** **

**ZÁRÓDOLGOZAT**

Készítették:

Kovács Dániel – Révész Lajos Bálint – Bitó Martin

Konzulens:

Németh Bence

Miskolc

2023.

Miskolci SZC Kandó Kálmán Informatikai Technikum

Miskolci Szakképzési Centrum

SZOFTVERFEJLESZTŐ- ÉS TESZTELŐ SZAK

ZÁRÓDOLGOZAT

Wherehouse

Raktárbérlő oldal

Kovács Dániel – Révész Lajos Bálint – Bitó Martin

2022-2023

**Tartalomjegyzék**

[**Bevezetés** 4](#_Toc132965938)

[**A Wherehouse-ról** 5](#_Toc132965939)

[**Fejlesztői környezet** 6](#_Toc132965940)

[XAMPP 6](#_Toc132965941)

[MySQL 6](#_Toc132965942)

[Visual Studio 7](#_Toc132965943)

[C# programozási nyelv 7](#_Toc132965944)

[.NET keretrendszer 8](#_Toc132965945)

[ASP.NET Core Web API 8](#_Toc132965946)

[Objektumorientált programozás 8](#_Toc132965947)

[Swagger 10](#_Toc132965948)

[Visual Studio Code 11](#_Toc132965949)

[React 11](#_Toc132965950)

[**A Wherehouse felépítése** 12](#_Toc132965951)

[Adatbázis 12](#_Toc132965952)

[Backend 13](#_Toc132965953)

[Frontend 15](#_Toc132965954)

[Főoldal 15](#_Toc132965955)

[Termékek 16](#_Toc132965956)

[RaktarSingle 17](#_Toc132965957)

[Rólunk 18](#_Toc132965958)

[Elérhetőség 19](#_Toc132965959)

[Felhasználói felület 20](#_Toc132965960)

[Raktáraim 22](#_Toc132965961)

[Admin 23](#_Toc132965962)

[**Tesztelés** 25](#_Toc132965963)

[**Forrásjegyzék** 25](#_Toc132965964)

# **Bevezetés**

Magyarország az elmúlt években egyre inkább a nemzetközi kereskedelem és logisztika központjává vált, és ennek köszönhetően a raktárak szerepe és fontossága is egyaránt növekszik. A raktárak nem csak az áruk tárolására szolgálnak, hanem a személyes tárgyak biztonságos megőrzésére is. Magyarországon a raktározás fejlesztése folyamatosan zajlik, hisz még nem teljesen kifejlett, mint más európai országokban. A raktár online lefoglalása lehetővé teszi a raktárak hatékonyabb működését és csökkentik a logisztikai késéseket, személyes gondokat. Emellett a raktárak fontossága Magyarországon azért is növekszik, mert az ország egyre inkább az e-kereskedelem központjává válik a régióban. Az online vásárlások növekedése miatt a raktári folyamatok hatékony kezelése és az időben történő kiszállítás kulcsfontosságú a vásárlói elégedettség és a vállalkozások sikeressége szempontjából. Összefoglalva, Magyarországon a raktárak fontos szerepet játszanak a vállalkozások sikerében és az egyéni tárgyak elhelyezésében, mivel az áruk mozgásának és tárolásának egyszerű kezelése lehetővé teszi a vállalkozások számára a hatékonyság és a termelékenység növelését, a költségek csökkentését, végül a vásárlói elégedettség javítását. Mindezen problémák elkerülése és a magyarországi raktárrendszer előrelépésének érdekében hoztuk létre a Wherehouse című projektet.

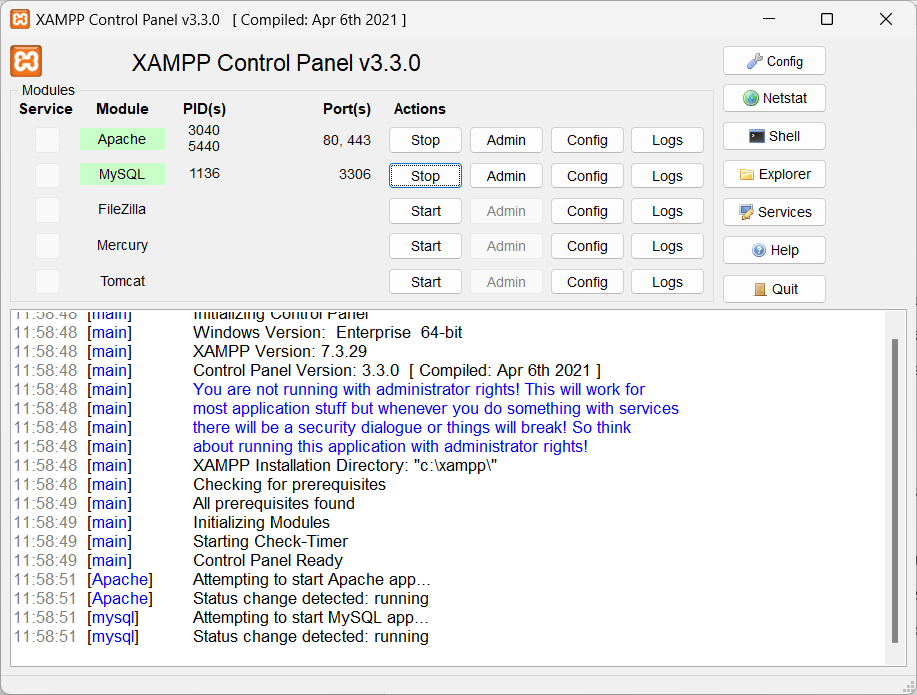
# **A Wherehouse-ról**

A Wherehouse egy raktár kiadó és -bérlő cég, amely lehetőséget biztosít az ügyfeleknek, hogy kényelmesen és gyorsan bérelhessenek és adjanak ki raktárakat Magyarországon belül, akár otthonról is. Cégünk neve az angol warehouse (magyarul raktár) szóból ered, melyet átalakítottunk Wherehouse-ra, így az ügyfelek az összefoglaló és leíró névnek köszönhetően megtudhatják, hogy oldalunk mivel foglalkozik. Logója egy raktár és egy kulcs, amely a raktár sikeres kibérlését reprezentálja. Az oldalon több raktár áll rendelkezésre, amelyek különböző méretűek és típusúak, így biztosan megtalálható az ideális raktár az adott igényekhez. Könnyen kezelhető felülettel rendelkezik, így az ügyfelek gond nélkül rátalálhatnak a kívánt raktárra. Az ügyfeleknek lehetősége van azonnal foglalni a raktár megtekintése után, így hatékony módon intézhetik a bérlési folyamatot, a felhasználói felület segítségével pedig bármikor lemondható a már nem kívánt vagy hibásan kibérelt raktár. A vevőknek lehetőségük van a nálunk dolgozó munkatársakkal felvenni a kapcsolatot, így a nap nagy részében segítséget tudunk nyújtani bármilyen problémával kapcsolatban.

1. ábra - A Wherehouse logója

# **Fejlesztői környezet**

## XAMPP

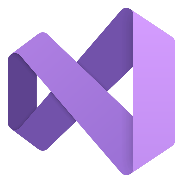
Az XAMPP egy ingyenes, nyílt forrású szoftver, amelyet a webszerverek és adatbázisok helyi telepítéséhez használnak. Tartalmazza az Apache webszervert, a PHP programozási nyelvet, az OpenSSL titkosítási szoftvert, valamint az adatbázis-kezelő rendszerként szolgáló MySQL-t. 

2. ábra - A XAMPP Control Panel felülete

### MySQL

A MySQL egy relációs adatbázis-kezelő rendszer, amelyet széles körben használnak webes alkalmazások és weboldalak tárolására és kezelésére. Az adatbázisok MySQL-ben történő létrehozása és kezelése nagyon egyszerű és rugalmas, és lehetővé teszi a felhasználók számára, hogy könnyedén tárolják és lekérdezzék az adatokat. Az XAMPP használata mellett a MySQL telepítése és konfigurálása segíti, hogy a fejlesztők a helyi számítógépükön fejlesszenek és teszteljenek webes alkalmazásokat, mielőtt azokat a valódi szerverekre telepítenék.

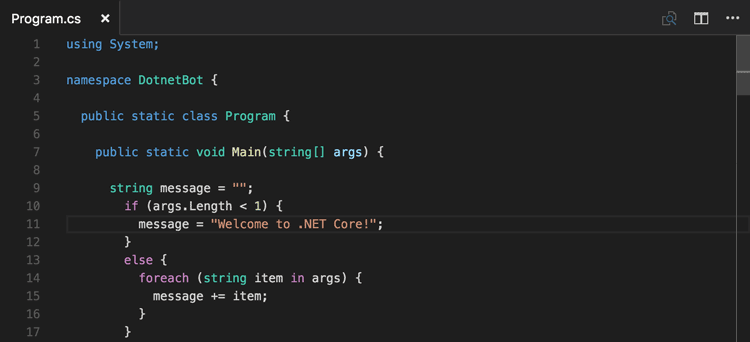
## Visual Studio

A Visual Studio egy Microsoft által fejlesztett integrált fejlesztői környezet (IDE), amelyet a szoftverfejlesztők használnak alkalmazások és weboldalak fejlesztésére. Az IDE tartalmazza az értelmezőt, a fordítót, a hibakereső eszközöket, valamint a szükséges szerkesztői funkciókat.

3. ábra - A Visual Studio logója

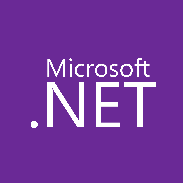
### C# programozási nyelv

A C# (C-sharp) egy modern, objektumorientált programozási nyelv, amelyet a Microsoft fejlesztett ki az általános célú szoftverfejlesztésre. A C# a Java nyelvhez hasonlóan nagyon jól skálázható és nagy teljesítményű, és nagy népszerűségnek örvend a Windows alkalmazások, az asztali és webes alkalmazások, valamint a videojátékok fejlesztése során. A Visual Studio lehetővé teszi a C# programozási nyelv alkalmazásainak fejlesztését, tesztelését és hibakeresését. Az IDE számos szolgáltatást kínál a C# programozók számára, például kódkiegészítést, szintaxiskiemelést, verziókezelést és hibakeresési eszközöket. A C# és a Visual Studio együtt nagyon hatékony eszközök a szoftverfejlesztéshez, amelyek segítségével a programozók könnyen és hatékonyan fejleszthetnek alkalmazásokat és weboldalakat.



4. ábra - Példa a C# programozásra

### .NET keretrendszer

A .NET (dot net) egy Microsoft által fejlesztett szoftverkeretrendszer, amely lehetővé teszi a szoftverfejlesztőknek a Windows-ra és más operációs rendszerekre szánt alkalmazások, weboldalak és szolgáltatások fejlesztését és futtatását. A keretrendszer több programozási nyelvet támogat, például a C#-t, a Visual Basicet és a F#-t.

5. ábra - A .NET keretrendszer logója

A .NET keretrendszer számos előnnyel rendelkezik a szoftverfejlesztés során, többek között:

* Nagy teljesítmény: A .NET futtatási környezetben történő végrehajtása nagyon gyors és hatékony.
* Biztonság: A .NET beépített biztonsági funkciókat tartalmaz, amelyek segítenek a szoftverek védelmében a kártékony támadásoktól.
* Egyszerűség: A .NET keretrendszer lehetővé teszi az alkalmazások könnyű telepítését és frissítését.
* Támogatás: A .NET széles körben elterjedt, és nagy közösség áll rendelkezésre a segítségnyújtáshoz és a fejlesztői forrásokhoz.

A .NET keretrendszer számos hasznos funkciót kínál, például a dinamikus memóriakezelést, az adatbázis-kezelést, az XML-feldolgozást és az aszinkron programozást.

### ASP.NET Core Web API

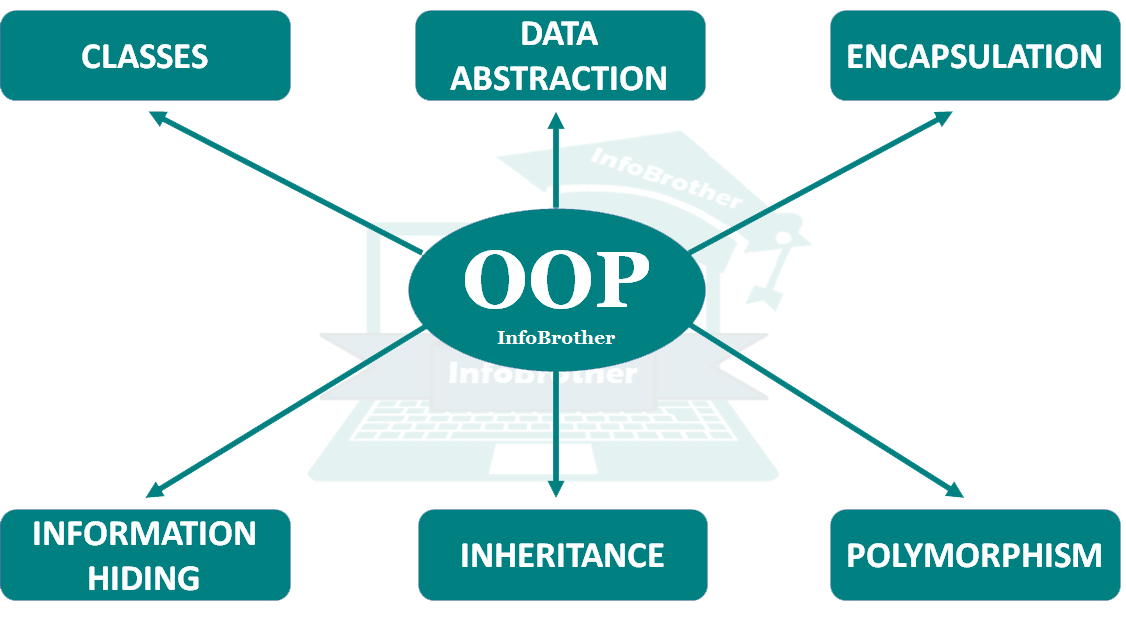
Az ASP.NET Core Web API megengedi a fejlesztőknek, hogy RESTful API-kat fejlesszenek és üzemeltessenek a .NET Core platformon. Az API-k kialakítása során az architektúrális mintákat, például a model nézet vezérlőt (MVC) használják, és az API-khoz általában JSON adatformátumot használnak.

6. ábra - Az ASP.NET alapsémája

### Objektumorientált programozás

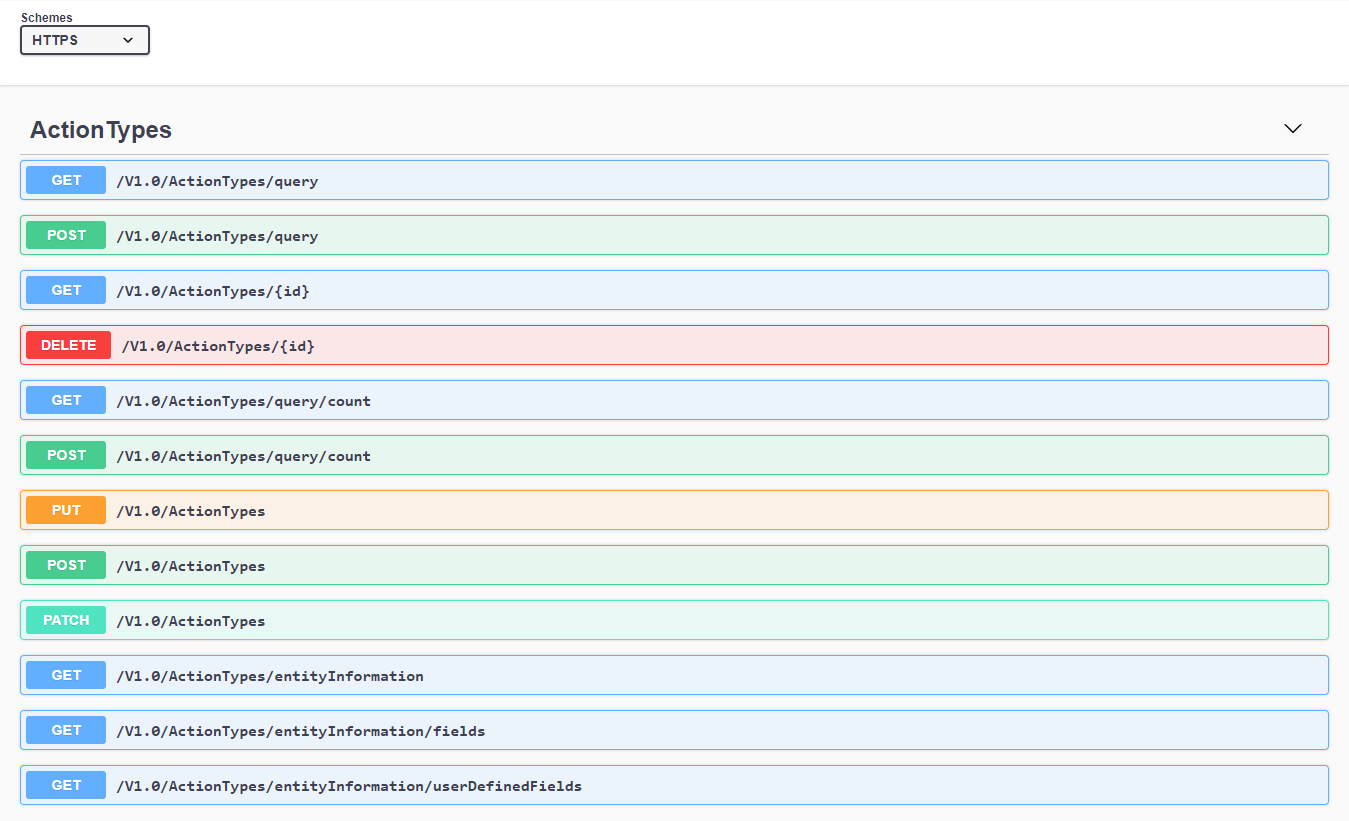
Az objektumorientált programozás (OOP) egy paradigmája vagy módszertana a programozásnak, amelyben az adatokat és az azokon végrehajtandó műveleteket egy egységes entitásba, az ún. objektumba szervezzük. Ezen módszertan, az adatokat és az azokon végrehajtandó műveleteket összekapcsolja, mely jóváhagyja az alkalmazások moduláris felépítését és az újrahasználhatóság növelését. Vannak alapvető fogalmai, mint például az osztály, az objektum, az öröklődés és a polimorfizmus.

* Az **osztály** az objektumok sablonja, amely definiálja az objektumok (adattagok) tulajdonságait és viselkedését (metódusok).
* Az **objektum** egy példánya az osztálynak, amely rendelkezik az osztály tulajdonságaival és viselkedésével.
* Az **öröklődés** lehetővé teszi, hogy egy osztály tulajdonságait és műveleteit átörökítse egy másik osztálynak, de azokat még továbbfejlesztheti, kiegészítheti. Az osztályok közötti öröklődés biztosítja az újrafelhasználható kód létrehozását és az alkalmazások hierarchikus felépítését.
* A **polimorfizmus** engedélyezi, hogy egy adott műveletet többféleképpen lehessen megvalósítani, tehát az objektumok különböző módokon viselkedhetnek ugyanazon művelet meghívásakor, amely dinamikus programozást és az absztrakt típusok használatát eredményezi.

Számos programozási nyelv támogatja, többek között a Java, a C++, a Python és a C#. Elterjedtsége részben annak köszönhető, hogy az alkalmazásokat könnyebb karbantartani és bővíteni.

7. ábra - Az OOP ábrája

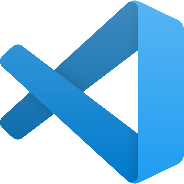
### Swagger

A Swagger egy nyílt forráskódú eszközkészlet, melyen keresztül a fejlesztők a RESTful API-kat egységesen, könnyen olvasható formában dokumentálhatják és tesztelhetik. Ezen JSON vagy YAML formátumban írt dokumentum, megjeleníti az API-k végpontjait, a végpontok által elfogadott paramétereket és az API-k által visszaadott válaszokat, hibakódokat. Mivel többféle programozási nyelveket és keretrendszereket támogat, így használatával a programozók gyorsabban és hatékonyabban dolgozhatnak, és csökkenthetik a hibák számát az alkalmazások fejlesztése során.

8. ábra - A Swagger logója

9. ábra - Az API-k megjelenítése Swaggerben

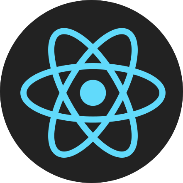
## Visual Studio Code

A Visual Studio Code egy nyílt forráskódú kódszerkesztő és integrált fejlesztői környezet (IDE). A Visual Studio Code lehetővé teszi a fejlesztők számára, hogy különböző programozási nyelvekben dolgozzanak, mint például a JavaScript, a TypeScript, a Python, az HTML, a CSS és sok más.

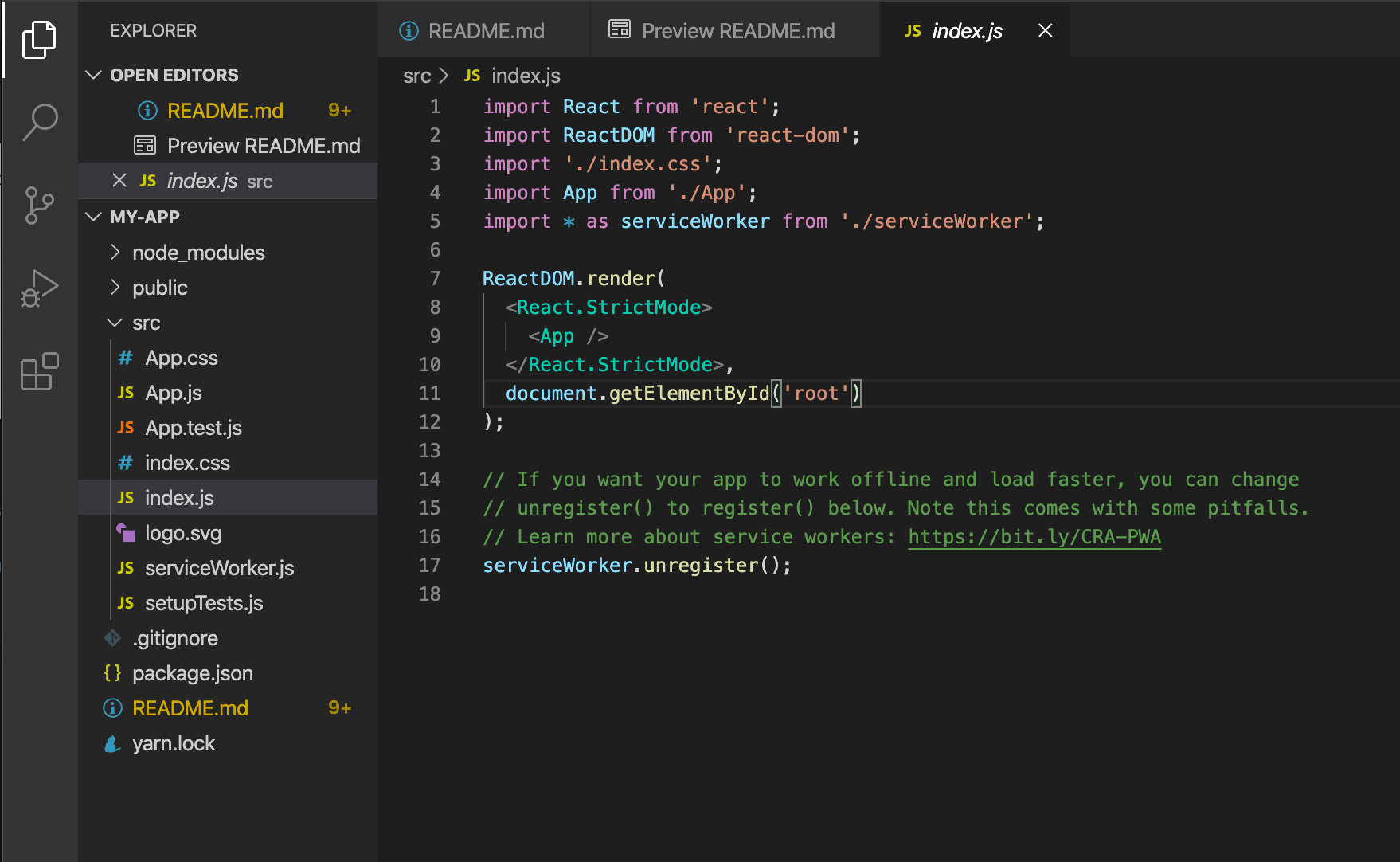
10. ábra - A Visual Studio Code logója

Az IDE számos hasznos funkciót kínál, mint például a kódszintaxis kiemelést, az automatikus kódbehelyettesítést, a kódsablonokat, a Git integrációt, a beépített terminált és az értesítéseket.

### React

A React egy nyílt forrású, komponens alapú JavaScript könyvtár, amelyet a Meta fejlesztett ki, és arra szolgál, hogy interaktív felhasználói felületeket hozzunk létre webalkalmazásokban. Virtuális DOM (Document Object Model) rendszert használ, amely hatékonyan kezeli a nagy mennyiségű adatokat, illetve jeleníti meg a dinamikus felhasználói felületeket. Manapság az egyik legnépszerűbb keretrendszer a frontend webfejlesztésben. Alapvetően arra épül, hogy elkülönítse az alkalmazás állapotát (state) és az azt megjelenítő felhasználói felületet (UI). A komponensek létrehozása egyszerűsíti az alkalmazás összetevőinek kezelését.

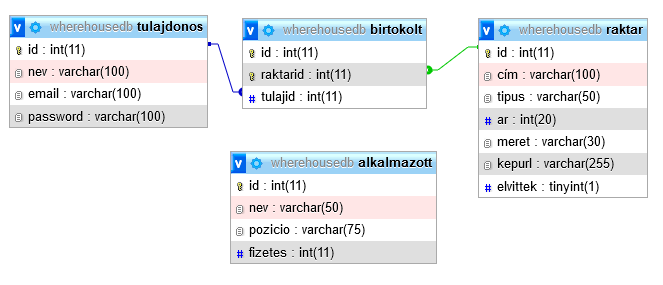
11. ábra - A React logója



12. ábra - A React környezete

# **A Wherehouse felépítése**

## Adatbázis

13. ábra - A Wherehouse adatbázisának tervező nézete

Az adatbázisban összesen négy tábla található: Alkalmazott, Birtokolt, Raktar és Tulajdonos. Mindegyik táblának a legfontosabb eleme az „id” mező, ezzel lehet megkülönböztetni az azonos rekordokat.

**Az Alkalmazott tábla három mezőt tartalmaz:**

* nev: Az alkalmazott neve
* pozicio: Az alkalmazott beosztása
* fizetes: Az alkalmazott fizetése (ez egy privát adat, csak az admin számára elérhető)

A **Birtokolt tábla** a Raktár és Tulajdonos táblákkal kapcsolódik össze foreign key (idegen kulcs) kapcsolattal. Ez a tábla **két mezőt tartalmaz:**

* raktarid: A lefoglalt raktár id-je
* tulajid: A lefoglalt raktár tulajdonosának az id-je

**A Raktar tábla hat mezőt tartalmaz:**

* cím: A raktár pontos helyszíne
* tipus: A raktár típusa
* ar: A raktár bérlésének díja
* meret: A raktár mérete négyzetméterben mérve
* kepurl: A raktárhoz tartozó kép URL-je

Végül **a Tulajdonos tábla három mezőt tartalmaz:**

* nev: A regisztrált tulajdonos felhasználóneve
* email: A tulajdonos e-mail címe
* password: A tulajdonos jelszava

## Backend

A képen szöveg látható

Automatikusan generált leírásBackendünk felelős az adatok feldolgozásáért, valamint az adatbázisba történő írásért és olvasásért. Az oldal az adatbázis segítségével a kliensek által kért adatokat visszaküldi. A programunk backendjében található mappák és fájlok az alkalmazás különböző részeire vonatkoznak. Adatbázisunkat a NuGet Package Manager Console-ba illesztett két paranccsal tudjuk létrehozni:

* Add-Migration initial
* Update-Database

Azonban, ha a **Migrations** mappa nem üres, akkor a két legenerált osztályt kitörölve a fentebb említett kódokkal tudjuk újból létrehozni.

14. ábra - Backendünk felépítése

A **Controllers** mappa tartalmazza az egyes adattáblákhoz kapcsolódó vezérlő osztályokat, amelyek a kliensek által küldött kéréseket fogadják, és visszaadják a megfelelő választ. A controllers tartalma:

* AlkalmazottController.cs
* BirtokoltController.cs
* LoginController.cs
* RaktarController.cs
* TulajdonosController.cs

A **Models** mappában megtalálható az adatbázisunkban szereplő adattáblákat leíró modellek osztályai, amelyek a következőek:

* Alkalmazott.cs
* Birtokolt.cs
* Raktar.cs
* Tulajdonos.cs

A **Repository** mappa tartalmazza az adatbázis lekérdezéseket és módosításokat végző "WherehouseRep.cs" osztályt.

A **Service** mappa tartalmazza a "WherehouseService.cs" osztályt, amely az összetettebb lekérdezéseket dolgozza fel. Ezen osztály függőségeként használja a **Repository** mappában található osztályokat.

A képen asztal látható

Automatikusan generált leírásA "WherehousedbContext.cs" fájlban van az Entity Framework DbContext osztálya, amely az adatbázis kapcsolatot kezeli.

15. ábra - A Wherehouse CRUD (Create, Read, Update, Delete) műveletei

## Frontend

Projektünkben minden oldalon megtalálható egy navigációs sáv (avagy navbar), amely segítségével könnyedén tudunk navigálni az oldalakon keresztül. A navbar a következőket foglalja magába:

* Főoldal
* Termékek
* Rólunk
* Elérhetőség
* Felhasználói felület

A képen Webhely látható

Automatikusan generált leírásEzen felül a lapok alján egy állandó footer (lábléc) rész látható, amely a kapcsolatunkat és a közösségi felületünket, valamint egy „Copyright”-ot és egy „a lap tetejére” gombot fog egybe. Az oldal főbb színei a szürke, a sárga és a fehér.

### Főoldal

A **Főoldal** egy fejléccel üdvözli a látogatókat, amely egy nagy címet és egy bevezető szöveget tartalmaz a Wherehouse logójának kíséretével. Továbbá megjelenik a weboldal szolgáltatásai és a kiemelt ajánlatai. A raktárak adatait az API-ról szedi le a **fetch** segítségével és az adatokat egy szöveges objektumból JSON objektummá alakítja át. Az adatok betöltése a **useEffect** hook használatával történik, és amikor az adatok betöltődnek, akkor a **setRaktar** függvény segítségével beállítja a raktárakat. A **useState** hook használatával a komponensben tároljuk az adatokat és a betöltés állapotát is (**isFetchPending**). A komponens további részei közé tartozik a szolgáltatások és a kiemelt ajánlatok bemutatása képekkel és szövegekkel.

16. ábra - Főoldal

### Termékek

A **Termékek** fül kódja az adatbázisban lévő termékek megjelenítését végzi, a React Hook **useState**-je által, amelynek segítségével állapotokat tud tárolni, valamint a **useEffect**-el, amely hasznos lehet adatok lekérésekor és a raktár állapotának megfigyelésére. Az adatokat egy tömbben tárolja, majd ezt a tömböt a termékek megjelenítéséhez használja. A termékek megjelenítéséhez Bootstrap keretrendszert is használunk, például a **container**, **row**, és a **col** stílusosztályokat egyedi tulajdonságokkal. A termékek alapvetőleg egy-egy stílusos, reszponzív div-ben találhatóak, melyhez az egér ráhúzásakor egy zöld, lekerekített border (szegély) társul egy minimális transition-nel (átmenettel). Ezen div-en belül a raktár típusa, címe, képe található. A kód használ egy **Link** komponenst is az „Információ” gombon, amely a hozzátartozó raktár oldalára irányít át a **raktar** objektum által, viszont ez egyben jelzi is a raktár elérhetőségét, tehát ha a raktár még nem lelt gazdára, akkor az „Információ” gomb él, ellenben egy szürke „A raktár jelenleg nem elérhető!” kikapcsolt gomb található. Ha a felhasználó adminisztrátor, akkor számára nem jelenik meg, azonban a sima felhasználók láthatják a raktárak az „Információ” gombot. A raktár elérhetőségét szintén jelzi a négyzetben lévő szín (piros, ha foglalt; zöld, ha még szabad), illetve az ehhez társuló bootstrap ikon. A képen diagram látható

Automatikusan generált leírás

17. ábra - Termékek oldal 2 foglalt, 1 szabad raktárral

### RaktarSingle

A képen Webhely látható

Automatikusan generált leírásEzen az oldalon jelenik meg egy raktár adatai az „Információ” gomb rányomását követően. A raktárhoz tartozik egy kép, cím, típus, méret és a bérleti díj. A **useParams** hook segítségével a komponens kinyeri az URL-ben lévő raktár azonosítóját, majd a **useEffect** hook segítségével aszinkron módon lekéri a raktár adatait a szerverről. Ha a kérés sikeres, akkor az adatokat a **useState** hook segítségével beállítja, és megjeleníti azokat a reszponzív „card”-ban (kártya). Ha a felhasználó be van jelentkezve (a felhasználó azonosítója megtalálható a **localStorage**-ben), akkor a "Kibérlem" gombra kattintva a komponens egy POST kérést küld a szervernek, hogy rögzítse a bérlést. Ha a bérlés rögzítése sikeres, akkor a komponens egy PUT kérést küld a raktár elvittek tulajdonságának frissítésére, majd visszairányítja a felhasználót a **Termékek** oldalra. Ha a felhasználó nincs bejelentkezve, akkor a "Kérjük lépj be, a bérléshez!" üzenet jelenik meg. A „Vissza” gombbal lehet a **Termékek** oldalra visszatérni. Fontos megjegyezni, hogy ha egy raktárat már kibéreltek, akkor erre az oldalra nem lehetséges a visszatérés, csakis akkor, ha már szabad a raktár.

18. ábra - A megrendelői felület

### Rólunk

A képen diagram látható

Automatikusan generált leírásA **Rólunk** felület megjeleníti a jelenlegi alkalmazottjainkat. A **useState** hook segítségével inicializáljuk az "alkalmazottak" állapotot üres tömbként, majd a **setAlkalmazott** segítségével állítjuk be az állapotot, amikor a lekérdezés eredménye megérkezik. A **useEffect** hook segítségével az oldal betöltésekor lefut egy egyszeri lekérdezés, ami lekéri az alkalmazottak listáját az API-ról, majd beállítja az állapotot egy reszponzív div-be. Az oldal megjelenítésekor a "alkalmazottak" állapotot map-eljük, és minden alkalmazottra egy "team-item" elemet jelenítünk meg. A kép és a szöveges tartalom az alkalmazott objektumának megfelelően változik. Egy "team-social" osztályban az alkalmazott elérhetőségeit jelenítjük meg ikonok segítségével, melyet úgy érünk el, ha ráhúzzuk a kurzort a kívánt személy kártyájára.

19. ábra - A Rólunk oldal jelenlegi alkalmazottjainkkal (középsőn "hover" esemény látható)

### Elérhetőség

Az oldalon három oszlop található, amelyek mindegyike egy-egy kapcsolatfelvételi lehetőséget képvisel. A három div a következőket tartalmazza:

* cím
* telefonszám
* e-mail

A képen térkép látható

Automatikusan generált leírásMinden lehetőség egy dobozba (contact-item) került, melyek szintén egy kis animációt örököltek, illetve a közepén egy ikon (contact-icon) található. Az ikonok a Bootstrap ikonkészletből (bi) származnak. Az oldal alján egy Google Térkép **iframe** látható, amely Miskolcot ágyazza be a weboldalba. Az iframe osztálya lehetővé teszi, hogy a térkép a böngésző szélességéhez igazodjon, és hogy mobilbarát legyen.

20. ábra - A kapcsolatfelvételi lehetőségek

### Felhasználói felület

A képen szöveg látható

Automatikusan generált leírásA projektünkben lehetőség van egy felhasználó regisztrálására, illetve bejelentkeztetésére. A minden oldalon megjelenő navigációs sáv jobb szélén lévő profil ikonra kattintva, megjelenik két opció:

* Regisztráció
* Bejelentkezés

21. ábra - A felhasználói felület

A képen Webhely látható

Automatikusan generált leírásA **regisztráció** fülre kattintva egy modális ablak jelenik meg a felhasználó regisztrációjához. A modális ablak egy űrlapot tartalmaz, amely a felhasználó nevét, e-mail címét és jelszavát kéri, illetőleg lehetőség adódik a jelszó megjelenítésére a kis „szem” ikonra kattintva. Az űrlap benyújtásakor a komponens a felhasználó adatait elküldi egy API-nak, majd a sikeres regisztráció esetén bezárja az ablakot és megerősítő üzenetet jelenít meg. Az űrlap benyújtásakor a komponens blokkolja a lap görgetését, hogy elkerülje az ablak mögötti tartalom hozzáférését. Az űrlap bármely mezőjének módosításakor a komponens az adatokat az állapotban tárolja. Az állapot módosításakor a komponens újra renderelődik, és az űrlap megjeleníti az aktuális állapotot. Az állapot változását a **useState** hookok segítségével követi nyomon, és az ablak bezárásakor az állapotot visszaállítja az eredeti értékre.

22. ábra - Regisztrációs modal

A képen szöveg látható

Automatikusan generált leírásA képen szöveg látható

Automatikusan generált leírásA képen Webhely látható

Automatikusan generált leírásA **bejelentkezés** fülre kattintva egy modális ablak jelenik meg, amelynek segítségével a felhasználók be tudnak jelentkezni a weboldalra. A **useState** hook-kal tároljuk a felhasználónév és jelszó mezők értékeit. A **handleSubmit** függvény az űrlap elküldésekor fut le, ami egy POST kérést küld el az ehhez tartozó végponthoz a megadott felhasználónévvel és jelszóval. Ha a szerver „ok” értékkel tér vissza, akkor a felhasználó adatait elmentjük a böngésző helyi tárhelyébe (**localStorage)**, és megjelenik a felhasználó neve az ikon mellett, valamint a bejelentkezés és regisztráció füleket felváltja a „Raktáraim” és a „Kijelentkezés” fülek. Az admin bejelentkezése esetén a „Raktáraim” helyett egy „Admin” fül jelenik meg. Ha a szerver hibás adatokkal tér vissza, akkor egy hibaüzenet jelenik meg a felhasználó számára.

24. ábra (bal) - Alap jogosultságú felhasználó, 25. ábra (jobb) - Admin

23. ábra - Bejelentkezés modal

### Raktáraim

A képen szöveg látható

Automatikusan generált leírásA raktáraim fülre kattintva a felhasználót átdobja a **Raktáraim** oldalra, amely a lefoglalt raktárait listázza ki. Alapvetően két **useEffect** hook-ot használ, az egyik fetch API-t hív a Birtokolt végponton, a másik pedig a Raktar és Birtokolt végpontokon történő **Promise** összesítést végzi el. A „Lemondás” gomb a birtokolt raktár eltávolításáért felelős. Amennyiben az illető nem rendelkezik raktárral, akkor a címsorban „Jelenleg nincs raktárad” üzenet jelenik meg. A képen Webhely látható

Automatikusan generált leírás

27. ábra - Raktár nélküli felhasználó esetén megjelenő szöveg

26. ábra - A proba1 felhasználó által birtokolt raktár

### Admin

Az **Admin** oldalra belépve lehetőségünk adódik:

* egy új raktár létrehozására
* egy meglévő raktár módosítására, törlésére
* egy új alkalmazott hozzáadására
* egy alkalmazott adatainak módosítására, törlésére
* tulajdonosok felhasználó fiókjainak módosítására, törlésére

Az oldal raktár része három **useEffect** hookot használ a raktárak, alkalmazottak és tulajdonosok lekérésére egy API végpontról. A komponens további része három függvényt definiál:

* **addRaktar**: új raktár hozzáadása az adatbázishoz egy POST kérés segítségével
* **deleteRaktar**: a megadott raktárt törli az adatbázisból és az összes kapcsolódó adatot a Birtokolt táblából
* **updateRaktar**: a raktár tulajdonságainak frissítését egy PUT kérés segítségével

Az oldal alkalmazott része a következő függvényeket definiálja:

* **addAlkalmazott**: új alkalmazott hozzáadása az adatbázishoz
* **deleteAlkalmazott**: egy alkalmazott törlése az adatbázisból az azonosítója alapján
* **updateAlkalmazott**: egy alkalmazott adatainak módosítása az azonosítója alapján

Az oldal tulajdonos része a következő függvényeket definiálja:

* **deleteTulajdonos**: egy tulajdonos és a tulajdonoshoz tartozó táblákból az összes kapcsolódó adat törlése az adatbázisból az azonosítója alapján,
* **updateTulajdonos**: egy tulajdonos módosítása az azonosítója alapján

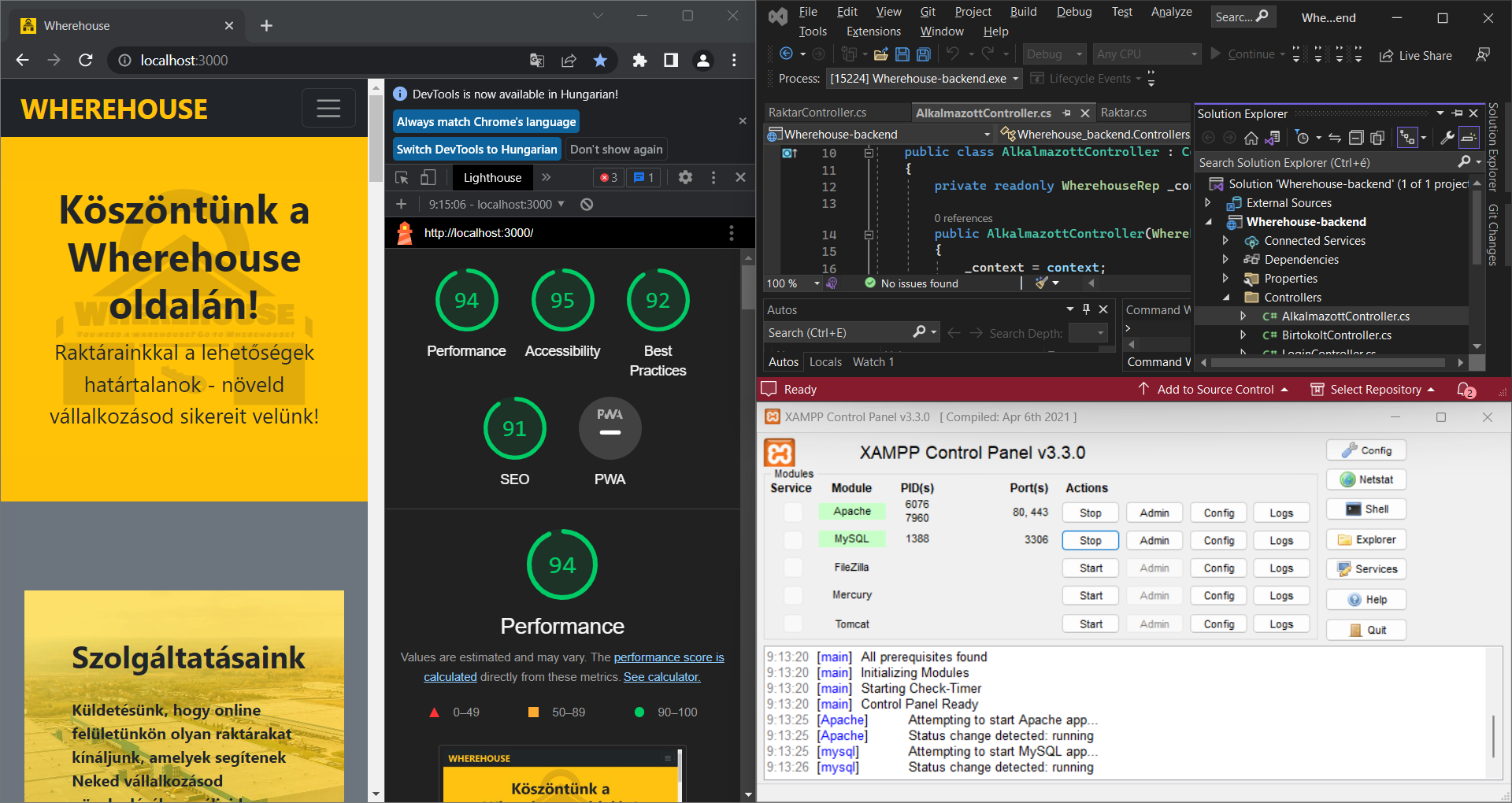
Minden függvény a megfelelő adatokat JSON formátumban küldi el az adatbázis számára, majd a visszakapott eredményeket is JSON formátumban dolgozza fel. Ha a hálózati kommunikáció valamilyen hibát jelez, akkor a konzolra kiírja a hibaüzenetet. Végül minden függvény frissíti az oldalt a **window.location.reload()** segítségével.



28. ábra - A teljes admin oldal

# **Tesztelés**

Mi a **Lighthouse** nevű Google Chrome-os bővítménnyel teszteltük le az oldalunkat a backend és a XAMPP futása közben, melynek teljesítménye az alábbi képen látható:



29. ábra – A Wherehouse tesztelése a Lighthouse bővítmény segítségével

# **Forrásjegyzék**

<https://www.inf.u-szeged.hu/~gnemeth/adatbgyak/exe/MySQL_XAMPP_JDBC/index.html>

<https://www.kersoft.hu/szoftverek/microsoft/visual-studio/>

<https://csharptutorial.hu/docs/hellovilag-hellocsharp/2-a-nyelv-alapjai/a-c-programozasi-nyelv-alapjai/>

<https://gyires.inf.unideb.hu/GyBITT/31/ch18s03.html>

<https://www.c-sharpcorner.com/article/asp-net-core-5-0-web-api/>

<https://www.techtarget.com/searchapparchitecture/definition/Swagger>

<https://csharptutorial.hu/docs/hellovilag-hellocsharp/13-net-core-es-net/visual-studio-code/>

<https://hu.reactjs.org/tutorial/tutorial.html>