amely úgy működik jelen esetben, hogy leképezi az adatbázis tábláit, mezőit, kapcsolatait C# osztályokká és objektumokká, amelyet a továbbiakban már LINQ segítségével (a C# saját lekérdező, "query" nyelve) közvetlenül tudunk kezelni, nincs szükség arra, hogy folyamatosan SQL lekérdezéseket alkalmazzunk. Ennek a gyakorlati formája a Models könyvtárba található WarehouseContext.cs osztály. A WarehouseContext osztály egy-egy példányát DI (Dependency Injection) segítségével példányosítjuk mikor szükség van rá.



16. kép projekt struktúra

A Model könyvtárban található egyéb, nem DTO-val végződő osztály egy egy adatbázis táblának rekordját reprezentálja, mint adatmodell.

A DTO osztályok feladata az, hogy a kliens szerver kommunikáció során a HTTP üzenetekben csak a szükséges információ kerüljön átadásra az üzenet törzsében. Erre látunk később példát.

Már csak a Controller könyvtárban szereplő Controller-re végződő osztályok ismertetése van hátra. Általánosságban elmondható, hogy a kontrollerek felelnek a HTTP üzenet küldésért és fogadásáért. Az üzenet törzsébe JSON objektumként kerülnek a közölt információk. Nézzünk egy példát, a TransactionController osztályt, majd hogy ez hogyan is néz ki végpontként.

```
namespace WarehouseBackend.Controllers
{
    [Route("api/[controller]")]
    [ApiController]
    public class TransactionsController : ControllerBase
    {
        private readonly WarehouseContext _context;

        public TransactionsController(WarehouseContext context)
        {
            _context = context;
        }

        #region // GET: api/Transactions
        [Authorize(Roles = "user,superuser,admin")]
        [HttpGet]
        public async Task<ActionResult<IEnumerable<Transaction>>> GetTransactions()
        {
            // GET: api/Transactions/5
            // POST: api/Transactions
        }
        #endregion
    }
}
```

17. kép osztály szerkezete

1. Az osztály attól lesz web api controller, hogy szerepel benne az [ApiController] attribútum (amely controller specifikus viselkedéseket engedélyez, a [Route...] attribútum, amely megadja milyen URL útvonalon érhető el a controller és annak metódusai, valamint hogy az osztály a ControllerBase osztály egy leszármazottja, ami lehetővé teszi az alapfunkciókat (HTTP válaszüzenetek pl.).
2. Szerepel még az osztályban a WarehouseContext beinjektálása konstruktoron keresztül, ami az adatbázis műveletek miatt fontos.
3. Itt jön az aktuális végpont definiálása. Fontos részei:

- *[Authorization..]* attribútum, amely a végpont védelmét hivatott ellátni, és csak a zárójelben szereplő role-al rendelkező felhasználó képesek elérni. Ez elhagyható, de ekkor nem lesz védve, bárki küldhet rá kérést.
- A *[HttpGet]* attribútum amely megadja, hogy milyen típusú kéréseket kezel ez a végpont.
- Majd jön egy asszinkron módban (TASK) végrehajtott ActionResult típusú Gettransaction() függvény amit azt definiálja hogy mit tartalmazzon a válasz. Egy kontroller több végpontot is tartalmazhat, és a [HttpGet]-en kívül lehetőség van többfajta végpont létrehozása ([HttpPut], [HttpPost], stb)

Lássuk részletesebben ezt a függvényt:

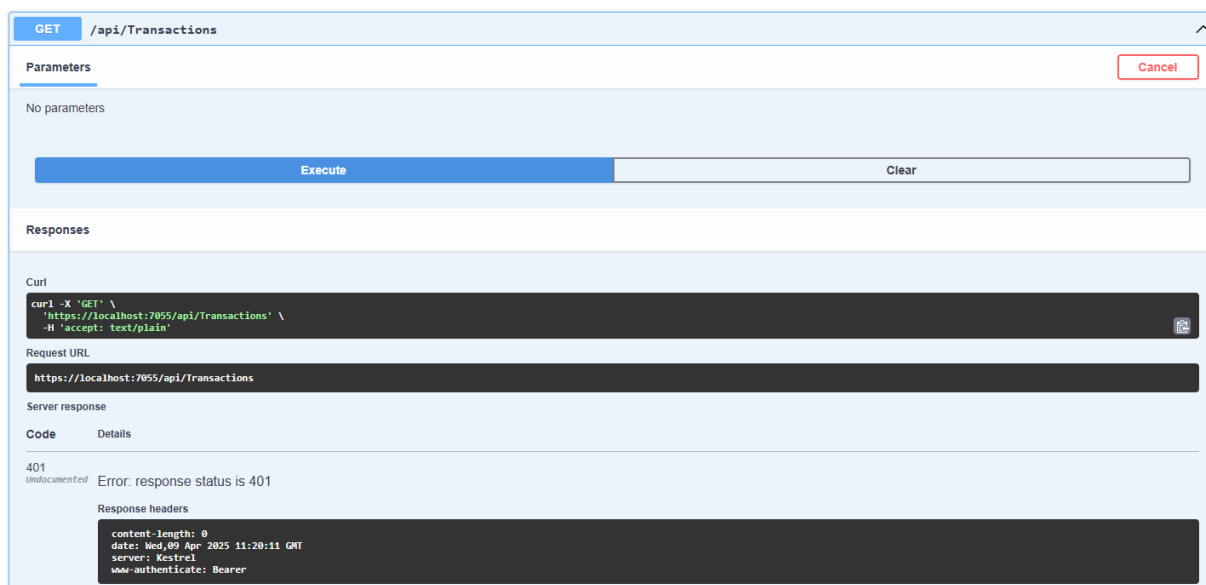
```
public async Task<ActionResult<IEnumerable<Transaction>>> GetTransactions()
{
    try
    {
        var result = await
        (
            from transaction in _context.Transactions
            join material in _context.Materials
            on transaction.MaterialId equals material.MaterialId
            join locationFrom in _context.Locations
            on transaction.TransactionFromId equals locationFrom.LocationId
            join locationTo in _context.Locations
            on transaction.TransactionToId equals locationTo.LocationId
            select new GetAllTransaction...
        ).ToListAsync();

        return Ok(new { Result = result, message = "Successfull request" });
    }
    catch (Exception ex)
    {
        return StatusCode(500, new { Result = "", Message = ex.Message });
    }
}
```

18. kép végpont szerkezete

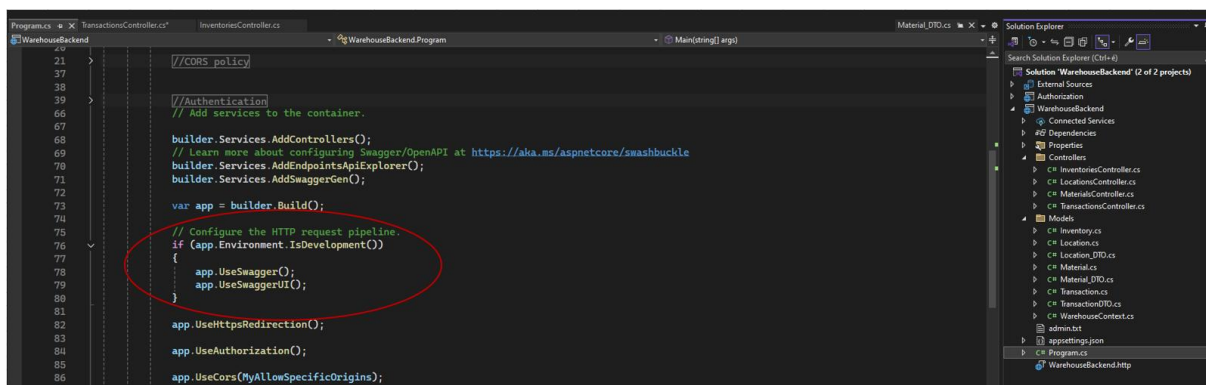
Ez ennek a függvénynek a feladata a transaction táblában lévő összes tranzakció adatainak összegyűjtése és a kliens számára történő elküldése. Tartalmaz egy LINQ [1] lekérdezést, amely az entity framework "táblák" összekapcsolásán keresztül egy lista objektumba, majd egy OK státuszkódú üzenetben, törzsében a json formátumra alakított listával válaszol. Amennyiben valamilyen kivétel történik a művelet során, akkor a 500-as státuszkódú, hibaüzenetet tartalmazó üzenetet küld vissza a kliens felé.

Hogy mindez hogyan néz ki a fogadó oldal szemszögéből azt a swagger nevű program segítségével lehet tesztelni. Ez küld egy üzenetet a megfelelő végpontra, és megjeleníteni a szervertől kapott választ.



19. kép swagger felület

A Swagger használata a ASP.NET 8 as verziójának része, a projekt program.cs osztályában kerül beállításra az alábbi képen látható módon, így tesztelhetjük a kialakított végpont viselkedését.



20. kép swagger hozzáadása projekthez

Az projekthez tartozó végpontok az alábbi képen láthatóak. Lényegében alap, CRUD műveleteket hajtanak végre a táblákon. Mindegyik végpont védett, és user role függő az elérhetősége. Az inventory csak GET műveletet tartalmaz, mert annak tartalma egy, a Transaction controller, POST művelete után végrehajtódó trigger módosítja.

Inventories		^
GET	/api/Inventories	▼
GET	/api/Inventories/CustomQuery	▼
Locations		^
GET	/api/Locations	▼
POST	/api/Locations	▼
GET	/api/Locations/{id}	▼
PUT	/api/Locations/{id}	▼
DELETE	/api/Locations/{id}	▼
Materials		^
GET	/api/Materials	▼
POST	/api/Materials	▼
GET	/api/Materials/{id}	▼
PUT	/api/Materials/{id}	▼
DELETE	/api/Materials/{id}	▼
Transactions		^
GET	/api/Transactions	▼
POST	/api/Transactions	▼
GET	/api/Transactions/{id}	▼

21. kép létrehozott warehouse végpontok







Authentication projekt

A Microsoft Identity csomag felhasználásával készült. Feladata az autentikáció (felhasználó azonosítása, regisztrálás, jelszó útján) valamint az autorizáció (milyen részeit jogosult a programnak a használni az adott felhasználó). Ebben a projektben (vagyis a hozzátartozó adatbázisban) kerül tárolásra a felhasználó adatai (pl. jelszó megfelelő titkosítással), jogosultságai, valamint ez felel a belépés után előállítani azt a JWT token, aminek birtokában az adott felhasználó elérheti a felhasználói szerepköréhez tartozó végpontokat. Ebben a projektben található végpontokon keresztül lehet a felhasználó managementet végrehajtani. A definiált végpontok:

Auth		^
POST	/auth/Register	▼
POST	/auth/Login	▼
POST	/auth/Assignrole	▼
POST	/auth/Removerole	▼
DELETE	/auth/DeleteUser	▼
GET	/auth/GetAllUser	▼

22. kép létrehozott autentikációs végpontok

A projektekben felhasznált csomagok:

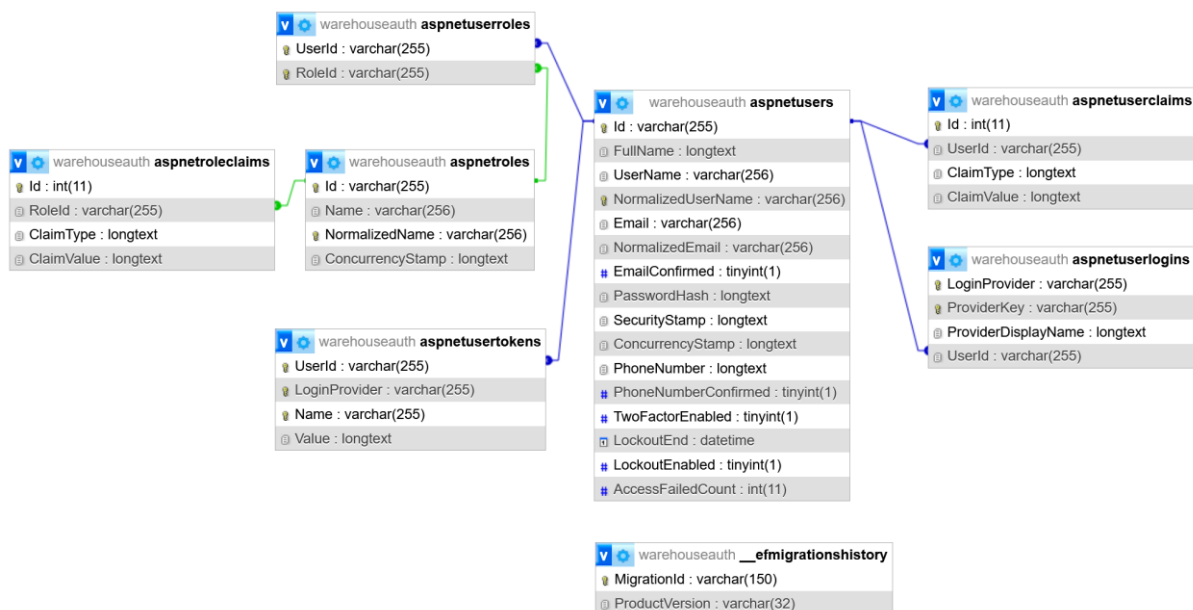
	Microsoft.AspNetCore.Authentication.JwtBearer by Microsoft	8.0.14
	ASP.NET Core middleware that enables an application to receive an OpenID Connect bearer token.	9.0.4
	Microsoft.AspNetCore.Identity.EntityFrameworkCore by Microsoft	8.0.14
	ASP.NET Core Identity provider that uses Entity Framework Core.	9.0.4
	Microsoft.EntityFrameworkCore.Tools by Microsoft	8.0.13
	Entity Framework Core Tools for the NuGet Package Manager Console in Visual Studio.	9.0.4
	Microsoft.VisualStudio.Web.CodeGeneration.Design by Microsoft	8.0.7
	Code Generation tool for ASP.NET Core. Contains the dotnet-aspnet-codegenerator command used for generating controllers and views.	9.0.0
	MySQL.EntityFrameworkCore by Oracle Corporation	8.0.11
	MySQL.EntityFrameworkCore adds support for Microsoft Entity Framework Core.	9.0.0
	Swashbuckle.AspNetCore by domaindrivendev	6.6.2
	Swagger tools for documenting APIs built on ASP.NET Core	8.1.0

23. kép felhasznált csomagok

5.3. Adatbázis

Az alábbiakban a szoftver adatbázis oldala kerül bemutatásra. Nem cél a technikai részletek ismertetése, sokkal inkább az adatbázis szerkezete és funkciójának megismerése.

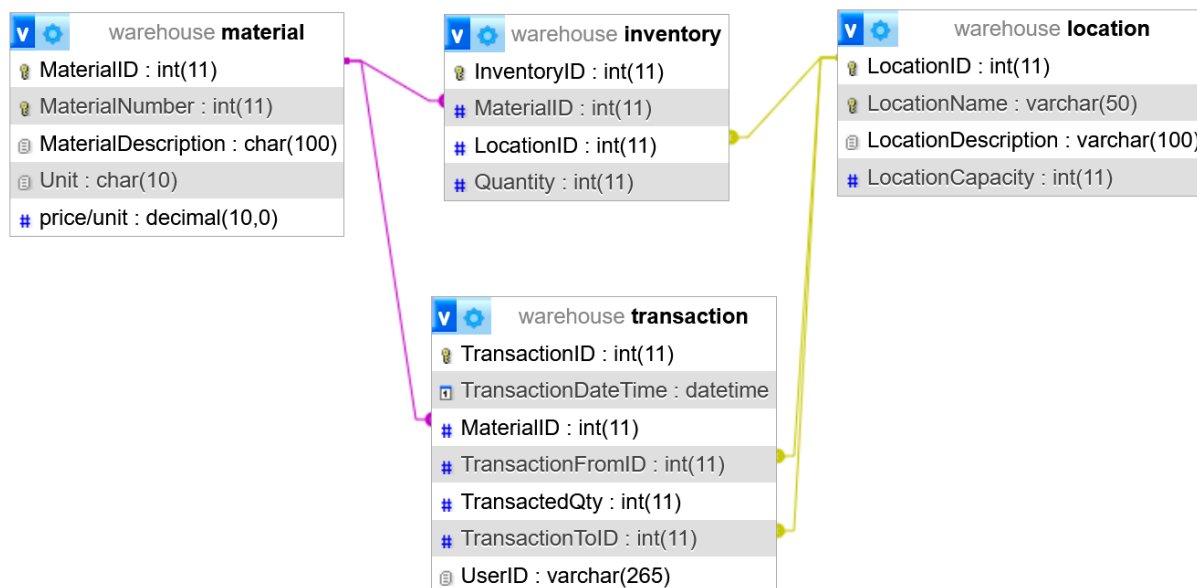
A szoftver a gyakorlatban két adatbázist tartalmaz. Az egyik adatbázis magát a raktározási műveletekkel kapcsolatos adatokat tartalmazza (warehouse.sql), míg a másik az ASP.NET core Identity nevezetű autorizációs és autentikációs moduljának adatbázisát (warehouseauth.sql), amelyben a felhasználói accountok és jogkörök és tokenek kerülnek tárolásra. Mindkét adatbázis MySQL adatbázis kezelő rendszeren fut. Először a felhasználó menedzser adatbázist mutatom be néhány szóban.



24. kép autorizációs adatbázis szerkezete

Az adatbázis szerkezetét maga az Identity modul határozza meg, de lehetőség van arra, hogy az *aspnetusers* táblát, - amely a felhasználói attribútumokat tartalmazza - saját attribútumokkal egészítsük ki. Az adatbázist Entity Framework-el hozza létre és menedzseli az Identity modul megfelelő osztálya.

A korábban már említett másik adatbázis, ami magát a raktározási feladatot látja el a következő megfontolások szerint lett kialakítva.



25. kép warehouse adatbázis szerkezete

Az adatbázis nem bonyolult, mindössze 4 tábla alkotja. Itt a cél egy olyan alap kialakítása volt, amely a legalapvetőbb funkciókat tartalmazza, miszerint: van valaminak tudni szeretnénk a hollétét, mennyiségét, azt hogy mikor került oda mekkora mennyiségben, és ki rakta oda. Képesnek kell lennie kezelni azt is, hogy ha valamit átrakunk valahova máshova. Valamint ha szeretnénk tudni, hogy hol miből mennyi van éppenséggel.

A “valamit” a *material* tábla reprezentálja, amelyben definiálhatunk anyagi jellemzőket a tábla mezőinek keretein belül. A tábla mezőinek nevei magukért beszélnek, talán a *Unit* mezőt érdemes elmagyarázni. Ez az adott anyag mértékegysége, pl: darab, m³, kg, lehet.

MaterialID	MaterialNumber	MaterialDescription	Unit	price/unit
1	1001	Anyag 1	m	100
2	1002	Anyag 2	liter	150
3	1003	Anyag 3	db	80
4	1004	Anyag 4	db	60

26. kép material tábla példa tartalom

A “valahová”-t a *location* tábla reprezentálja, ahol szintén magunk tudunk különféle “tárhelyeket” definiálni. A ki- és bevételezés egy-egy tárhelyként van definiálva.

LocationID	LocationName	LocationDescription	LocationCapacity
1	Bevételezés	Anyag készletre vétele	0
2	Kivezetés	Anyag kivezetése a rendszerből	0
3	Tárhely 1	Tárhely 1 leírás	100
4	Tárhely 2	Tárhely 2 leírás	100

27. kép location tábla példa tartalom

A tárolás folyamatát a *transaction* tábla reprezentálja. Itt kerülnek tárolásra az anyagmozgatás részletei.

TransactionID	TransactionDateTime	MaterialID	TransactionFromID	TransactedQty	TransactionToID	UserID
1	2025-04-06 10:43:10	1	1	20	3	06de4a98-4eb5-42f7-b3b1-8275ec220b88

28. kép transaction tábla példa tartalom

Az aktuális készletet pedig az *inventory* tábla tartalmazza. Miből éppen mennyi van. Ennek a táblának a módosításáért egy trigger felel, amely a transaction adatoknak megfelelően módosítja a készletet. (módosít meglévő rekordot , hozzáad rekordot ha nincs még olyan anyag-lokáció páros a táblázatba amit módosítani lehetne)

InventoryID	MaterialID	LocationID	Quantity
1	1	3	20

29. kép inventory tábla példa tartalom

6. Irodalomjegyzék

[Mi az a MySQL, és hogyan használják általánosan a webfejlesztésben? - EITCA Akadémia](#)

[Tutoriál: Bevezetés a Reactbe – React](#)

[\[Csapatmunka, Git-el\] 1. rész: Mi is az a Git? - INTO](#)

[Git alapok - Kezdőknek és haladóknak](#)

[Learn - Git - 1. Elmélet](#)

[Router - Korom Richárd oktatás](#)

[vanta - npm](#)

[What is Visual Studio Code?](#)

[Discord: a szórakozás és együttműködés platformja - Modern Háziasszony](#)

[Get Started with JSON Web Tokens](#)

7. Ábra jegyzék

1. kép XAMPP indítókép	5
2. kép phpMyadmin fő felülete	6
3. kép Visual Studio felülete	7
4. kép Trello felülete.....	9
5. kép applikáció kezdőoldal	10
6. kép bejelentkező oldal.....	11
7. kép kijelentkezés	12
8. kép regisztrációs oldal	13
9. kép elérhető lokáció listája.....	14
10. kép lokáció hozzáadása felület.....	15
11. kép elérhető anyagok listája.....	15
12. kép új anyag hozzáadása felület.....	16
13. kép tranzakciók listája.....	16
14. kép új tranzakció hozzáadása felület.....	17
15. kép készlet lista	18
16. kép projekt struktúra.....	19
17. kép osztály szerkezete	19
18. kép végpont szerkezete	20
19. kép swagger felület	21
20. kép swagger hozzáadása projekthez	21
21. kép létrehozott warehouse végpontok	22
22. kép létrehozott autentikációs végpontok	22
23. kép felhasznált csomagok	23
24. kép autorizációs adatbázis szerkezete	23

25. kép warehouse adatbázis szerkezete	24
26. kép material tábla példa tartalom.....	24
27. kép location tábla példa tartalom	25
28. kép transaction tábla példa tartalom	25
29. kép inventory tábla példa tartalom.....	25

8. Mellékletek

8.1. Github link

<https://github.com/StorageDevs/WarehouseApp.git>

8.2. Trello link

<https://trello.com/invite/b/67f0d529bd7de4dd75fc7455/ATTIb02932fc97548c041ca41590c784912905EBE0/warehouse-app-warehouse-dev>