**ENERGETIKAI TECHNIKUM ÉS KOLLÉGIUM**

**Vizsgaremek**

**Consumption Management System**

**2025.**

|  |  |
| --- | --- |
| **Szakma:** | **Készítették:** |
| **Szoftverfejlesztő és -tesztelő** | **Dobosi Gábor** |
| **5 0613 12 03** | **Mák Luca** |
|  | **Orbán Barnabás** |

Tartalomjegyzék

[1 Bevezetés 4](#_Toc192504093)

[2 Témaválasztás indoklása 4](#_Toc192504094)

[2.1 Indoklás és aktualitás 4](#_Toc192504095)

[2.2 Kutatások, követelmények 5](#_Toc192504096)

[2.3 Eszközök és eljárások 5](#_Toc192504097)

[2.3.1 Általános fejlesztéshez használt eszközök 5](#_Toc192504098)

[2.3.2 Frontend fejlesztéshez használt eszközök 6](#_Toc192504099)

[2.3.3 Adatbázis fejlesztéshez használt eszközök 7](#_Toc192504100)

[3 Fejlesztői dokumentáció 9](#_Toc192504101)

[3.1 Frontend 9](#_Toc192504102)

[3.1.1 A programmal kapcsolatos követelmény: 9](#_Toc192504103)

[3.1.2 Megjelenési követelmény: 9](#_Toc192504104)

[3.2 Backend 11](#_Toc192504105)

[3.2.1 Specifikáció 11](#_Toc192504106)

[3.2.2 Az alkalmazott fejlesztői eszközök 11](#_Toc192504107)

[3.2.3 Adatmodell leírása 14](#_Toc192504108)

[3.2.4 Táblák és modelljeik: 15](#_Toc192504109)

[3.2.5 Kontextus osztály 22](#_Toc192504110)

[3.2.6 Dependency Injection (DI) 22](#_Toc192504111)

[3.2.7 OnModelCreating Metódus és Entitás Kapcsolatok Konfigurálása 23](#_Toc192504112)

[3.2.8 DTO-k (Data Transfer Object) 23](#_Toc192504113)

[3.2.9 Controllerek 24](#_Toc192504114)

Alulírott **Dobosi Gábor** büntetőjogi felelősségem teljes tudatában nyilatkozom arról, hogy az itt szereplő vizsgaremek csoportmunka eredménye és sem részeiben sem egészében nem került még kereskedelmi forgalomba, ill. publikálásra, a GPL licenszelésű programrészek kivételével.

Paks, 2025. április 18.

**Dobosi Gábor**

Alulírott **Mák Luca** büntetőjogi felelősségem teljes tudatában nyilatkozom arról, hogy az itt szereplő vizsgaremek csoportmunka eredménye és sem részeiben sem egészében nem került még kereskedelmi forgalomba, ill. publikálásra, a GPL licenszelésű programrészek kivételével.

Paks, 2025. április 18.

**Mák Luca**

Alulírott **Orbán Barnabás** büntetőjogi felelősségem teljes tudatában nyilatkozom arról, hogy az itt szereplő vizsgaremek csoportmunka eredménye és sem részeiben sem egészében nem került még kereskedelmi forgalomba, ill. publikálásra, a GPL licenszelésű programrészek kivételével.

Paks, 2025. április 18.

**Orbán Barnabás**

# Bevezetés

Csapatunk egy valós problémára keresett megoldást, amely egy olyan szoftver kifejlesztéséhez vezetett, mely szállodákban, fürdőkben és egyéb komplex szolgáltatóhelyeken alkalmazható. A program célja az volt, hogy megkönnyítse az intézmény dolgozóinak munkáját, miközben a vendégek számára kényelmesebb élményt biztosít. A rendszer egy helyre gyűjti a felhasználók költéseit, melyet egy ID vagy vonalkód segítségével tudnak beazonosítani az alkalmazottak.

Példának okáért vegyünk egy fürdőt amire a programunk is alapul, de kis változtatással bármely más szolgáltatást nyújtó területre könnyen átalakítható. A vendégek belépéskor kapott karszalagjukhoz kapcsolódva igénybe vehetik a különböző szolgáltatásokat anélkül, hogy azonnali fizetésre lenne szükség. Úgy oldható meg hogyha a karszalagjukra vannak terhelve az igénybe vett szolgáltatások.

Ez több szempontból is előnyös. Először is nem kell egész nap magukkal hordani a pénztárcájukat vagy bankkártyájukat, ami kényelmi és biztonsági szempontból is praktikus. Emellett nem kell minden igénybe vett szolgáltatásnál fizetni külön-külön, hanem egyszerre kényelmesen eltudják intézni a látogatásuk végén. Fizikai eszközök is rendelkezésünkre álltak így felhasználtuk azokat is a munkához, ezzel is szimulálva az éles környezetet.

# Témaválasztás indoklása

## Indoklás és aktualitás

A témaválasztás nem volt egyszerű. Szerettünk volna egy olyan alkalmazást létrehozni, ami kihívások elé állít nem csak a minimumot teljesíteni. Számos ötlet is felmerült, például webshop, Rubik kocka időmérő. A választásunk végül egy szolgáltatás kezelő rendszerre esett. Éles környezet minél jobb reprezentációja érdekében fizikai eszközöket is bevontunk a folyamatba, így került sor a karszalagok, az RFID olvasó és a vonalkódolvasó alkalmazására. Figyelembe véve azt, hogy intézményünk területén jelenleg is alkalmaznak hasonló eszközöket a beléptetéshez, a kezdeti tesztek ezeknek a segítségével mentek végbe.

Aktualitás szempontjából is megállja a helyét, mivel mindenki egyre jobban törekszik a gyorsaságra, kényelemre, egyszerűségre. A programunk pontosan ezt kívánja megvalósítani.

A fejlesztés során a legkorszerűbb technológiákra alapoztunk, így a projekt .NET 8-ra és a React keretrendszerre épül. Ezáltal nem csak a legújabb fejlesztéseket használtuk ki, de a dinamikus, API-központú webfejlesztés irányelveit is követtük, amely a jövőbeli bővíthetőség szempontjából is kiemelten fontos.

## Kutatások, követelmények

Első lépésként áttekintettük a vizsgakövetelményeket annak érdekében, hogy azonosítsuk a kötelező elemeket. Az alkalmazásnak vagy weboldalnak tartalmaznia kellett egy frontend részt, amely a megjelenítésért és az adatok betöltéséért felelős. Emellett szükség volt egy backend rendszerre is, amely az adatok tárolását, a jogosultságkezelést, az adatbázis-táblák és kapcsolatok pontos definiálását, valamint az API végpontok meghatározását biztosítja, lehetővé téve a frontend és a backend közötti kommunikációt.

A hatékonyság érdekében frontend oldalról React keretrendszert alkalmaztunk. A backend fejlesztéséhez az ASP.NET Core Web API technológiát választottuk, amely a C# programozási nyelven alapul.

A szoftver adatbázis MSSQL alapú, amely az Entity Framework segítségével könnyen kezelhető. A programunk megfelel az alapvető adatkezelési műveleteknek (CRUD), amely az adatok létrehozását (Create), lekérdezését (Read), módosítását (Update) és törlését (Delete) foglalja magában. Ezek a backend oldalon GET, POST, PUT és DELETE HTTP-metódusokként valósultak meg.

Igyekeztünk a tiszta kód elveit alkalmazni, amelyek elősegítik a fejlesztés hatékonyságát, az átláthatóságot, valamint a későbbi módosítások egyszerűbb elvégzését. Ennek érdekében például az egyértelmű változóneveket a PascalCase elnevezési konvenció szerint definiáltuk, és logikus, jól strukturált mappaszerkezetet alakítottunk ki.

## Eszközök és eljárások

### Általános fejlesztéshez használt eszközök

#### Trello

A Trello egy vizuális projektmenedzsment eszköz, amely Kanban-alapú táblák, listák és kártyák segítségével segíti a feladatok szervezését. Fejlesztőcsapatok számára kiválóan alkalmas sprint tervezésre, backlog kezelésre és feladatkövetésre. Integrálható GitHubbal, Slackkel és más fejlesztői eszközökkel.

#### GitHub

A GitHub egy népszerű forráskód-kezelő és verziókövető platform, amely a Git rendszeren alapul. Lehetővé teszi a fejlesztők számára a kód tárolását, verziózását, csapatmunkát és CI/CD (folyamatos integráció és telepítés) folyamatok kezelését. Nyílt forráskódú és privát projektekhez egyaránt használható.

### Frontend fejlesztéshez használt eszközök

#### Visual Studio Code:

A **VSCode** egy ingyenes, könnyű, mégis erőteljes kódszerkesztő, amelyet a Microsoft fejlesztett. Beépített támogatást nyújt JavaScript, TypeScript és React fejlesztéshez, valamint számos kiegészítő (pl. ESLint, Prettier) elérhető hozzá a fejlesztési élmény javítása érdekében.

#### Vite

A Vite egy modern fejlesztési szerver és build eszköz, amely gyorsabb fejlesztői élményt nyújt a React és TypeScript alapú alkalmazások számára. Használata lehetővé teszi a gyorsabb hot-reloadot és optimalizált végső kódot.

#### Material-UI (MUI)

A Material-UI egy népszerű React UI komponenskönyvtár, amely a Google Material Design elveit követi. Segítségével modern, reszponzív és könnyen testreszabható felhasználói felületeket lehet fejleszteni.

#### Axios

Az Axios egy ígéreteken (Promises) alapuló HTTP kliens, amelyet adatok lekérésére és küldésére használnak API-khoz. Különösen népszerű a React alkalmazásokban, mivel egyszerűsíti az aszinkron hálózati kéréseket és kezelést.

#### React & React-DOM

A React egy népszerű JavaScript könyvtár interaktív felhasználói felületek fejlesztésére. A React-DOM a React komponensek DOM-ba való rendereléséért felelős, biztosítva a hatékony frissítéseket és a komponens-alapú fejlesztési modellt.

#### TypeScript

A TypeScript egy JavaScript alapú programozási nyelv, amely statikus típusellenőrzést biztosít. A React alkalmazásokban használva segít a hibák csökkentésében és a kód karbantarthatóságának javításában.

#### ESLint

Az ESLint egy JavaScript és TypeScript kódellenőrző eszköz, amely segít fenntartani a kódminőséget, betartani a kódolási szabványokat és azonosítani a lehetséges hibákat már fejlesztés közben.

#### Prettier

A Prettier egy automatikus kódformázó eszköz JavaScript, TypeScript, HTML, CSS és sok más nyelv számára. Segítségével egységes kódstílust tarthatunk fenn a projektben, anélkül hogy manuálisan kellene figyelni a formázásra.

#### Tailwind CSS

A Tailwind CSS egy utility-first megközelítésű CSS keretrendszer, amely előre definiált osztályokat biztosít a gyors és hatékony stílusozáshoz. Ahelyett, hogy kész komponenseket adna (mint pl. Bootstrap vagy Material-UI), kis, önálló osztályokat kínál, amelyekkel teljes mértékben testreszabható a dizájn.

#### Whimsical

A Whimsical egy vizuális tervezőeszköz, amely támogatja diagramok, drótvázak, folyamatábrák és gondolattérképek készítését. Ideális UX/UI tervezéshez, rendszerarchitektúrák megjelenítéséhez és csapatmunkában történő ötleteléshez.

### Adatbázis fejlesztéshez használt eszközök

#### SQLEXPRESS

A Microsoft SQL Server ingyenes, könnyített verziója, amely fejlesztési célokra ideális. Jól integrálható ASP.NET Web API alkalmazásokkal az Entity Framework Core használatával.

#### SQL Server Management Studio

Grafikus felhasználói felületet biztosító eszköz a Microsoft SQL Server kezelésére. Lehetővé teszi az adatbázisok kezelését, lekérdezések futtatását, tárolt eljárások létrehozását.

#### Dbeaver

Ingyenes, többplatformos adatbázis-kezelő eszköz, amely támogatja a SQL Server, MySQL, PostgreSQL, SQLite és más adatbázisokat, vizuális szerkesztővel és lekérdezési funkciókkal.

#### drawDB

A drawDB egy adatbázis-tervező eszköz, amely lehetővé teszi az ER-diagramok (Entity-Relationship) és adatbázisstruktúrák vizuális modellezését. Segítségével könnyedén megtervezhető egy rendszer adatmodellje, amely később SQL-kóddá alakítható.

# Fejlesztői dokumentáció

## Frontend

### A programmal kapcsolatos követelmény:

A program egy fürdő dolgozói számára lett kifejlesztve, figyelembe véve a felhasználói kör sajátosságait és igényeit. A fejlesztés során az összes lehetséges hibát és problémát feltérképeztük, amikor a felhasználó figyelmetlenségből vagy hibás döntés következtében súlyos problémákat okozhat. Ezek közé tartozhatnak kisebb működési zűrzavartól kezdve egészen a rendszer működését teljesen leállító vagy visszafordíthatatlan hibákig terjedő következmények. A kockázatok minimalizálása érdekében a programot alapos tesztelésnek vetettük alá, többféle tesztelési módszert alkalmazva annak érdekében, hogy a rendszer minden körülmény között megbízhatóan működjön.

### Megjelenési követelmény:

* Bejelentkezési felület:

A rendszer tartalmaz működő beviteli mezőket, ahol a felhasználónevet és a jelszót lehet megadni. Ezen kívül egy "Küldés" gomb is elérhető, amely lehetővé teszi a megadott adatok elküldését. Amennyiben hibás vagy nem létező adat kerül megadásra, egy felugró figyelmeztető üzenet jelenik meg. Hasonló figyelmeztetés jelenik meg abban az esetben is, ha a felhasználó nem ad meg felhasználónevet vagy jelszót. Ha a megadott adatok helyesek, a rendszer a kezdőoldalra irányítja a felhasználót. Kis képernyőn vagy mobil eszközön a bejelentkezési felület melletti kép a képernyő tetejére csúszik.

* Regisztrációs felület:

A felületen található felhasználónevet, vezetéknevet, keresztnevet, jelszót és feladatkört tartalmazó beviteli mező. A regisztrációs folyamatot egy gomb indítja, amely elküldi a megadott adatokat. Sikeres regisztráció után egy felugró ablak tájékoztatja a felhasználót a sikeres regisztrációról. Ez a felület kizárólag az adminisztrátorok számára elérhető, azonban a kötelező mezők kitöltését a rendszer minden esetben ellenőrzi, és hibaüzenetet küld, ha valamelyik mező üres marad. Kis képernyőn vagy mobil eszközön a regisztrációhoz tartozó kép a képernyő tetejére csúszik.

* Főképernyő:

Az oldalon, felhasználói jogosultságtól függően különböző kártyák jelennek meg, amelyek a többi felületre navigálnak. Amikor az egér a kártya fölé kerül, egy szöveg tűnik fel, amely az adott munkakört vagy munkafelületet jelöli.

* Étterem felület:

A bal oldalon egy navigációs sáv található, amely az elérhető ételtípusokat tartalmazza. Egy ételtípus csak akkor jelenik meg, ha van hozzá tartozó étel. Az ikonokra kattintva a terméksávban kizárólag az adott kategóriába tartozó ételek lesznek láthatók. Kis képernyőn a navigációs sáv a felső részre kerül.

A Rendelés sáv a folyamatban lévő rendelés tételeit listázza. A felsorolásban szerepel az étel ikonja, neve, darabszáma, ára és egy törlés gomb. A sáv tetején megjelenik a rendelésben lévő termékek összesített ára forintban.

A „Rendelés leadása” gombra kattintva egy párbeszédablak nyílik meg, ahol a felhasználó megadja a karszalag számát, ezzel hozzárendelve a rendelést. Az ablakban egy „Mégse” és egy „Leadás” gomb található. A „Leadás” gomb megnyomása után, ha a rendelés sikeres, egy felugró üzenet jelzi a sikeres leadást.

A termékek sávja a megjelenő ételeket listázza. Alapértelmezett képernyőméreten három kártya látható egymás mellett, de ez a szám a képernyő méretének csökkenésével arányosan csökken.

Minden kártyán szerepel az étel neve, kategóriája, képe, rövid leírása, valamint a rendelhetőségi státusza. A rendeléshez egy plusz és mínusz gomb segítségével adható hozzá az étel. Amikor az egér a gombok fölé kerül, azok színe megváltozik. Ha egy étel nem rendelhető, a hozzá tartozó kártya inaktívvá válik, és nem lehet hozzáadni a rendeléshez.

A fejléc tartalmazza a logót, a nevet, egy kezdőlap gombot, egy menü gombot, valamint a bejelentkezett felhasználó ikonját. Az ikonra kattintva egy legördülő menü jelenik meg, amelyben az általános funkciók érhetők el, például a kijelentkezés és a profil.

## Backend

### Specifikáció

A projektünk célja egy ASP.NET 8 backend alkalmazás fejlesztése, amely képes a fogyasztásnyilvántartó rendszer adatait szolgáltatni a frontend felé.

**Technológiák:**

* RESTful API-k, hitelesítési folyamatok JWT és token használata.

**Funkcionalitások:**

* CRUD műveletek, melyet a felhasználó jogosultságától függően tud elvégezni.

**Biztonsági szempontok**

* Érzékeny adatok, mint például a jelszavak hashelve kerülnek eltárolásra az adatbázisban.
* DTO-k használata, mely segítségével kizárólag a megfelelő adatok kerülnek megjelenítésre.

**Teljesítmény-igények**

* Gyors válaszidő.
* Több felhasználó egyidejűleg való kiszolgálása.

### Az alkalmazott fejlesztői eszközök

**Visual Studio 2022:**

A Visual Studio 2022 egy széles körben használt fejlesztőkörnyezet, amely támogatja az ASP.NET Web API fejlesztését C# nyelven. Beépített NuGet csomagkezelője lehetővé teszi a szükséges csomagok, például az Entity Framework Core, a Swashbuckle (Swagger) és egyéb API-fejlesztést támogató könyvtárak egyszerű hozzáadását és frissítését.

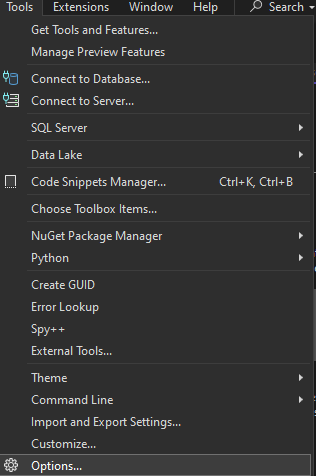
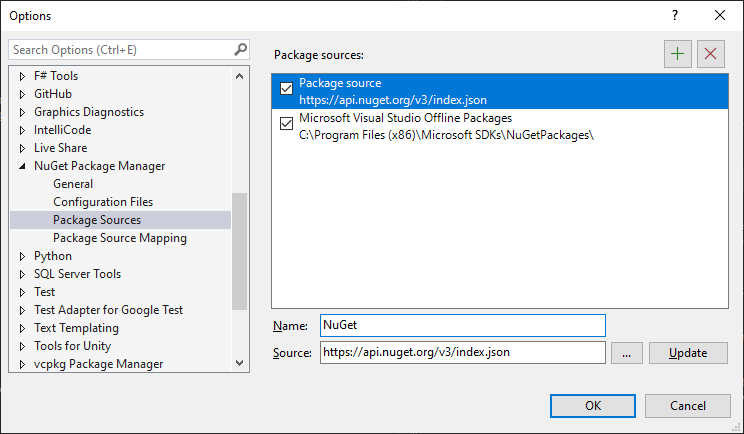
**Általunk alkalmazott NuGet csomagok (Dependency Management):**

A NuGet csomagokat NuGet Package Manager segítségével telepítettük és használtuk a programunkban. Alapértelmezetten a Visual Studio nem mindig tartalmazza a szükséges NuGet csomagforrást, ezért ezt manuálisan kell beállítani.

A beállítás elérhető a következő útvonalon:

*Tools* 🡪 *Options* 🡪*NuGet Package Manager* 🡪*Package Sources*

Itt adjunk hozzá egy új csomagforrást a következő értékekkel:

* **Name:** *NuGet*
* **Source:** *https://api.nuget.org/v3/index.json*
* BCrypt.Net-Next (by Chris McKee, Ryan D. Emerl, Damien Miller) – 4.0.3:
* Egy .NET-ben használható könyvtár, amely a BCrypt hash-elési algoritmus implementációját biztosítja. Lehetővé teszi jelszóhash-ek generálását és ellenőrzését, erős védelmet nyújtva a brute-force támadások ellen.
* Microsoft.AspNetCore.Authentication.JwtBearer (by Microsoft) – 8.0.13:
* Az Entity Framework Core egy modern objektum-relációs adatbázis leképző .NET keretrendszerhez, ami támogatja a LINQ lekérdezéseket, módosításokat, sémamigrációkat.
* Microsoft.EntityFrameworkCore (by Microsoft) – 9.0.2:
* Ez a middleware lehetővé teszi a JWT tokenek érvényesítését HTTP-kérések során, biztosítva a felhasználók hozzáférésének biztonságos kezelését.
* Microsoft.EntityFrameworkCore.Design (by Microsoft) – 9.0.2
* Az Entity Framework Core eszközeihez szükséges fejlesztési időben használt komponensek.
* Microsoft.EntityFrameworkCore.Sqlite (by Microsoft) – 9.0.2
* Sqlite adatbázis kezelő Entity Framework Core-hoz.
* Microsoft.EntityFrameworkCore.Tools (by Microsoft) – 9.0.2
* Az Entity Framework Core eszközeihez szükséges NuGet csomag.
* Microsoft.VisualStudio.Web.CodeGeneration.Design (by Microsoft) – 8.0.7 / 9.0.0:
* Kódgenerálás az ASP.NET Core-hoz, tartalmazza a dotnet-aspnet-codegenerator parancsot vezérlők és nézetek generálásához.
* Swashbuckle.AspNetCore (by domaindrivendev) – 6.6.2 / 7.2.0
* Kizárólag fejlesztői környezethez használt Swagger eszköz, mely képes a végpontok tesztelésére.

**Postman**

A Postman egy népszerű API-fejlesztő eszköz, amely lehetővé teszi HTTP-kérések küldését, API-k tesztelését és dokumentálását, támogatva az általunk is használt JWT hitelesítési módszert.

**Swagger UI**

A Swagger UI egy könnyen használható eszköz, amely lehetővé teszi a RESTful API-k dokumentációját. A fejlesztők megtekinthetik, tesztelhetik és kipróbálhatják az API végpontokat közvetlenül egy böngészőből, mely fejlesztési módban a backend szerver indításával automatikusan indul.

**Package Management Console**

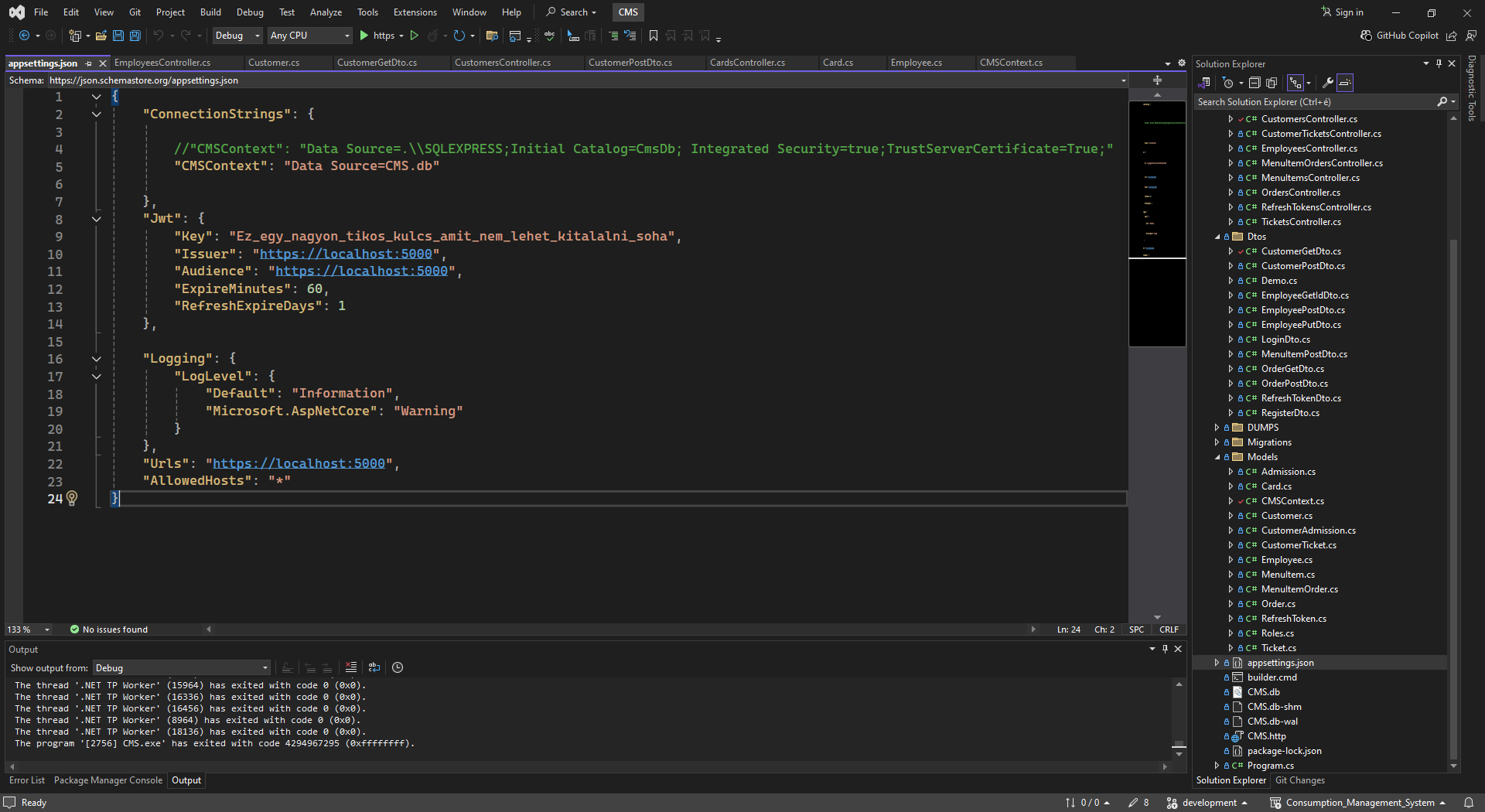
A Package Management Console a Visual Studio eszköze, amellyel NuGet csomagokat kezelhetünk és migrációkat futtathatunk, például Add-Migration és Update-Database parancsokkal.

### Adatmodell leírása

Az Entity Framework (EF) egy ORM (Object-Relational Mapping) keretrendszer, amely lehetővé teszi az adatbázis és az alkalmazás objektumai közötti leképezést. Az EF segítségével a fejlesztők közvetlenül objektumokkal dolgozhatnak SQL lekérdezések helyett, ami csökkenti a hibalehetőségeket és növeli a fejlesztési hatékonyságot.  
Az EF alapelvei közé tartoznak:

* **Code First:** Az adatbázis az osztályokból generálódik.
* **Database First:** Már meglévő adatbázisból generálódnak az osztályok.
* **Model First:** Modellező eszköz segítségével hozható létre az adatbázis.

Az EF támogatja az automatikus migrációkat, így az adatbázis változásai könnyen kezelhetők.

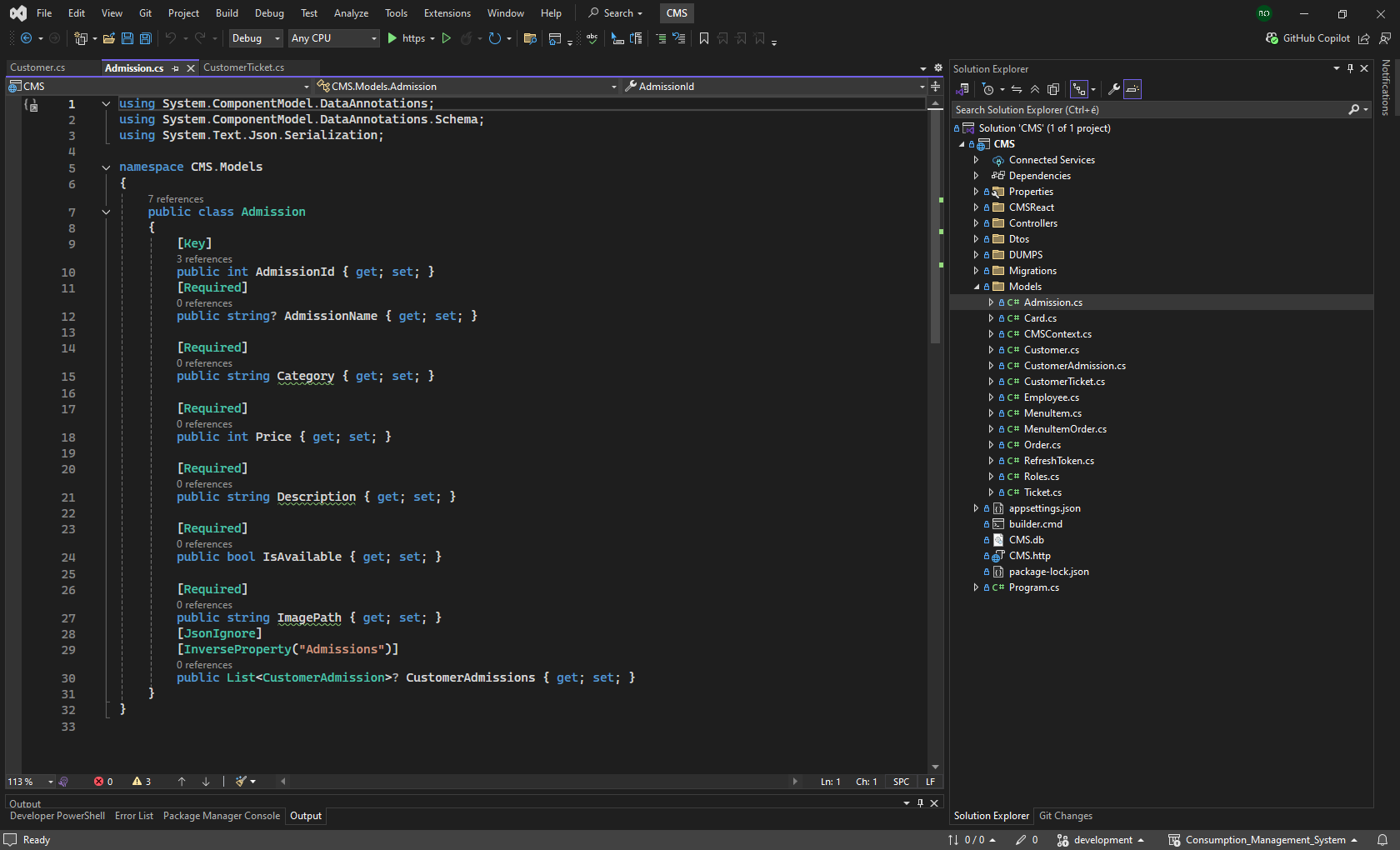
A backend legfontosabb beállításait, mint például a *ConnectionString* és a *Jwt* konfiguráció, az appsettings.json fájlban tároljuk.

A modellek használtuk az osztályok kialakításánál az annotációkat.

* **[Key]:** Az attribútumot az entitás azon tulajdonságának meghatározására használjuk, amely az elsődleges kulcsot képviseli az adatbázisban.
* **[Required]:** Arra használjuk, hogy megadjuk, hogy egy adott tulajdonság kötelező (nem lehet null). Az adatbázisban ennek megfelelően nem engedélyezi a NULL értékek tárolását.
* **[InverseProperty("*táblanév*")]:** A kapcsolódó entitások közötti viszonyok meghatározására használjuk. Az adott entitás kapcsolatát az összefüggő másik entitással jelöli. Ebben az esetben a *táblanév* a másik entitás tulajdonsága.

### Táblák és modelljeik:

#### Admissions / Admission

Ebben a táblában a különböző kiegészítő jegytípusok találhatóak, melyek az alapjegyhez vásárolhatók. Minden jegyhez tartozik egy egyedi azonosító, név, kategória, leírás, kép elérési útja, elérhetőség és ár. A tábla felel a kiegészítő jegyek egyszerű kezeléséért és értékesítéséért.

|  |  |  |  |
| --- | --- | --- | --- |
| Admissions | | | |
|  | Típus | Mezőnév | Megjegyzés |
| 🔑 | INTEGER | AdmissionId | Auto increment |
|  | TEXT | AdmissionName | A kiegészítő jegy típusa |
|  | TEXT | Category | A kiegészítő jegy kategóriája |
|  | TEXT | Description | A kiegészítő jegy rövid megnevezése |
|  | TEXT | ImagePath | A kiegészítő jegy képének relatív útvonala |
|  | INTEGER | IsAvailable | Elérhető-e az adott kiegészítő jegy |
|  | INTEGER | Price | A kiegészítő jegy ára |

#### Customers / Customer

Ebben a táblában tároljuk a vendégek adatait, mint például a kártyaazonosítót, nevet, létrehozási dátumot és aktivitási státuszt. A rendszer ezeket az információkat használja a vendég azonosítására, valamint a vásárlások és rendeléseik nyomon követésére. Az *IsActive* mező jelzi, hogy a vendég aktív-e a rendszerben.

|  |  |  |  |
| --- | --- | --- | --- |
| Customers | | | |
|  | Típus | Mezőnév | Megjegyzés |
| 🔑 | INTEGER | CustomerId | Auto increment |
|  | TEXT | CardId | A vendéghez hozzárendelt kártya azonosítója |
|  | TEXT | Name | Vendég keresztneve a megszólítás miatt |
|  | TEXT | CreatedAt | A vendég rekord létrehozásának dátuma |
|  | INTEGER | IsActive | A vendég engedélyezett-e |

#### CustomerAdmissions / CustomerAdmission

A *CustomerAdmissions* táblában tároljuk az *Admissions* és a *Customers* tábla közötti kapcsolatot. A *CustomerId* és az *AdmissionId* mezők hivatkoznak az egyes vendégek és kiegészítő jegyek azonosítójára, ezzel lehetővé téve, hogy egy vendég több kiegészítő jegyet is vásároljon. A *CustomerAdmissionId* mező egyedi azonosítót biztosít a rekordok számára.

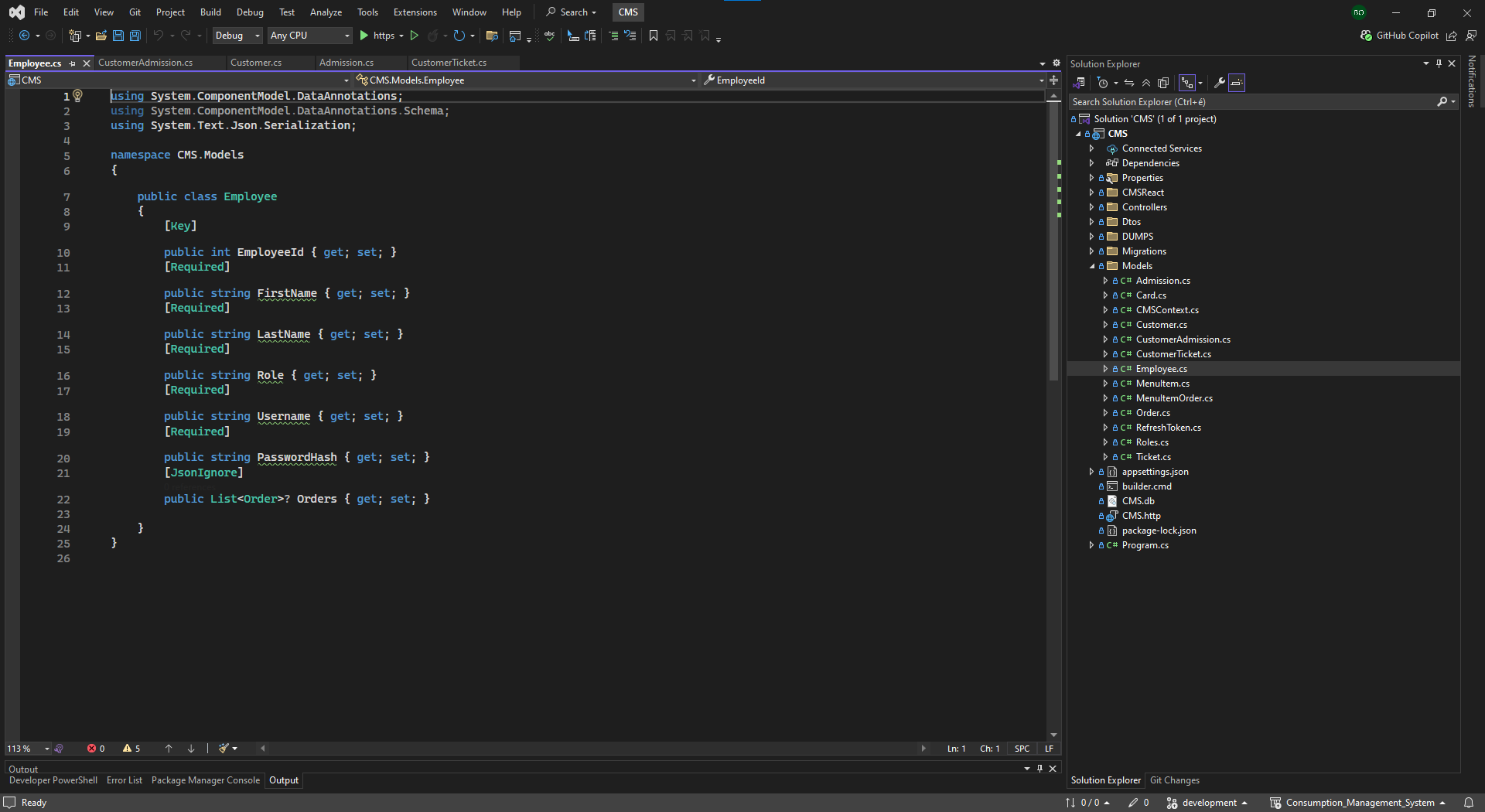
|  |  |  |  |
| --- | --- | --- | --- |
| CustomerAdmissions | | | |
|  | Típus | Mezőnév | Megjegyzés |
| 🔑 | INTEGER | CustomerAdmissionId | Auto increment |
|  | INTEGER | CustomerId | A „*Customers”* tábla azonosítója |
|  | INTEGER | AdmissionId | Az *„Admissions”* tábla azonosítója |

#### CustomerTickets / CustomerTicket

A *CustomerTicket*s tábla a *Customers* és a *Tickets* táblák közötti kapcsolatot tárolja. Egy vendég több jegyet is vásárolhat, így a tábla lehetővé teszi a vendégek és jegyek közötti kapcsolatok nyomon követését. A *CustomerId* és *TicketId* mezők az adott vásárló és jegy azonosítójára hivatkoznak.

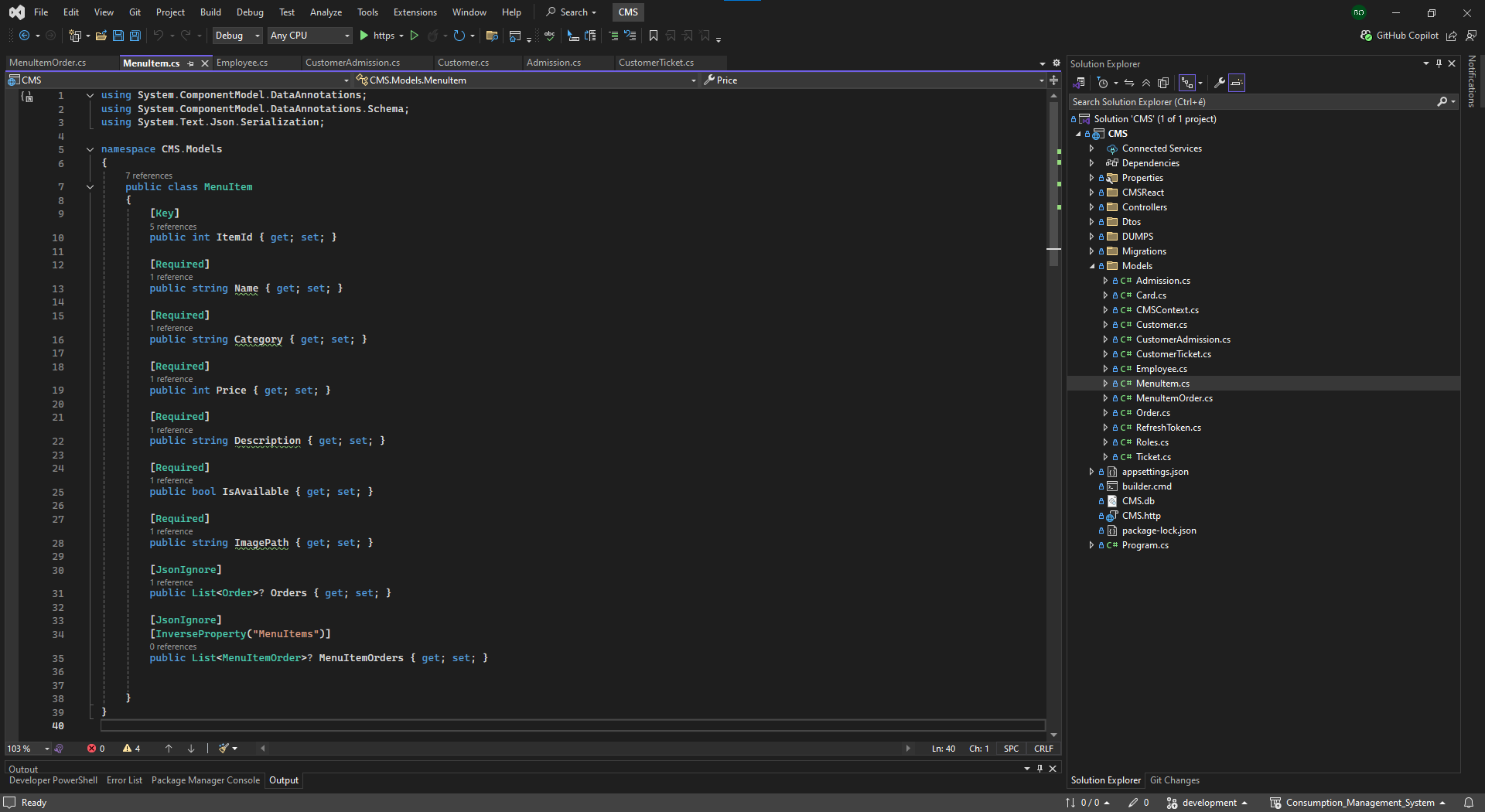
|  |  |  |  |
| --- | --- | --- | --- |
| CustomerTickets | | | |
|  | Típus | Mezőnév | Megjegyzés |
| 🔑 | INTEGER | CustomerTicketId | Auto increment |
|  | INTEGER | CustomerId | A „*Customers”* tábla azonosítója |
|  | INTEGER | TicketId | Az *„Ticket”* tábla azonosítója |

#### Employees / Employee

Ebben a táblában a dolgozók adatai vannak tárolva, amelyek a rendszerben való azonosításhoz és jogosultságkezeléshez szükségesek. A *EmployeeId* egyedi azonosítót ad, a *FirstName*, *LastName*, *Role*, *Username* és *PasswordHash* mezők pedig a dolgozók személyes adatait és hitelesítési információit tárolják.

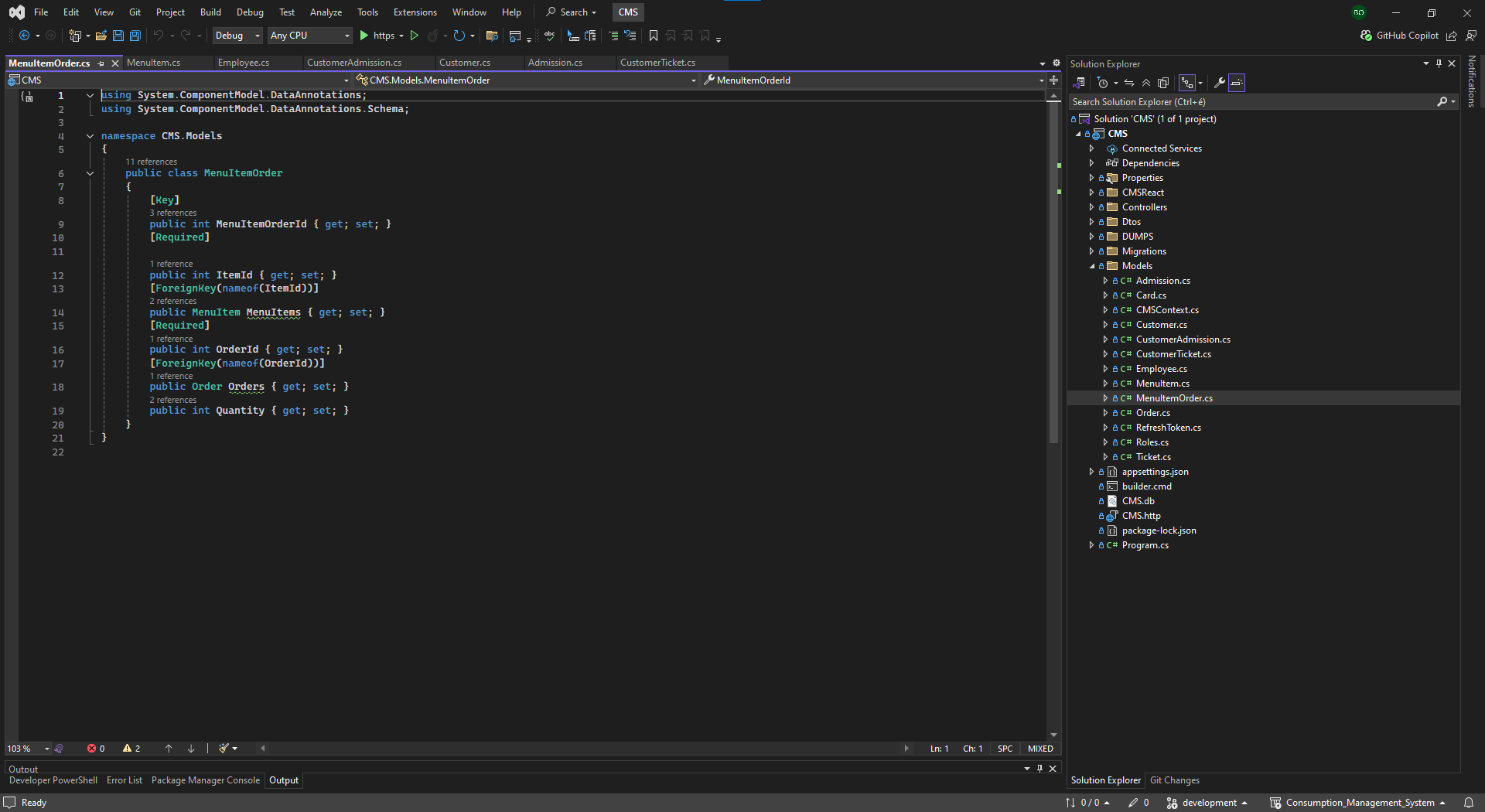
|  |  |  |  |
| --- | --- | --- | --- |
| Employees | | | |
|  | Típus | Mezőnév | Megjegyzés |
| 🔑 | INTEGER | EmployeeId | Auto increment |
|  | TEXT | FirstName | A dolgozó keresztneve |
|  | TEXT | LastName | A dolgozó vezetékneve |
|  | TEXT | Role | A dolgozó munkaköre, ez alapján tud belépni a különböző felületekre |
|  | TEXT | Username | A dolgozó felhasználóneve |
|  | TEXT | PasswordHash | Jelszó hash-elve |

#### MenuItems / MenuItem

Ebben a táblában tároljuk az étteremben elérhető termékek adatait. A *ItemId* mező egyedi azonosítót biztosít minden termék számára. A *Name* mező a termék nevét, míg a *Category* a termék kategóriáját tartalmazza. A *Price* mező a termék árát forintban, a *Description* pedig egy rövid leírást ad a termékről. Az *IsAvailable* mező jelzi, hogy a termék elérhető-e, míg az *ImagePath* a termék képének relatív elérési útját tárolja.

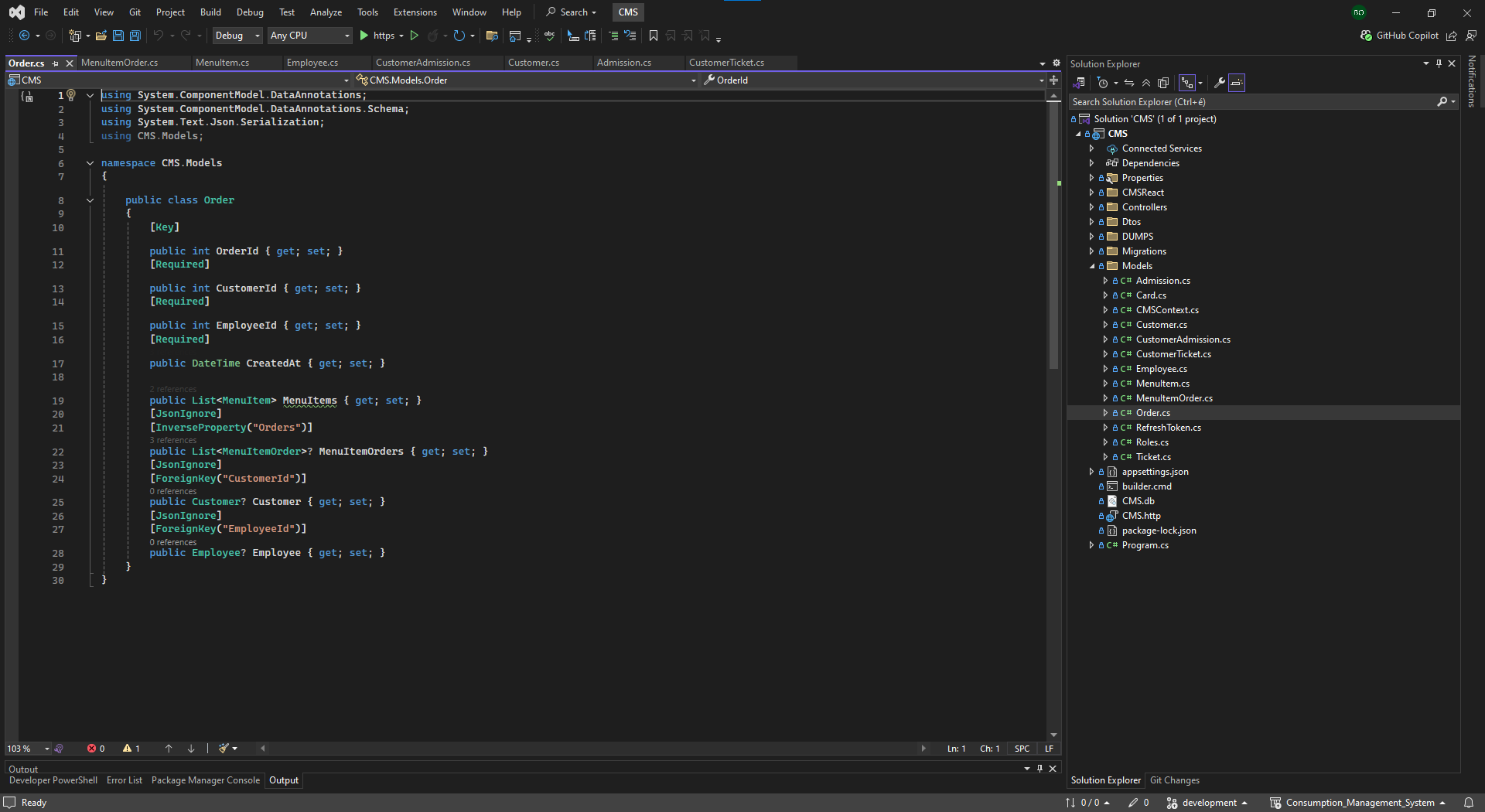
|  |  |  |  |
| --- | --- | --- | --- |
| MenuItems | | | |
|  | Típus | Mezőnév | Megjegyzés |
| 🔑 | INTEGER | ItemId | Auto increment |
|  | TEXT | Name | A termék neve |
|  | TEXT | Category | A termék kategóriája |
|  | INTEGER | Pice | A termék ára forintban |
|  | TEXT | Description | Egy rövid leírás a termékről |
|  | INTEGER | IsAvailable | Elérhető-e az adott termék |
|  | TEXT | ImagePath | A termék képének relatív útvonala |

#### MenuItemOrders / MenuItemOrder

A *MenuItemOrders* táblát kapcsolatként használjuk a *MenuItems* és az *Orders* táblák között, hogy egy étlap elem több rendelésben is szerepelhessen, illetve egy rendelés több étlap elemet is tartalmazhasson. Ezen felül a táblában szerepel egy *Quantity* mező, amely tárolja, hogy egy adott rendelésben hány darab étlap elem szerepel.

|  |  |  |  |
| --- | --- | --- | --- |
| MenuItemOrders | | | |
|  | Típus | Mezőnév | Megjegyzés |
| 🔑 | INTEGER | MenuItemOrderId | Auto increment |
|  | INTEGER | ItemId | A „*MenuItem”* tábla azonosítója |
|  | INTEGER | OrderId | Az *„Orders”* tábla azonosítója |
|  | INTEGER | Quantity | Mennyiséget meghatározó mező |

#### Orders / Order

Az *Orders* osztály a vendégek rendeléseit tárolja. Ez az osztály pontosan meghatározza, hogy egy adott rendelés mikor történt, ki a vendég, aki a rendelést leadta, és ki a dolgozó, aki rögzítette azt. Az *Orders* táblában tárolt adatok segítenek nyomon követni a rendeléseket az étterem vagy szolgáltatás számára.

|  |  |  |  |
| --- | --- | --- | --- |
| Orders | | | |
|  | Típus | Mezőnév | Megjegyzés |
| 🔑 | INTEGER | OrderId | Auto increment |
|  | INTEGER | CustomerId | A vendég azonosítója |
|  | INTEGER | EmployeeId | A dolgozó azonosítója |
|  | TEXT | CreatedAt | Rögzíti, hogy mikor került rögzítésre a rendelés |

#### RefreshTokens / RefreshToken

A *RefreshTokens* osztály feladata a tokenek tárolása, amelyek biztosítják a rendszert használó felhasználók bejelentkezését és hitelesítését. Minden token egy adott felhasználóhoz (dolgozóhoz) tartozik, és meghatározott idő után lejár.

|  |  |  |  |
| --- | --- | --- | --- |
| RefreshTokens | | | |
|  | Típus | Mezőnév | Megjegyzés |
| 🔑 | INTEGER | Id | Auto increment |
|  | TEXT | Token | A token kulcs |
|  | TEXT | Expires | A token lejárati dátuma |
|  | INTEGER | EmployeeId | Itt tárolódik, hogy az adott token melyik felhasználóhoz tartozik |

#### Tickets

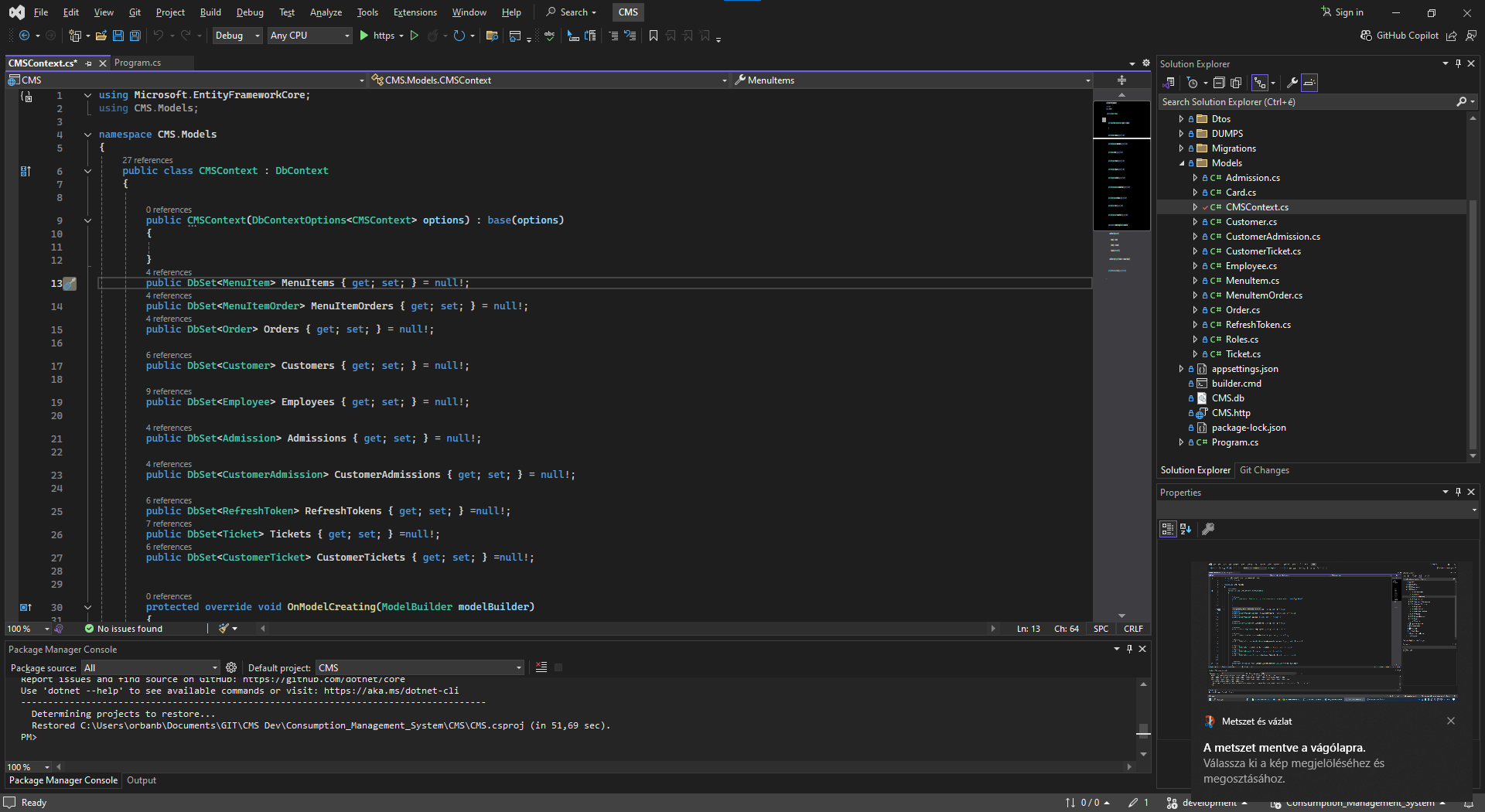
A *Tickets* osztály a fő kategóriába tartozó jegyek tárolására szolgál. Minden vendégnek először egy ilyen típusú jeggyel kell rendelkeznie a belépéshez.

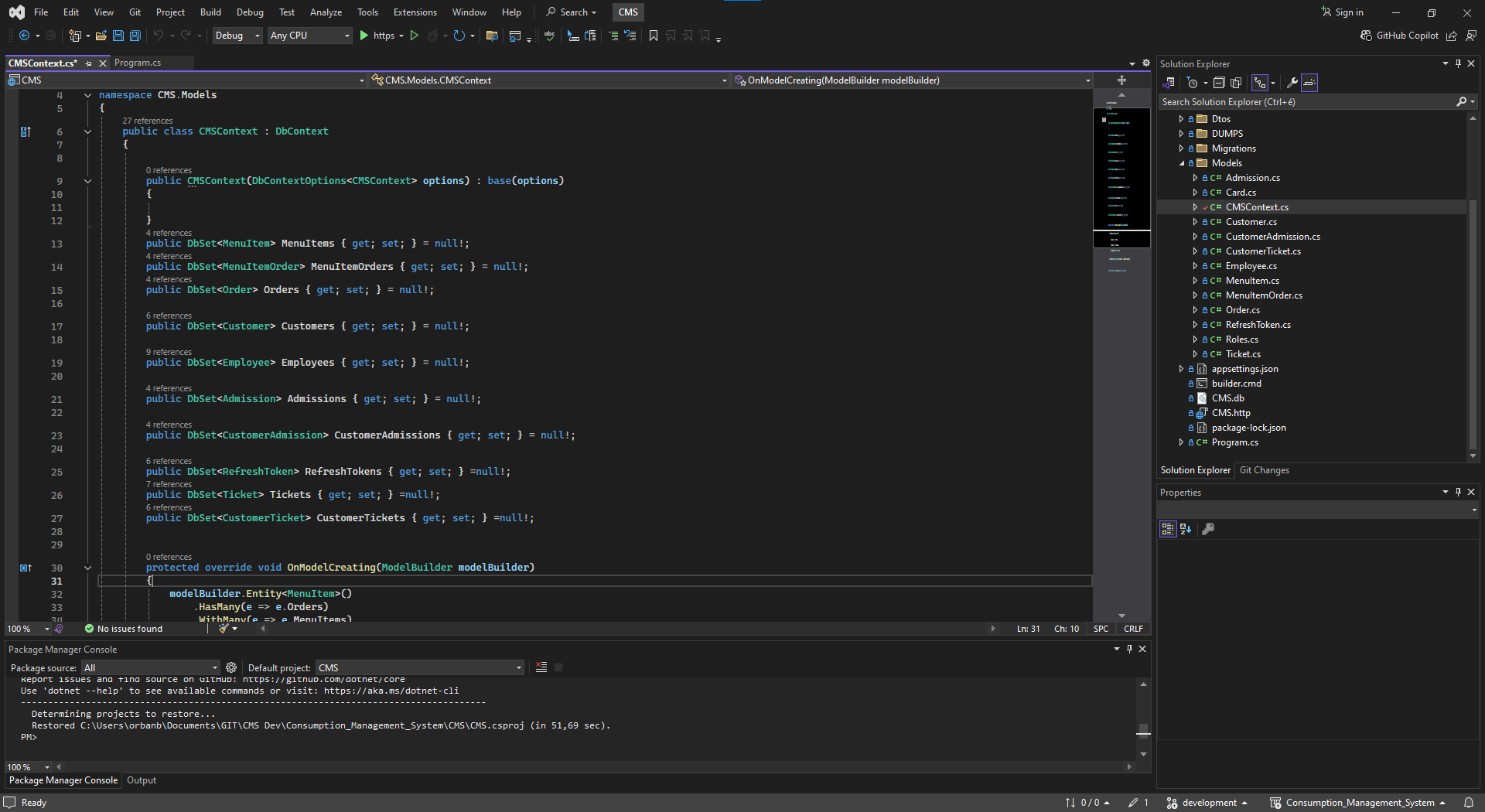
|  |  |  |  |
| --- | --- | --- | --- |
| Tickets | | | |
|  | Típus | Mezőnév | Megjegyzés |
| 🔑 | INTEGER | TicketId | Auto increment |
|  | TEXT | TicketName | A jegy pontos megnevezése |
|  | TEXT | Category | A jegy kategóriája |
|  | TEXT | Description | A jegy rövid megnevezése |
|  | TEXT | ImagePath | A jegy képének relatív útvonala |
|  | INTEGER | IsAvailable | Elérhető-e az adott jegy |
|  | INTEGER | Price | A jegy ára |

### Kontextus osztály

A *CMSContext.cs* osztály az Entity Framework Core alapértelmezett adatbázis-kapcsolat kezelője, amely a *DbContext* osztályból származik. A *DbContext* osztály biztosítja az adatbázis műveletek kezelését, és a *CMSContext.cs* osztály egyéni alkalmazás-specifikus implementációját tartalmazza.

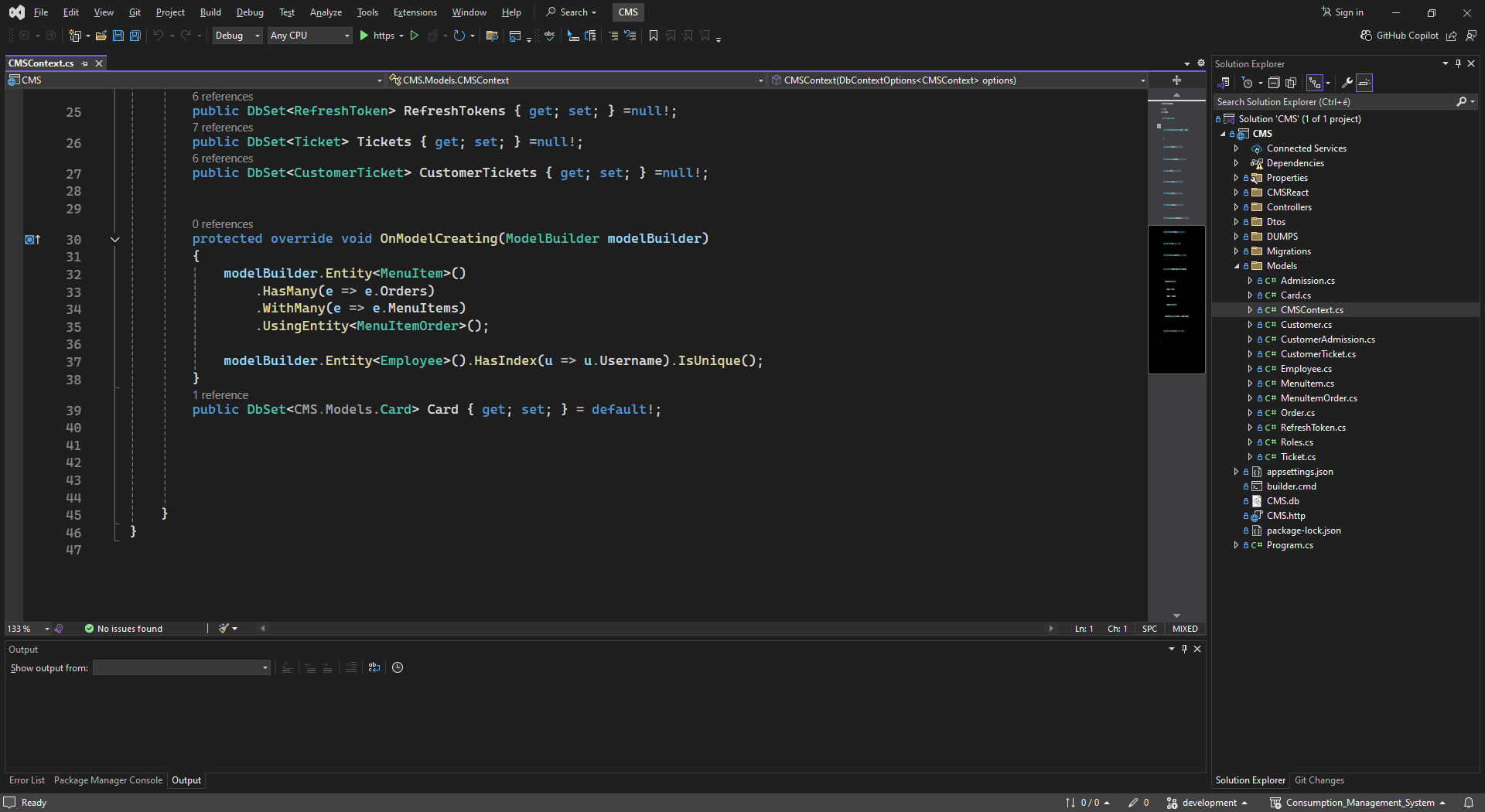
### Dependency Injection (DI)

A Dependency Injection (DI) egy tervezési minta, amely segít az alkalmazás modulárisabbá és tesztelhetőbbé tételében azáltal, hogy a szükséges függőségeket külső forrásból injektálja az osztályokba, ahelyett, hogy azok saját maguk hozzák létre.

A DbSet tulajdonságok az Entity Framework Core-ban az adatbázis táblákat reprezentálják. Itt definiáljuk az adatbázisban szereplő táblák neveit. Minden DbSet tulajdonság egy-egy entitást reprezentál, amely az adatbázis egy-egy táblájához kapcsolódik. Ezek a tulajdonságok a CMSContext osztályban találhatóak, és az adatbázis műveletek végrehajtásához szükségesek.

### OnModelCreating Metódus és Entitás Kapcsolatok Konfigurálása

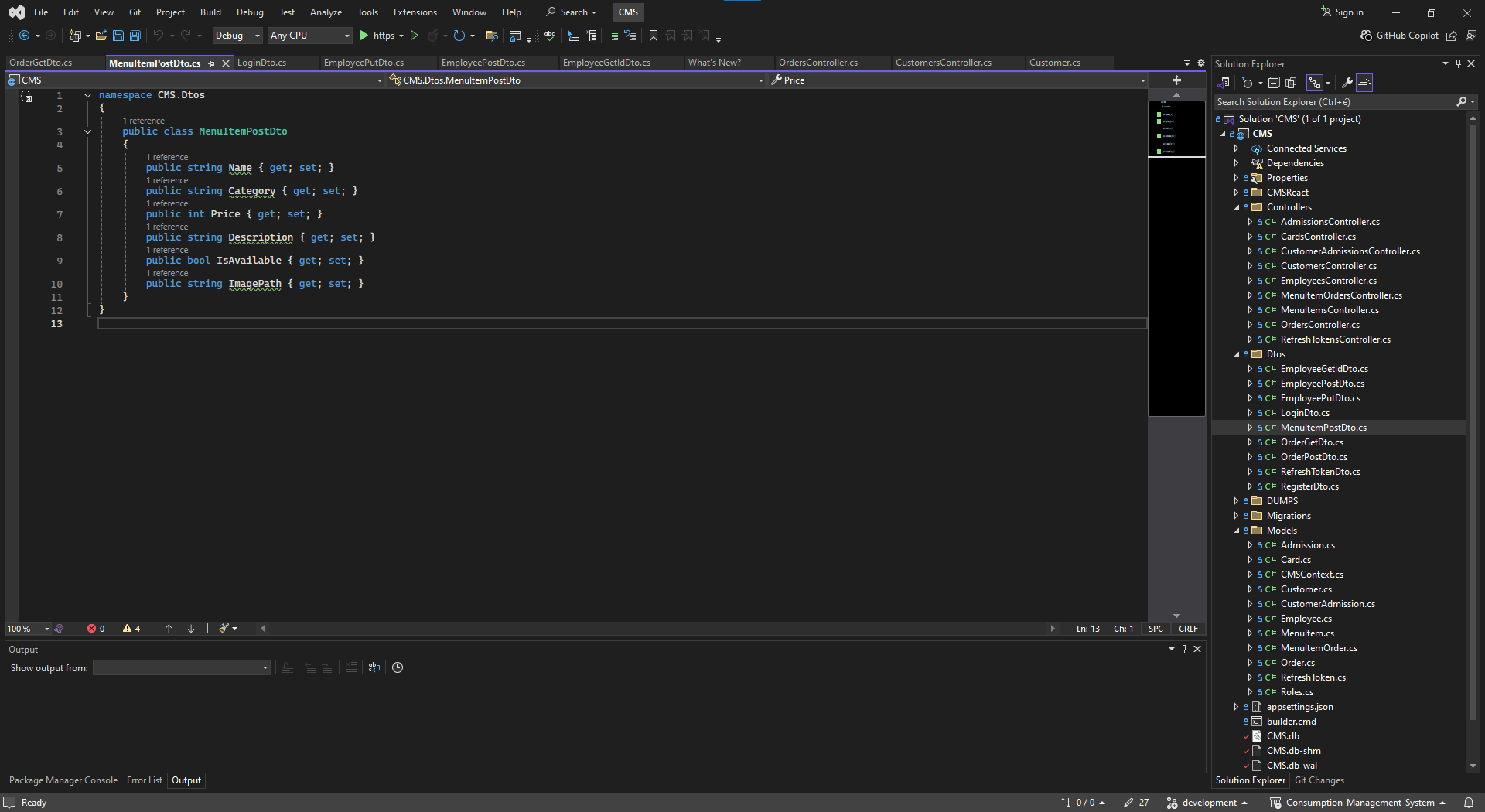
Az **OnModelCreating** metódus lehetővé teszi az entitások közötti kapcsolatok és egyedi szabályok konfigurálását az adatbázisban.

Itt kerül beállításra egy **több a többhöz** kapcsolat is a **MenuItem** és **Order** entitások között egy köztes **MenuItemOrder** entitás segítségével:

### DTO-k (Data Transfer Object)

A DTO (Data Transfer Object) egy egyszerű adatobjektum, amelyet az adatok különböző rendszerek vagy alkalmazásrétegek közötti továbbítására használnak. Az API hívásokban alkalmazva lehetővé teszi az adatok strukturált, egyszerűsített és biztonságos kezelését. A DTO-k segítenek csökkenteni a rendszer bonyolultságát, és biztosítják, hogy csak a szükséges információk kerüljenek továbbításra, miközben támogatják a hatékony kommunikációt a frontend és a backend között.

#### MenuItemPostDto

A MenuItemPostDto osztály az étterem rendelési rendszerében új menü tételek hozzáadására szolgál. A DTO mezői a következő adatokat tartalmazzák:

#### Tulajdonságok

* **Name**: Az étel vagy ital neve.
* **Category**: Az étel vagy ital kategóriája (pl. főétel, desszert).
* **Price**: Az étel ára.
* **Description**: Az étel leírása.
* **IsAvailable**: Az étel elérhetősége (elérhető vagy nem).
* **ImagePath**: Az ételhez tartozó kép elérési útja.

#### A DTO előnyei

* **Karbantarthatóság:** A DTO-k segítenek a kód karbantartásában, mivel minden adat egy jól meghatározott objektumban van, így az alkalmazás komponensei közötti kommunikáció egyszerűbbé válik.
* **Teljesítmény:** A DTO-k csak a szükséges adatokat továbbítják, csökkentve ezzel a hálózati terhelést és optimalizálva az adatforgalmat.
* **Biztonság:** Az érzékeny adatokat nem szükséges közvetlenül küldeni vagy tárolni, mivel a DTO-k lehetőséget biztosítanak az adatok biztonságos kezelésére.

### Controllerek

A **Controller** egy olyan osztály az **ASP.NET Core Web API** alkalmazásokban, amely az ügyfelektől érkező HTTP-kérések kezeléséért felel. A **MVC (Model-View-Controller)** architektúra egyik központi eleme, amely a **bejövő kéréseket feldolgozza, és visszaadja a választ** a kliensnek.

#### Útvonalkezelés (Routing)

* A **[Route]** attribútum segítségével meghatározza, hogy egy adott végpont (API endpoint) milyen URL-cím alatt érhető el.
* Példa: **[Route("api/[controller]")]** → Automatikusan az osztály nevéből  
  (pl. CustomersController) generálódik az útvonal: **api/Customers**.

#### HTTP-műveletek kezelése

* **[HttpGet]** – Adatok lekérése
* **[HttpPost]** – Új adatok létrehozása
* **[HttpPut]** – Meglévő adatok frissítése
* **[HttpDelete] –** Adatok törlése

#### Adatkezelés és a szűrés végrehajtása

Az **Entity Framework Core** segítségével adatbázis-műveleteket hajt végre (pl. adatok lekérése, módosítása, törlése).

Segédfüggvényeket tartalmazhat (pl. létezik-e az adott rekord).

Válaszok visszaadása a kliensnek

* **200 OK** – Sikeres kérés esetén.
* **201 Created** – Új adat sikeres létrehozásakor.
* **400 Bad Request** – Hibás kérés esetén.
* **404 Not Found** – Ha a keresett adat nem található.
* **500 Internal Server Error** – Ha szerverhiba történik.

#### MenuItemsController

A MenuItemsController a menüpontok kezelésére szolgál. Ez a vezérlő lehetővé teszi a menüelemek CRUD (Create, Read, Update, Delete) műveleteit.

Elérhető végpontok:

* GET /api/MenuItems: Az összes menüpont lekérése.
* GET /api/MenuItems/{id}: Egy adott menüpont lekérése az ID alapján.
* POST /api/MenuItems: Új menüpont létrehozása.
* PUT /api/MenuItems/{id}: Egy meglévő menüpont frissítése.
* DELETE /api/MenuItems/{id}: Egy adott menüpont törlése.

#### MenuItemOrdersController

A MenuItemOrdersController a menüpontok rendelésének kezelésére szolgál, és biztosítja a menüpontok és a rendelés közötti kapcsolatot.

Elérhető végpontok:

* GET /api/MenuItemOrders: Az összes rendelést tartalmazó menüpont lekérése.
* GET /api/MenuItemOrders/{id}: Egy adott rendelés részletes adatainak lekérdezése.
* POST /api/MenuItemOrders: Új rendelés létrehozása.

Fontosabb megjegyzések:

* A rendeléseket aszinkron módon kezeli, hogy biztosítsa a gyors válaszidőt.
* A rendszer biztosítja az adatvalidálást a bemeneti DTO-n keresztül, biztosítva a helyes adatstruktúrát.

#### OrdersController

Az OrdersController a rendelésekkel kapcsolatos adatokat kezeli, beleértve az új rendelések létrehozását és az egyes rendelésekkel kapcsolatos információk lekérdezését.

Elérhető végpontok:

* GET /api/Orders: Az összes rendelés lekérése.
* GET /api/Orders/{id}: Egy adott rendelés részletes adatainak lekérdezése.
* POST /api/Orders: Új rendelés létrehozása.

Fontosabb megjegyzések:

* Az OrderPostDto típus biztosítja a rendeléshez szükséges adatok validálását.
* A rendeléshez tartozó menüpontok az OrderMenuItem kapcsolat segítségével kerülnek tárolásra.

#### AuthController

Az AuthController a felhasználói hitelesítést és jogosultságokat kezeli, beleértve a regisztrációt, bejelentkezést, refresh token kezelését és a kijelentkezést.

Elérhető végpontok:

* POST /api/Auth/register: Új felhasználó regisztrálása.
* POST /api/Auth/login: Bejelentkezés, JWT token és refresh token generálása.
* POST /api/Auth/refreshToken: Frissítő token alapján új JWT token generálása.
* POST /api/Auth/logout: Kijelentkezés, refresh token törlése.

#### TicketsController

A TicketsController a jegyek (ticketek) kezelésére szolgál, amely lehetővé teszi a jegyek CRUD műveleteit a rendszerben.

Elérhető végpontok:

* GET /api/Tickets: Az összes jegy lekérése.
* GET /api/Tickets/{id}: Egy adott jegy lekérése.
* POST /api/Tickets: Új jegy létrehozása.
* PUT /api/Tickets/{id}: Egy jegy frissítése.
* DELETE /api/Tickets/{id}: Egy jegy törlése.

#### Általános Biztonsági Intézkedések és Validálás

Minden vezérlő külön figyelmet fordít az adatok validálására és az érvénytelen adatok elutasítására. Az *Entity Framework ORM* biztosítja a megfelelő adatbázis műveleteket és az adatintégritást. Az authentikáció és autorizáció alapvető biztonsági mechanizmusokkal rendelkezik, beleértve a *JWT* tokenek és a refresh tokenek kezelését.

A már említett *DTO*-k szerepe kulcsfontosságú az adatok áramlásában, mivel lehetővé teszik a bemeneti és kimeneti adatok validálását és tisztaságát, biztosítva ezzel a rendszer biztonságát és megbízhatóságát.