BZSH Külkereskedelmi Technikum
Székhely:1065 Budapest, Révay utca 16
OM azonosító: 100536

FEJLESZTŐI-DOKUMENTÁCIÓ

Vizsgázók nevei:

Fekete Gergely Zoltán

Andó Attila

Róth József

5 0613 12 03 - Szoftverfejlesztő és -tesztelő technikus szakma

2023

TARTALOMJEGYZÉK

Bevezetés

A dokumentációban bemutatjuk az MHEI projektmunka dokumentációját.
A projekt céljai közé tartozik ,hogy a magyar horgászni vágyó társadalmat a lehető legjobban illetve legmodernebb formában tudjon tájékoztatni az új rendeletekről, bevezetett szabályok/ szabványokról. Egy helyről tudjon értesülni minden általunk fontosnak tartott információról.
Főleg azért fontos mert a magyar társadalom egy 2022-es felmérés szerint több mint 80%-a a felmérésen úgy tapasztalta, hogy az interneten fellelhető weboldalakról nem kap megfelelő tájékoztatást. Nincs elég információ vagy olyan oldal ahol gyakorlatilag bármit meg tudna találni. Ezért is törekedtünk az oldalunk egyszerű felépítéséhez , hogy minden korosztály számára könnyen kezelhető legyen az oldal. És remélhetőleg mindenki talál magának megfelelő információt, amire szüksége van.
Az MHEI egy kitalált nem létező név. A rövidítés A Magyar Horgászok Egyesületi Információit tartalmazza.
Próbáltunk már a rövidítésre is kellően odafigyelni ,hogy megfelelően lefedje a téma választásunkat.

A dokumentációba fejlesztői szemszögből vizsgáljuk a projekt felépítését és működését. Főképp a project elosztására, értendő, Frontend, Backend, és Electron-ra lehet érteni.

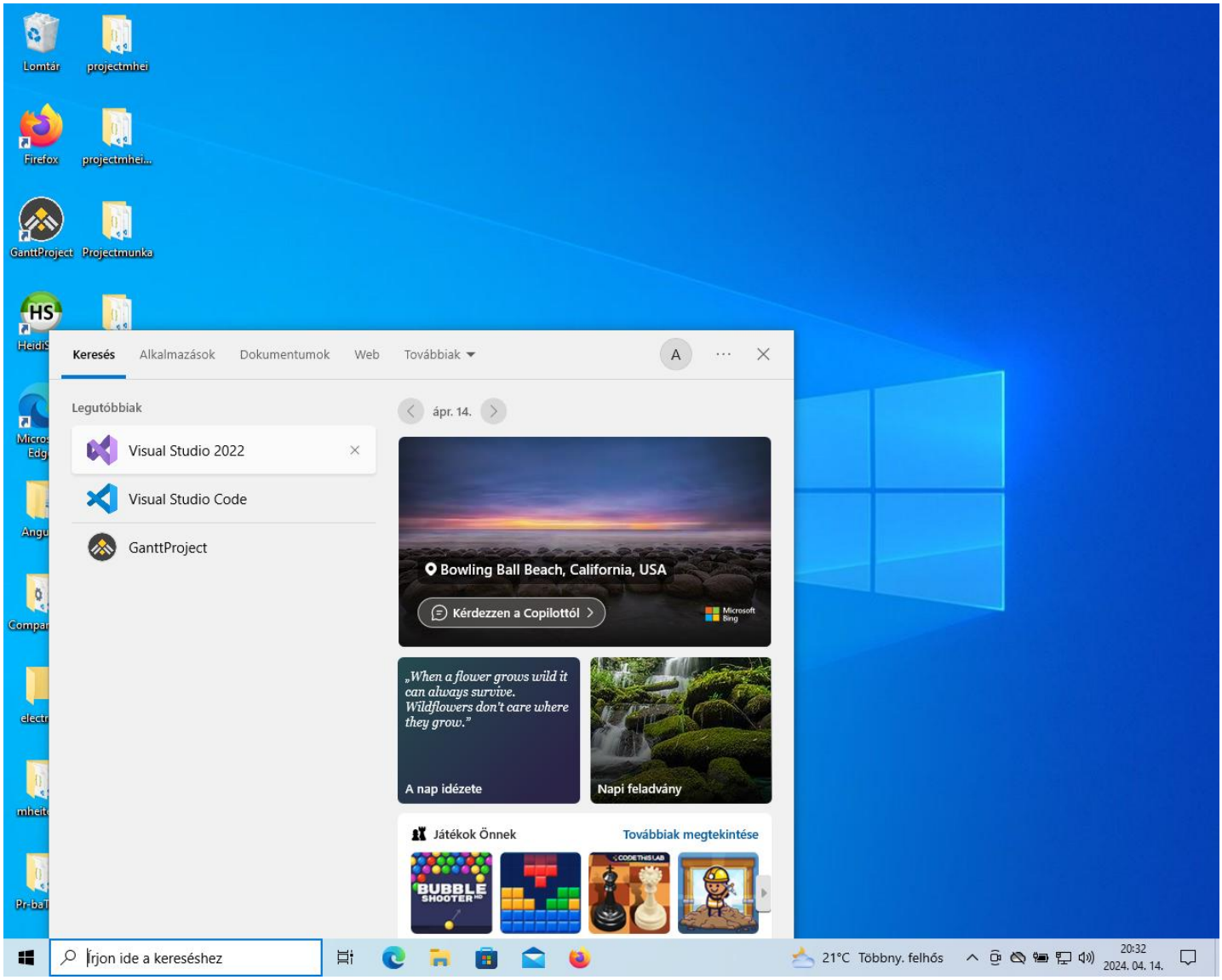
Projekt felosztását Gantt diagram formátumba szemléltetjük. Ott megfelelő formába látszik ,napokra lebontva ,hogy pontosan melyik nap ki foglalkozott a projekttel és mit szerkesztett rajta.

Felhasznált technológiák : A projekt során törekedtünk a tanórai anyag beépítésére, felhasználására. Az órával egyszerre haladni a projektünk megvalósításával. Illetve a fejlesztéshez kívánt programokról is a tanórán szerzett ismeretek szerint haladtunk.

NÉV	Felhasznált program	Verziószám
Fekete Gergely Zoltán	Angular	Angular CLI: 16.2.12
Andó Attila	C#	Microsoft Visual Studio: 17.2
Andó Attila	Electron	v.29.3.0
Róth József	Electron Setup - MHEI.exe	v.29.3.0
Róth József	Photoshop, DaVinci Resolve	25.6, 18.6

Backend C#

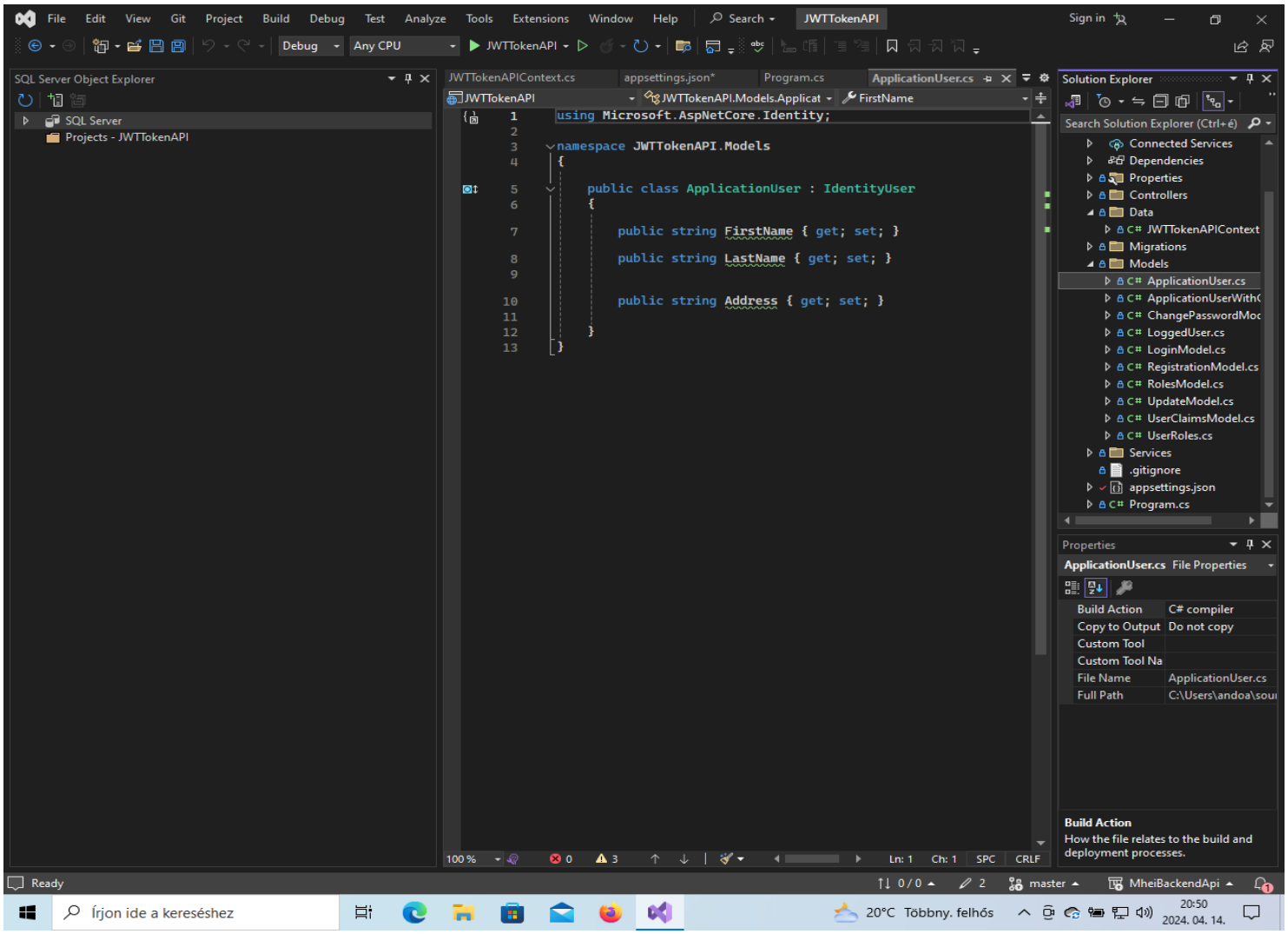
A Projektet visual studio 2022-be csináltuk. Első képeken a projekt megnyitása illetve az elindítása látszik.



Kiegészítettük még egy táblával amihez létrehoztunk egy modellt, amit Events néven neveztünk el és abban létrehoztunk változókat.Utána létrehozunk egy controllert ami EventController néven nevezzük el és megcsináljuk azokat a műveleteket amik kellenek nekünk(pl .Törlés Hozzáadás,Frissítés

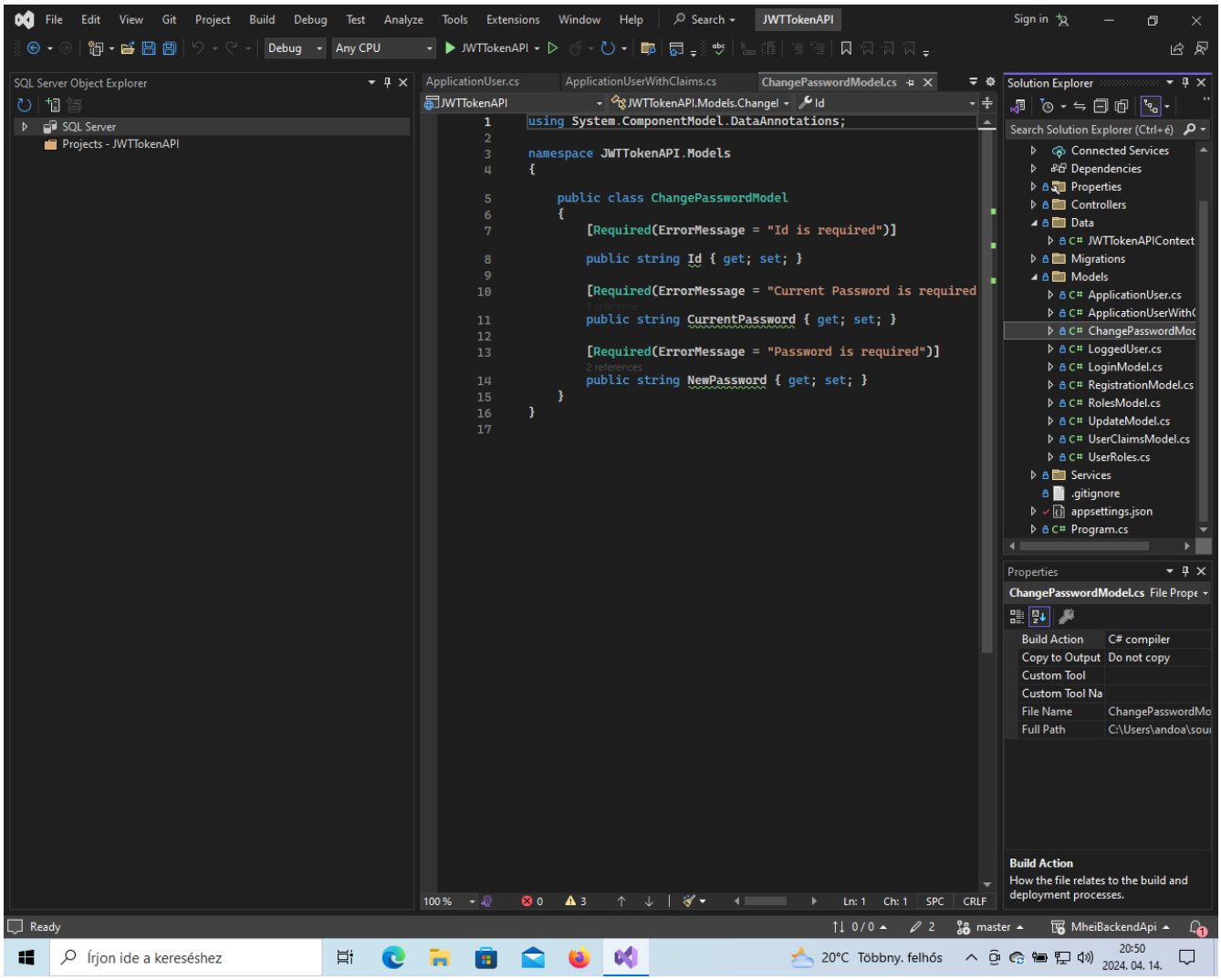
Kiválasztod neked megfelelő környezetet ebben az esetben a ASP.NET Core Web API. Adsz neki egy nevet pl. MheiBackendApi .

Után rákattintasz a Létrehozásra. Ott fogsz találni egy Controllers Mappát Egy Properties mappát egy Solutiont egy Program.cs és egy Weather Forecast filet. Kitörlöd a Controllers Mappából a fájlokat mert arra nem lesz szükség meg a WeatherForeCast Filet.Létrehozni először is egy Models mappát egy Services Mappát és egy Data mappát. A Models mappában létrehozol egy ApplicationUser létrehozását modellben s ezeket a változásokat hozzuk létre.



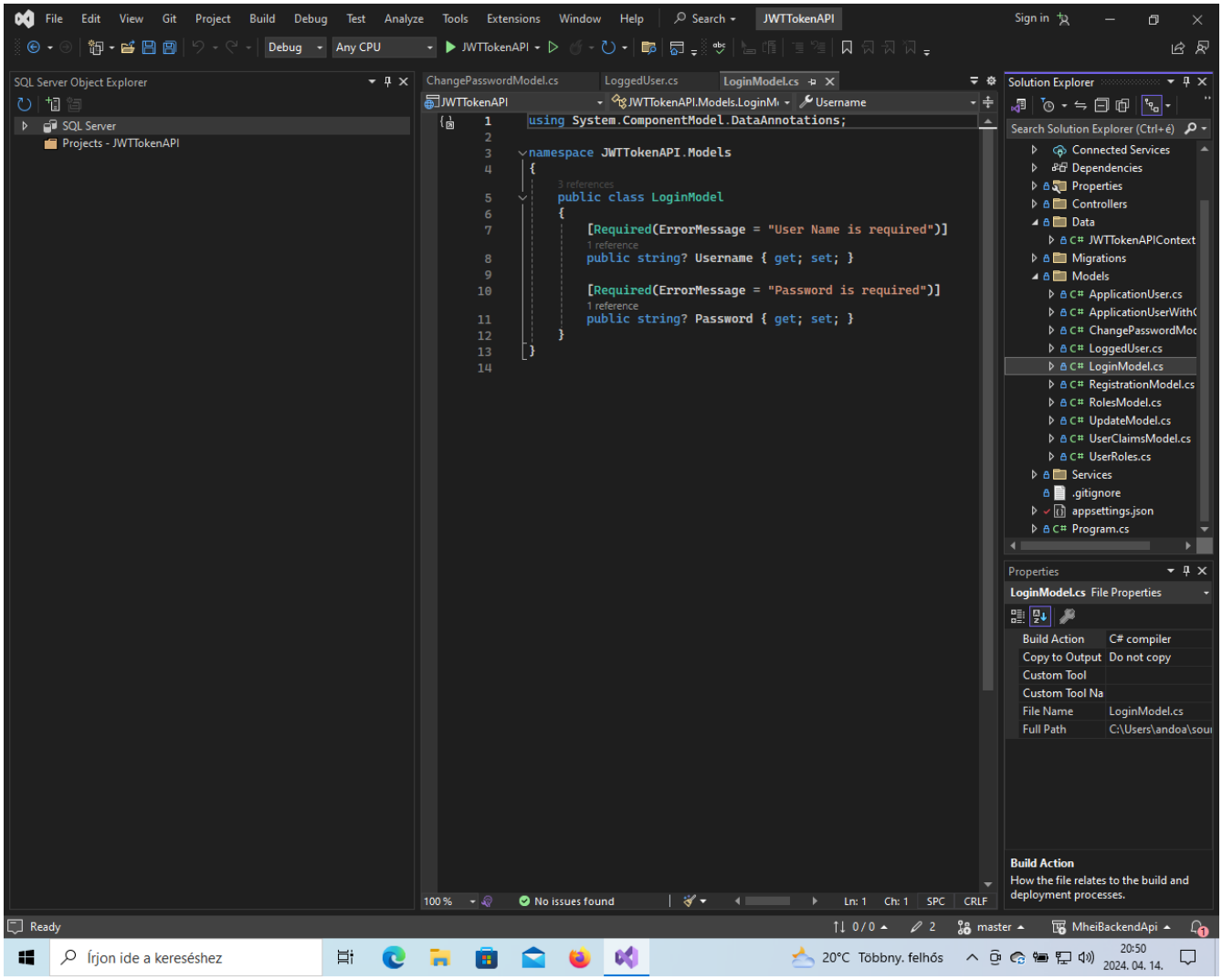
A saját osztályhoz hozzáadsz néhány egyedi tulajdonságot, mint például a keresztnév, vezetéknév és cím. Ezáltal bővíti az alap IdentityUser osztályt, hogy megfeleljen az alkalmazás egyedi igényeinek, és további felhasználói adatokat tárol hasson. Az ApplicationUserWithClaimsben létrehozzuk a Claims amihez nem rendeltünk hozzá értéket és az értéke is lehet nulla.

Most jön a ChangePassword nevű modell ami három tulajdonságot tartalmaz: Id, CurrentPassword és NewPassword.



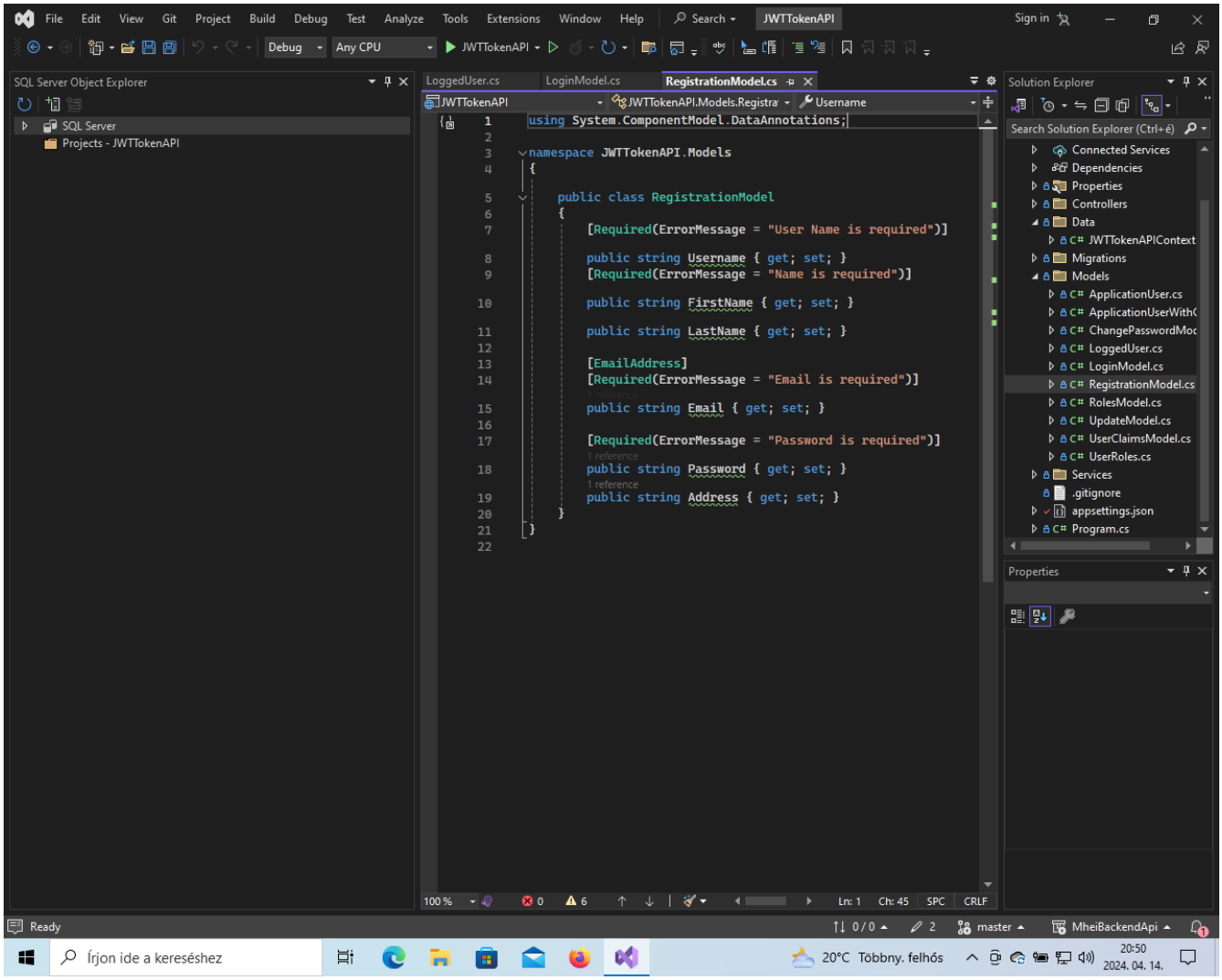
Ezek mindegyike kötelező (Required), és a megadott hibaüzenetet jeleníti meg, ha nincsenek megadva értékek. Ez a System.ComponentModel.DataAnnotations névtérben található attribútumokat használja az adatok validálásához.
Utána létrehozzuk a LoggedUser nevű modellt ami kettő tulajdonságot tartalmaz: User, amely egy ApplicationUser objektumot tárol, és Token, amely egy karakterláncot tárol. Az osztály tartalmaz egy konstruktort is, amely inicializálja ezeket a tulajdonságokat az osztály létrehozásakor.

Létrehozzuk a Login Modellt ami szintén kettő tulajdonságot tartalmaz Username, amely egy karakterláncot tárol, és Password, amely szintén egy karakterláncot tárol.



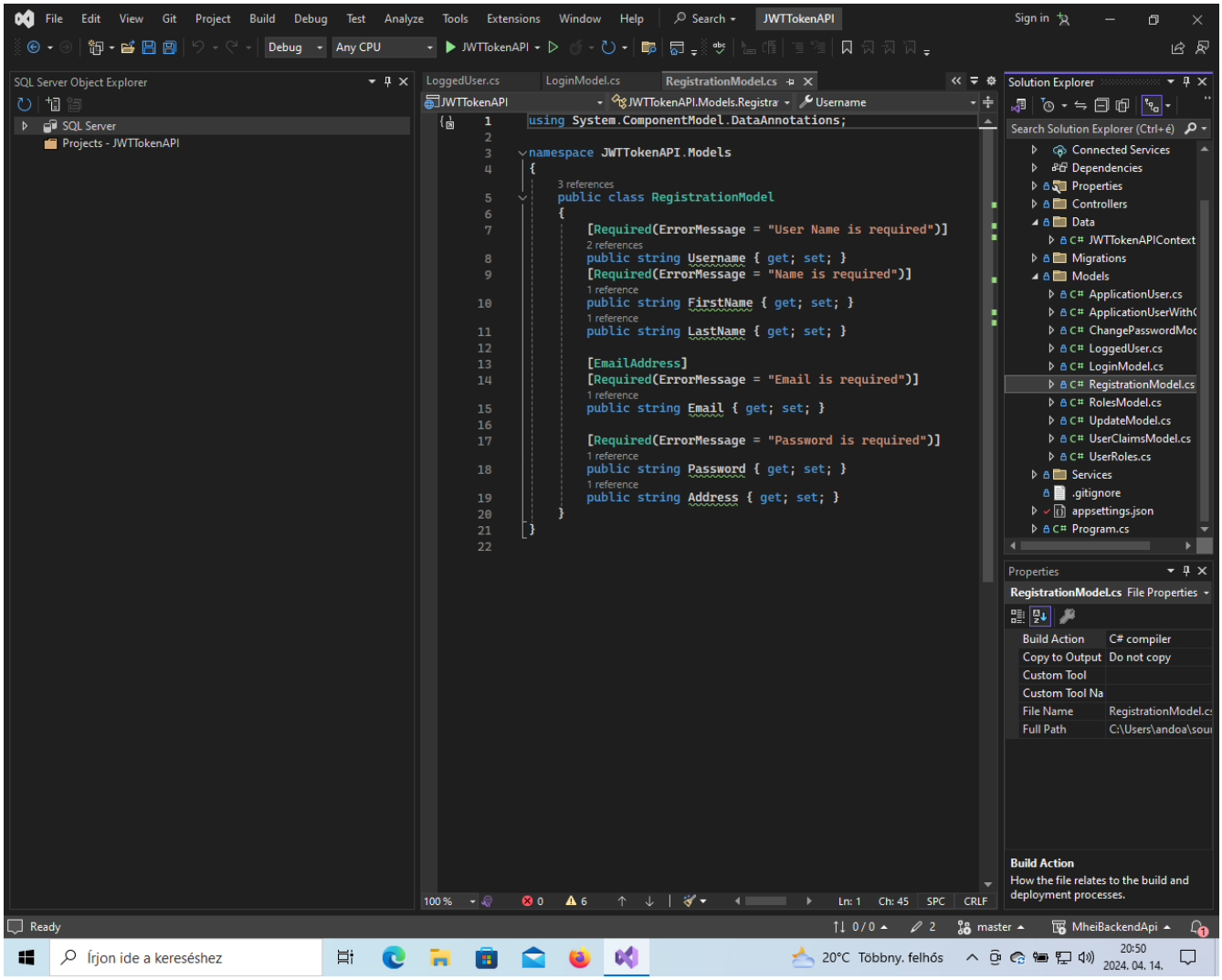
Mindkét tulajdonság megjelölve a Required attribútummal, ami azt jelenti, hogy mindkettő kötelezően kitöltendő. Ha valamelyiket nem adják meg, a megadott hibaüzenet jelenik meg. A System.ComponentModel.DataAnnotations névtérben található attribútumokat használja az adatok validálásához.
Ezek után létrehozzuk a Registration nevű modellt ami több tulajdonságot is tartalmazhat mint például Username, FirstName, LastName, Email, Password és Address. Ezek közül néhány attribútumot is tartalmaz:

- Username, FirstName, Email és Password tulajdonságokat megjelöli a Required attribútummal, ami azt jelenti, hogy ezek kitöltése kötelező.
- Az Email tulajdonsághoz az EmailAddress attribútumot is hozzárendeli, ami ellenőrzi, hogy a megadott érték valóban e-mail cím-e.

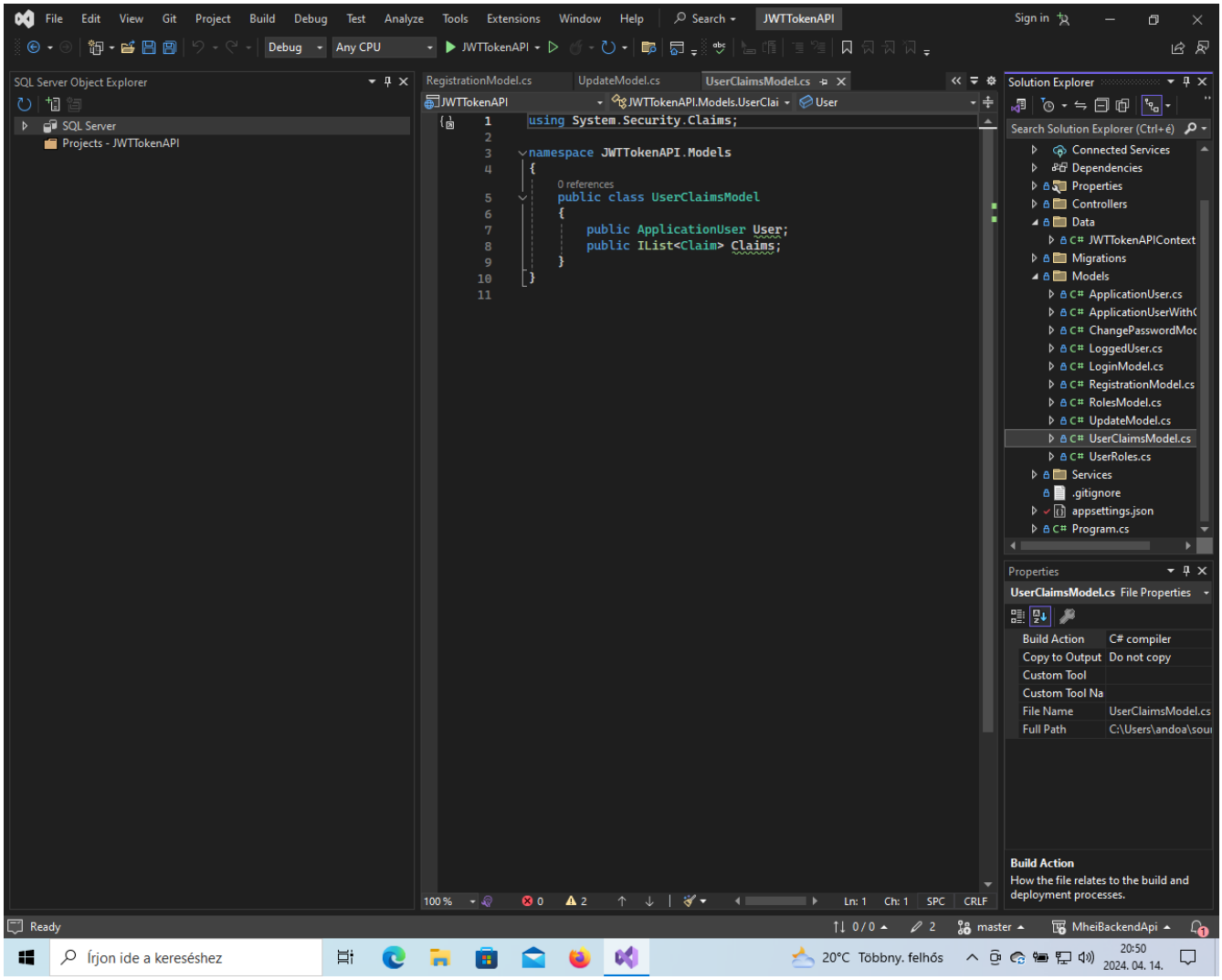


• Minden attribútumhoz hibaüzeneteket is rendel, amelyek megjelennek, ha az adott tulajdonságok nem megfelelően vannak kitöltve.
Létrehozzuk a Roles modellt ami szintén két tulajdonságot tartalmaz Id, amely egy karakterláncot tárol, és Roles, amely egy karakterlánc tömböt tárol. Ez az osztály arra szolgálhat, hogy egy felhasználóhoz tartozó szerepköröket tárolja.

A következő lépés hogy létrehozzuk az Update nevű modellt ami több tulajdonságot tartalmaz, mint például Id, Username, FirstName, LastName, Email és Address. Ezek a tulajdonságok valószínűleg egy felhasználói profil frissítését szolgálják a webalkalmazásban.



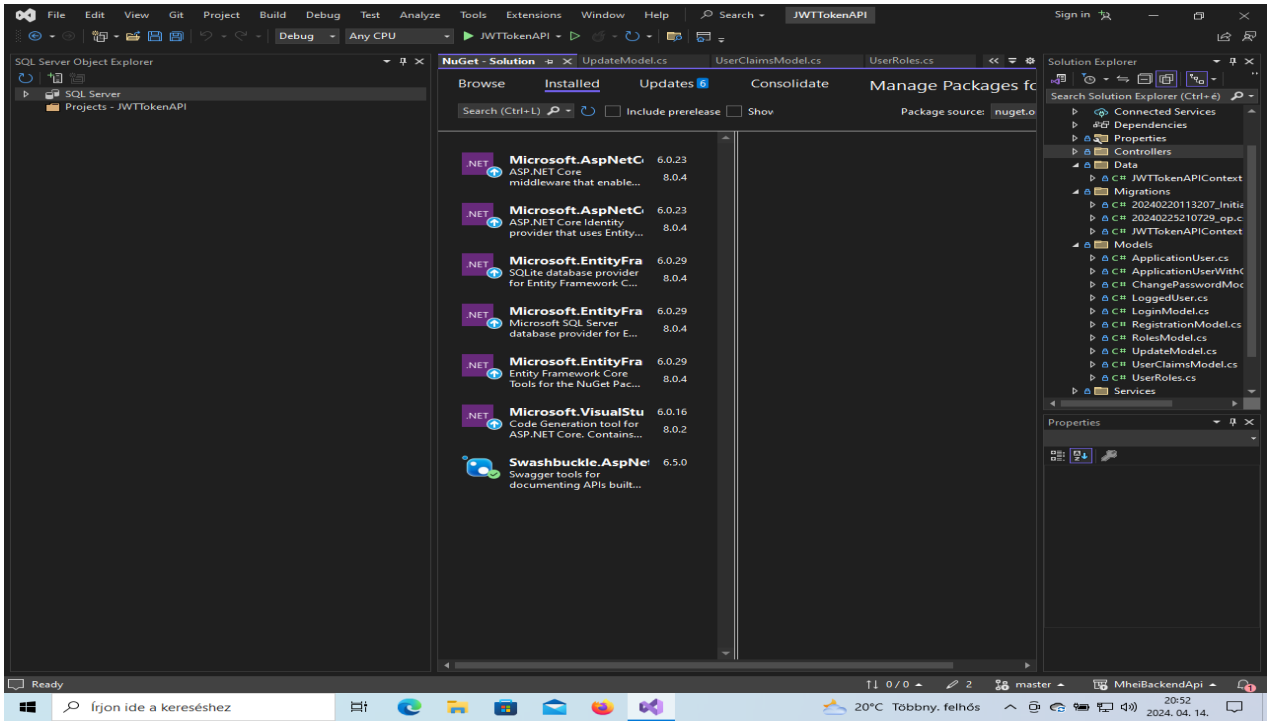
A Következő lépés hogy létrehozzuk a UserClaims nevű modellt ami megint csak két tulajdonságot tartalmaz User, amely egy ApplicationUser objektumot tárol, és Claims, amely egy Claim objektumokat tartalmazó listát tárol.



Ez az osztály a felhasználóhoz kapcsolódó jogosultságokat (claims) tárolja, amelyeket a felhasználó azonosításakor kapott tokennel együtt használhatnak az alkalmazásban. A System.Security.Claims névtérben található objektumokat használja az azonosítási és hitelesítési adatok kezeléséhez.

A következő lépés hogy létrehozzuk a UserRoles nevű modellt ami négy konstans értéket tartalmaz: Admin, User, SAdmin és AdministratorOrSuperAdministrator. Ezek a konstansok a felhasználók különböző szerepköreit reprezentálják az alkalmazásban.

A UserRoles osztály lehetővé teszi az alkalmazás számára, hogy könnyen hivatkozhasson ezekre a szerepkörökre, például az engedélyek kezelése vagy a jogosultságok ellenőrzése során. Utána létrehozol a Data mappában létrehozol egy JwtTokenApiContext.cs nevű fájlt.



Ami egy adatbázis kontextus osztályt definiál az Entity Framework Core használatával. Az osztály neve JWTTokenAPIContext, és az IdentityDbContext<ApplicationUser> osztálytól származik. Ez azt jelenti, hogy az alkalmazás felhasználó kezelését az Identity keretrendszerrel valósítja meg.
Az osztálynak egy konstruktora van, amely megkapja az adatbázis kapcsolódási beállításait (DbContextOptions<JWTTokenAPIContext>), majd ezeket továbbadja az ős osztálynak.
Megnyitod a toolst ott kiválasztod a Nuget Package Managert azon belül kiválasztod először a Manage Nuget Package for Solutiont és a Browseba a hozzá szükséges bővítményeket letöltöd. Kell a Microsoft AspNetCore.Authetication.JwtBearer, Microsoft AspNetCore.Identity.EntityFrameworkCore, Microsoft .EntityFrameworkCore.SqlServer, Microsoft .EntityFrameworkCore.Tools, Microsoft.VisualStudio.Web.CodeGeneration.Design, Microsoft.EntityFrameworkCore.Sqlite, Swashbuckle.AspNetCore és letöltöd a hozzá szüksége verziót pl 6.0.0.
A Microsoft.AspNetCore.Authentication.JwtBearer névtér és a hozzá tartozó csomag a JWT (JSON Web Token) alapú hitelesítési rendszer implementációját biztosítja az ASP.NET Core alkalmazások számára.

Ez a csomag lehetővé teszi az alkalmazások számára, hogy JWT-k alapján hitelesítsd az ügyfeleket. A JWT egy nyílt szabványú (RFC 7519) token formátum, amelyet széles körben használnak a webes alkalmazásokban a felhasználói hitelesítéshez és az állapotmentes autentikációhoz.

Az Microsoft.AspNetCore.Authentication.JwtBearer csomag lehetővé teszi az alkalmazások számára, hogy a következőket végezzék el:

JWT-k elfogadása: Az alkalmazás elfogadhatja és érvényesítheti az érkező JWT-kat, hogy meghatározza, hogy a kérő fél jogosult-e az adott erőforráshoz vagy szolgáltatáshoz.

Token validáció: Az alkalmazás ellenőrizheti a JWT-kat a kibocsátás helyessége, a token érvényessége és az aláírás ellenőrzése alapján.

Felhasználó azonosítása: Az alkalmazás azonosíthatja a JWT-k alapján azokat a felhasználókat vagy szolgáltatásokat, amelyek kérdéseket intéznek.

Biztonsági paraméterek beállítása: Az alkalmazás beállíthatja a JWT validálásához szükséges biztonsági paramétereket, például az érvényességi időtartamot, az aláírás ellenőrzését, az érvényességet meghatározó kiadó és céladatokat stb.

Az Microsoft.AspNetCore.Authentication.JwtBearer névtér tartalmazza azokat az osztályokat és interfészeket, amelyek lehetővé teszik az alkalmazások számára a JWT alapú hitelesítés integrálását az ASP.NET Core infrastruktúrájába.

Microsoft.AspNetCore.Identity.EntityFrameworkCore névtér a .NET Core identitáskezelő rendszerének Entity Framework Core (EF Core) adatbázis tárolását és műveleteit biztosító része. Ez a névtér a IdentityDbContext<TUser> osztályt tartalmazza, amely az IdentityDbContext osztály leszármazottja, és együttműködik az Entity Framework Core-rel az alkalmazás felhasználói adatok tárolásához.

Ez a névtér tartalmazza azokat az osztályokat és interfészeket, amelyek segítségével az ASP.NET Core alkalmazások könnyen kezelhetik a felhasználói hitelesítés és azonosítás funkcióit, mint például a felhasználók regisztrálása, bejelentkezése, jelszó kezelése stb.

A Microsoft.EntityFrameworkCore.SqlServer névtér és a hozzá tartozó csomag az Entity Framework Core SQL Server specifikus részét tartalmazza. Ez a névtér és csomag teszi lehetővé az Entity Framework Core (EF Core) használatát SQL Server adatbázisokhoz való kapcsolódásra, lekérdezésekre és adatmanipulációra.

Ez a csomag tartalmazza azokat az osztályokat és interfészeket, amelyek szükségesek az EF Core használatához SQL Server adatbázissal:

SqlServerDbContextOptionsExtensions: Ez az osztály tartalmaz kiterjesztési metódusokat a DbContextOptionsBuilder osztályhoz, amelyek

lehetővé teszik a SQL Server specifikus beállítások, például a kapcsolódási karakterlánc megadását.

SqlServerDatabaseProvider: Ez az osztály a SQL Server adatbázis szolgáltatót implementálja az EF Core számára, amely lehetővé teszi a kapcsolódást és az adatbázis műveletek végrehajtását a SQL Server adatbázisban.

SqlServerMigrationBuilder: Ez az osztály az EF Core migrációk SQL Server adatbázisokra történő alkalmazásához szükséges.

SqlServerQuerySqlGenerator: Ez az osztály a LINQ lekérdezések SQL Server specifikus SQL generálás-áért felelős.

SqlServerConnection: Ez az osztály az SQL Server adatbázishoz történő kapcsolódás ért felelős, és lehetővé teszi az SQL Server adatbázis kapcsolati információinak megadását és kezelését.

Ezek az osztályok és interfészek az EF Core számára biztosítják a szükséges funkcionalitást a SQL Server adatbázisokkal való interakcióhoz. A Microsoft.EntityFrameworkCore.SqlServer csomag segítségével az EF Core könnyen és hatékonyan integrálható SQL Server adatbázisokkal való munkára az ASP.NET Core alkalmazásokban.

Néhány fontos osztály és interfész a következők:

1. IdentityDbContext<TUser>: Az Entity Framework Core adatbázis kontextus osztálya, amely a felhasználói adatok tárolását végzi.
2. IdentityUser: Az alapértelmezett felhasználó osztály, amely tartalmazza a felhasználóhoz kapcsolódó alapvető információkat, mint például a felhasználónév és a jelszó.
3. IdentityRole: Az alapértelmezett szerep osztály, amely azonosítja a felhasználók csoportjait.
4. UserManager<TUser>: Az osztály, amely lehetővé teszi a felhasználók kezelését, beleértve a felhasználók létrehozását, szerkesztését, törlését stb.
5. SignInManager<TUser>: Az osztály, amely lehetővé teszi a felhasználók bejelentkezését és kijelentkezését az alkalmazásban.

Ezek az osztályok és interfészek lehetővé teszik az alkalmazások számára, hogy könnyen integrálják az ASP.NET Core identitáskezelő rendszerét az EF Core adatbáziskezelővel, és egyszerűen kezeljék a felhasználói hitelesítés és azonosítás funkcióit az alkalmazásban.

A Microsoft.EntityFrameworkCore.Tools névtér és a hozzá tartozó csomag az Entity Framework Core eszközeit és parancssori segédprogramjai-t tartalmazza. Ezek az eszközök lehetővé teszik az EF Core migrációk kezelését, adatbázis sémájának létrehozását és frissítését, valamint az adatbázisokkal való interakciót.

Ezek az eszközök segítségével fejlesztők könnyedén kezelhetik az adatbázis sémájának változásait, migráció-it, valamint hozzáférhetnek az adatbázisba integrált eszközökhöz, például az adatbázis szolgáltató-specifikus funkciókhoz.

Néhány fontos eszköz és parancs, amelyeket a Microsoft.EntityFrameworkCore.Tools csomag biztosít:

Migration Commands: Ezek a parancsok lehetővé teszik az adatbázis-migrációk létrehozását (add-migration), alkalmazását (update-database), visszavonását (remove-migration), valamint a migrációk státuszának ellenőrzését (list-migrations).

DbContext Scaffold Command: Ez a parancs lehetővé teszi egy meglévő adatbázis sémájának alapján egy DbContext osztály automatikus generálását, így az EF Core könnyen használható az adott adatbázisban.

Database Update Command: Ez a parancs végrehajtja a migrációkat az adatbázison, amelyek a kódban lévő változásokat alkalmazzák az adatbázissémára.

DbContext Info Command: Ez a parancs információkat nyújt a projektben található DbContext-ekről és a hozzájuk tartozó adatbázisokról.

Reverse Engineering: A dotnet ef dbcontext scaffold parancs segítségével fordítva lehet generálni egy DbContext osztályt egy meglévő adatbázisból.

Ezek az eszközök segítenek a fejlesztőknek hatékonyan kezelni az adatbázissal kapcsolatos feladatokat az EF Core segítségével, és lehetővé teszik az adatbázissal való könnyű és hatékony kommunikációt a fejlesztés során.

A Microsoft.VisualStudio.Web.CodeGeneration.Design névtér és a hozzá tartozó csomag az ASP.NET Core alkalmazásokhoz használt kódminták, scaffold-olás és generálási eszközöket tartalmazza. Ezek az eszközök lehetővé teszik a fejlesztők számára, hogy gyorsan generáljanak kódot az ASP.NET Core alkalmazásaikban, például MVC kontrollereket, Razor oldalakat, adatmodell osztályokat stb.

A Microsoft.VisualStudio.Web.CodeGeneration.Design csomag többek között az alábbiakat tartalmazza:
egy függvényt használ az ellenőrzés végrehajtására. Ebben az esetben a CanActivateFn típusú függvény egy függvényt vár paraméterként, ami a guard logikáját tartalmazza.

A guard célja, hogy ellenőrizze, hogy a felhasználó rendelkezik-e "SAdmin" jogosultsággal. Ehhez az AuthService szolgáltatást használja, amelynek SadminSub tulajdonsága egy BehaviorSubject, ami jelzi, hogy a felhasználó rendelkezik-e "SAdmin" jogosultsággal.

A sadminGuard függvényben egy console.log parancs található azzal a céllal, hogy nyomon kövesse a guard működését. Ezután visszatér a SadminSub értékével, amely egy BehaviorSubject, ami a guard döntését határozza meg: ha true, akkor a guard engedi a navigációt, ha false, akkor pedig megakadályozza azt.

Controller Scaffolding: Lehetővé teszi a fejlesztők számára, hogy gyorsan generáljunk MVC kontrollereket az alkalmazásokhoz az adott adatmodell osztályok alapján. Ez a funkcionalitás a Visual Studio-ban kódgenerálási eszközként jelenik meg.

Razor Page Scaffolding: Lehetővé teszi a Razor oldalak gyors generálását és frissítését az alkalmazás különböző részeihez.

Identity Scaffolding: Az ASP.NET Core Identity funkcióinak gyors implementálását teszi lehetővé, például felhasználókezelés, regisztráció, bejelentkezés stb.

DbContext Scaffold: Ez a parancs lehetővé teszi egy meglévő adatbázis sémájának alapján egy DbContext osztály automatikus generálását.

Service Class Scaffolding: Lehetővé teszi az alkalmazás szolgáltatási rétegének gyors generálását, például az üzleti logika és az adatelérési rétegek.

Dependency Injection Scaffolding: Ez a funkció lehetővé teszi az alkalmazás szolgáltatási konfigurációjának gyors generálását és regisztrálását.

Ezek az eszközök és funkciók jelentősen felgyorsíthatják az ASP.NET Core alkalmazások fejlesztését, és lehetővé teszik a fejlesztők számára, hogy gyorsabban hozzanak létre és frissítsenek kódot az alkalmazásban.

A Microsoft.EntityFrameworkCore.Sqlite névtér és a hozzá tartozó csomag az Entity Framework Core (EF Core) SQLite adatbázissal való kapcsolódását és műveleteit biztosítja. Ez a csomag lehetővé teszi az EF Core használatát az

SQLite adatbázissal való interakcióhoz, ami egy könnyű, általános célú, helyi adatbázis motor.

Az Microsoft.EntityFrameworkCore.Sqlite csomag tartalmazza azokat az osztályokat és interfészeket, amelyek szükségesek az EF Core számára az SQLite adatbázisokkal való kommunikációhoz. Ezek az osztályok és interfészek közé tartoznak:

SqliteDbContextOptionsExtensions: Ez az osztály tartalmaz kiterjesztési metódusokat a DbContextOptionsBuilder osztályhoz, amelyek lehetővé teszik a SQLite specifikus beállítások, például a kapcsolódási karakterlánc megadását.

SqliteDatabaseProvider: Ez az osztály az SQLite adatbázis szolgáltatót implementálja az EF Core számára, amely lehetővé teszi a kapcsolódást és az adatbázis műveletek végrehajtását az SQLite adatbázisban.

SqliteDatabaseCreator: Ez az osztály felelős az SQLite adatbázis létrehozásáért, ellenőrzéséért és inicializálását.

SqliteMigrationSqlGenerator: Ez az osztály felelős a migrációk SQL generálásáért az SQLite adatbázisokra.

SqliteConnection: Ez az osztály az SQLite adatbázishoz történő kapcsolódás ért felelős, és lehetővé teszi az SQLite adatbázis kapcsolati információinak megadását és kezelését.

Ezek az osztályok és interfészek biztosítják az EF Core számára a szükséges funkcionalitást az SQLite adatbázisokkal való kommunikációhoz. Az Microsoft.EntityFrameworkCore.Sqlite csomag segítségével az EF Core könnyen és hatékonyan integrálható az SQLite adatbázisokkal való munkára az ASP.NET Core alkalmazásokban.

Az Swashbuckle.AspNetCore csomag egy .NET könyvtár, amely segít az ASP.NET Core alkalmazásoknak Swagger (OpenAPI Specification) dokumentáció generálásában és az API-k felfedezésében.

Ez a csomag a Swashbuckle könyvtár ASP.NET Core verzióját tartalmazza. A Swashbuckle egy nyílt forráskódú könyvtár, amely lehetővé teszi az API dokumentáció automatikus generálását az alkalmazás forráskódjából. Az OpenAPI Specification (korábban Swagger Specification) egy szabványos, platformfüggetlen módszer az API-k leírására és dokumentálására. Az OpenAPI Specification segítségével az API-k leírása egyszerűen történhet JSON formátumban.

Az Swashbuckle.AspNetCore lehetővé teszi a Swagger UI és a Swagger JSON generálását az ASP.NET Core alkalmazások számára. A Swagger UI egy felhasználóbarát, interaktív felület, amely lehetővé teszi az API felfedezését és kipróbálását a dokumentáció alapján. A Swagger JSON egy

OpenAPI Specification formátumban generált JSON fájl, amely részletes leírást tartalmaz az API-ról.

A Swashbuckle segítségével az API dokumentálása automatikus lehet, ami megkönnyíti az API-k karbantartását és az új fejlesztők számára való megértését. A Swagger UI használata pedig lehetővé teszi az API-k tesztelését és felfedezését egy egyszerű és interaktív felületen keresztül.

S a Program.cs fileban beírjuk a AllowAnyHeader,AnyAllowMethod és AllowAnyOrigin sort
Ez a kód beállítja az alkalmazásnak a CORS (Cross-Origin Resource Sharing) engedélyezését. A CORS egy biztonsági mechanizmus, amely szabályozza, hogy egy webalkalmazás milyen erőforrásokhoz férhet hozzá más eredetű kérdésekből.

A UseCors metódus hozzáad egy köztes szoftverkomponenst az alkalmazás HTTP kérés pipeline-jához. Ez a komponens megvizsgálja az érkező kéréseket, és hozzárendeli hozzájuk az engedélyezett CORS politikát.

A megadott konfiguráció engedélyezi az összes eredetből érkező kérést (AllowAnyOrigin), az összes HTTP módszert (AllowAnyMethod), valamint az összes fejlécet (AllowAnyHeader). Ezáltal az alkalmazás bármely forrásból fogadhat kéréseket és válaszolhat rájuk.

Ez a rész azzal a céllal van, hogy lehetővé tegye az alkalmazás számára, hogy kéréseket fogadjon el más eredetű forrásokból, például más domainekről vagy portokról. Fontos azonban megjegyezni, hogy a CORS engedélyezésekor ügyelni kell a biztonságra, hogy ne tegyük ki az alkalmazást a különböző web biztonsági támadásoknak.

S az appsettings.jon be írjuk ezt a sort "JWTTokenAPIContext": "Server=(localdb)\\mssqllocaldb;Database=JWTTokenAPI.DataSzoftII1;Trusted_Connection=True;MultipleActiveResultSets=true",

Ami egy adatbázis kapcsolódási konfigurációt tartalmaz a JWTTokenAPIContext adatbázis kontextus használatára. Ez a kapcsolódási konfiguráció megmondja az Entity Framework Core-nak, hogy mely adatbázis szerverrel és adatbázissal kell kapcsolódni.

A kapcsolódási konfiguráció a következőket tartalmazza:

Server=(localdb)\\mssqllocaldb: Ez meghatározza az adatbázis szerver nevét és típusát. Az (localdb)\\mssqllocaldb az SQL Server Express LocalDB-t jelöli, amely egy könnyű verziója a SQL Szervernek, amely könnyen telepíthető és kezelhető.

Database=JWTTokenAPI.DataSzoftII1: Ez az adatbázis neve, amelyet az alkalmazás használni fog. Ebben az esetben az adatbázis neve JWTTokenAPI.DataSzoftII1.

Trusted_Connection=True: Ez azt jelzi, hogy az adatbázishoz való kapcsolódáshoz Windows hitelesítési információkat kell használni. Ez a beállítás azt jelenti, hogy az alkalmazás az aktuális Windows felhasználó hitelesítési adatait fogja használni az adatbázishoz való kapcsolódáshoz.

MultipleActiveResultSets=true: Ez a beállítás lehetővé teszi több aktív eredmény készlet kezelését az adatbázis kapcsolaton belül. Ez hasznos lehet, ha az alkalmazásnak több eredmény készletet kell kezelnie egy időben.

Ez a konfiguráció lehetővé teszi az alkalmazásnak, hogy csatlakozzon az adatbázishoz és használja azt a JWTTokenAPIContext adatbázis kontextusban.

Ezek után migrálnunk kell hogy létrehozza az adatbázist s a táblákat.Ezt először az Add-Migration paranccsal tehetjük meg ami után meg adjuk a migrációs file nevét. Ezután frissítjük az adatbázist a Update-Database paranccsal működni fog a Controller Létrehozás.
Controllert úgy hozzunk létre hogy Megadjuk a hozzá tartozó modell nevét és Hozzáadjuk a JwtTokenApiContext.cs fájlt és még a Controller nevét.

Utána a Controllersben Létrehozol AutheticatonControllert

• azon belül Létrehozol egy get logint ami fogadja és feldolgozza a bejelentkezési kérelmeket.Ellenőrzi a modell érvényességét majd meghívja az ’IAuthService’ Szolgáltatás ’Login’ metódusát az autentikáció végrehajtására.Ha sikeres a bejelentkezés,akkor visszaküldi az autentikált felhasználót az eredményben. azon belül Létrehozol egy get registert ami fogadja és feldolgozza a bejelentkezési kérelmeket.Ellenőrzi a modell érvényességét majd meghívja az ’IAuthService’ Szolgáltatás ’ Register’ metódusát az regisztráció végrehajtására.Ha sikeres a regisztráció,akkor visszaküldi az eredményt.
• Utána létrehozol egy UserControllert amiben létrehozod ezeket a műveleteket.

• A Get() metódus lekéri és visszaadja az összes felhasználót, amelyekhez az SAdmin vagy Admin szerepkör tartozik. Ehhez az Authorize attribútumot használja.

• A Get(string id) metódus lekéri és visszaadja a megadott azonosítójú felhasználót.
• A DeleteUser(string id) metódus törli a megadott azonosítójú felhasználót. Ehhez az SAdmin szerepkört kell rendelkeznie a hívó felhasználónak.
• Az Update(UpdateModel model) metódus frissíti a felhasználó adatait.
• A ChangePassword(ChangePasswordModel model, string id) metódus megváltoztatja a megadott azonosítójú felhasználó jelszavát.
• A ChangeMyPassword(ChangePasswordModel model) metódus megváltoztatja az aktuális felhasználó jelszavát.
Ezek a műveletek különböző felhasználói műveleteket implementálnak, mint például a regisztráció, bejelentkezés, jelszóváltoztatás, stb. Az Authorize attribútumok segítségével biztosítja, hogy csak az engedélyezett felhasználók férjenek hozzá ezekhez a műveletekhez.

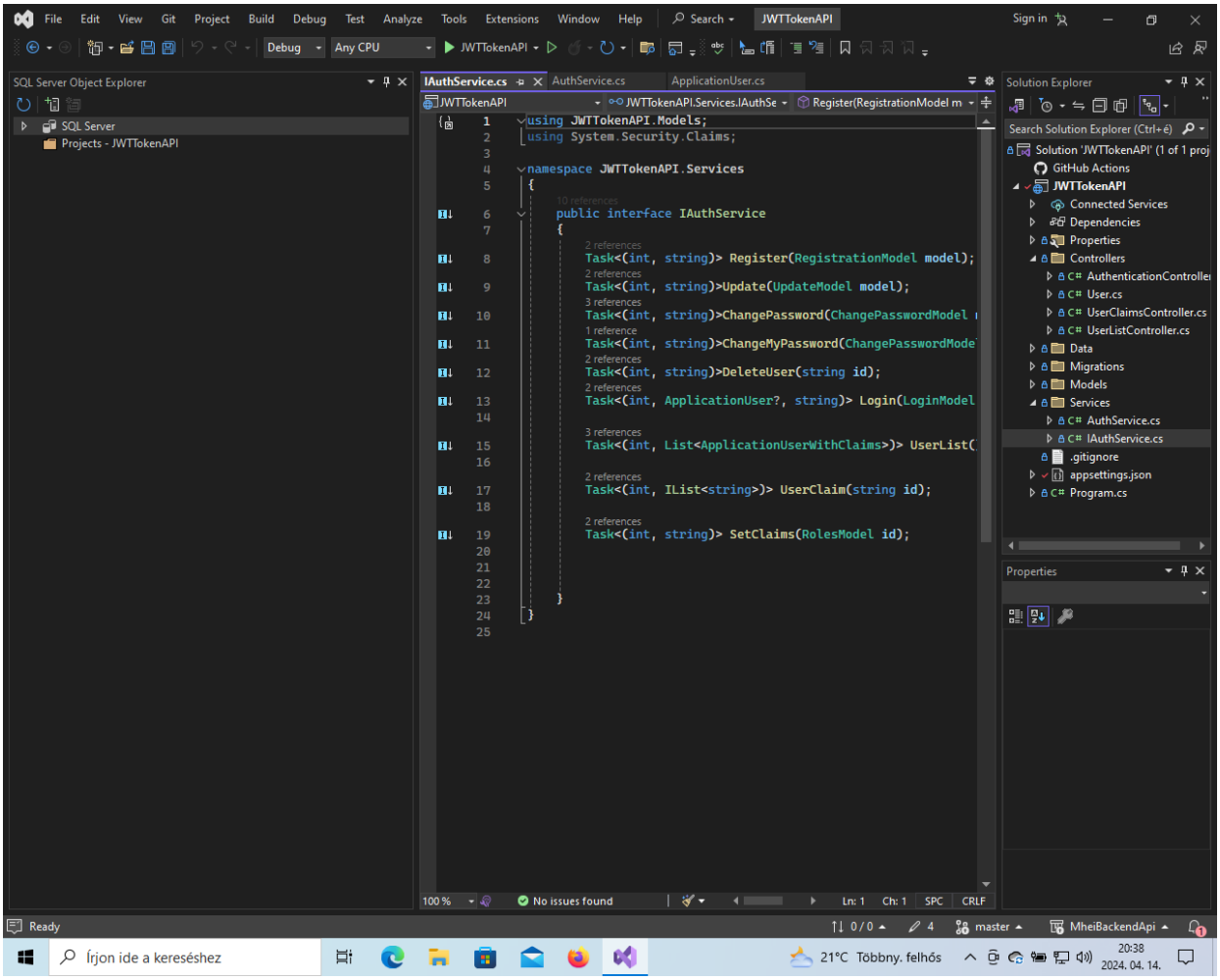
Utána létrehozol egy UserClaimsControllert s elvégzed ezeket a műveleteket.

Get(string id) metódus lekéri és visszaadja a megadott azonosítójú felhasználó jogosultságait. Ehhez az Authorize attribútumot használja, hogy csak a hitelesített felhasználók férjenek hozzá
• A Post(RolesModel id) metódus felhasználói jogosultságokat állít be a megadott felhasználóhoz. Ehhez az SAdmin szerepkört kell rendelkeznie a hívó felhasználónak.
Ezek a műveletek lehetővé teszik a felhasználói jogosultságok lekérését és beállítását az alkalmazáson belül. Az Authorize attribútumok biztosítják, hogy csak a megfelelő hitelesített felhasználók érjék el ezeket a műveleteket.
Utána létrehozol egy UserListControllert és ezeket elvégzed a felhasználói lista láthatóságát.

• A Get() metódus lekéri és visszaadja az összes felhasználót, amelyeket az IAuthService szolgáltatás UserList metódusa visszaad. A felhasználók listáját és azok adatait tartalmazza az eredmény.
Ez a művelet lehetővé teszi az összes felhasználó listázását az alkalmazáson belül, és azok adatainak visszaadását. Az Authorize attribútumok jelenleg ki vannak kommentelve, így a művelet nyilvános, és nem igényel hitelesítést a híváshoz.
Ezzel befejeződött a Controllerben a beállítások és most össze kapcsoljuk a modellekkel .

Ezzel már nincs több modell beállitás most jön a Services mappa. Először létrehozzuk az AuthService nevű szervizt.

Az osztály több metódust tartalmaz, amelyek különböző felhasználói műveleteket valósítanak meg, például felhasználói lista lekérdezése, felhasználói jogok lekérdezése, jogosultságok beállítása, felhasználó törlése, jelszóváltoztatás, felhasználói adatok frissítése, regisztráció és bejelentkezés. Az osztály használ néhány más névteret is, például JWTTokenAPI.Models, Microsoft.AspNetCore.Identity, Microsoft.IdentityModel.Tokens, System.IdentityModel.Tokens.Jwt, System.Security.Claims, System.Text, Newtonsoft.Json és Microsoft.EntityFrameworkCore. Ezek a névterek az alkalmazás különböző részeihez kapcsolódó osztályokat és funkcionalitásokat tartalmaznak.
Ezek után létrehozzuk az IAuthService nevű servicet ami egy interfészt definiál.



Az interfész különböző metódusokat definiál, amelyek az autentikációs és autorizációs műveleteket végzik az alkalmazásban. Ezek a metódusok magukban foglalják a regisztrációt, adatok frissítését, jelszóváltoztatást, felhasználó törlését, bejelentkezést, felhasználói lista lekérdezését, felhasználói jogok lekérdezését és jogosultságok beállítását.
Az interfész definiálásra kerül a szükséges bemeneti paraméterekkel és visszatérési értékekkel együtt, melyeket az AuthService osztály implementálhat. Az interfész a JWTTokenAPI.Models névtérben található modell osztályokat (RegistrationModel, UpdateModel, ChangePasswordModel, LoginModel, RolesModel,

ApplicationUserWithClaims) használja, valamint a System.Security.Claims névtérben található ClaimsIdentity típust is importálja a jogosultságok kezeléséhez.

Ezek elvégzése Után Megint Migránunk kell Az Add-Migration Elso paranccsal.Ami létre a szükséges táblákat. Az első a migrációs fájl neve. S ez után jön Az Update-Database parancs ami frissíti az adatbázis és mindent. Ezek után Buildelünk egyet. Abuild menüpontnál kiválasztjuk BuildSolution vagy rebuild vagy a BuildJwtTokenApi ponttal. S utána lefuttatjuk az adatbázist ha nem fut le akkor töröljük a migrációs fájlokat és A View menüpontnál kiválasztjuk s azon belül SqlServerObjectExplorer menüpontot. S ott kinyitjuk az SqlServert azon belül lenyitjuk a localdb ott kiválasztjuk a Database menüpontot azon belül kitöröljük a JwtTokenApiData.SzoftII1 s töröljük adatbázis szervert. Ezután újra migrálunk s akkor már minden müködni fog.

Elérési Útvonalak:

POST/api/Authentication/login	GET/api/Events/{id}
POST/api/Authentication/register	POST/api/Events
	GET/api/Events
GET/api/user/userlist	PUT/api/Events/{id}
GET/api/user/{id}	DELETE/api/Events/{id}
DELETE/api/user/{id}	
	PUT/api/user/{id}
	PUT/api/user/changePassword/{id}
POST/api/userClaims	PUT/api/user/changemyPassword
GET/api/userClaims/{id}	GET/api/user/userlist

ChangeMyPasswordModel	{
	
	id* string
	minLength: 1
	currentPassword* string
	minLength: 1
	newPassword* string
	minLength: 1
	}

Event	{
	id integer($int32)
	userId string
	nullable: true
	name string
	nullable: true
	description string
	nullable: true
	link string
	nullable: true
	}

LoginModel	{
	username* string
	minLength: 1
	password* string
	minLength: 1
	}

RegistrationModel	{ username* string minLength: 1firstName* lastName string nullable: true email* string($email) minLength: 1

	password* string minLength: 1 address string nullable: true }

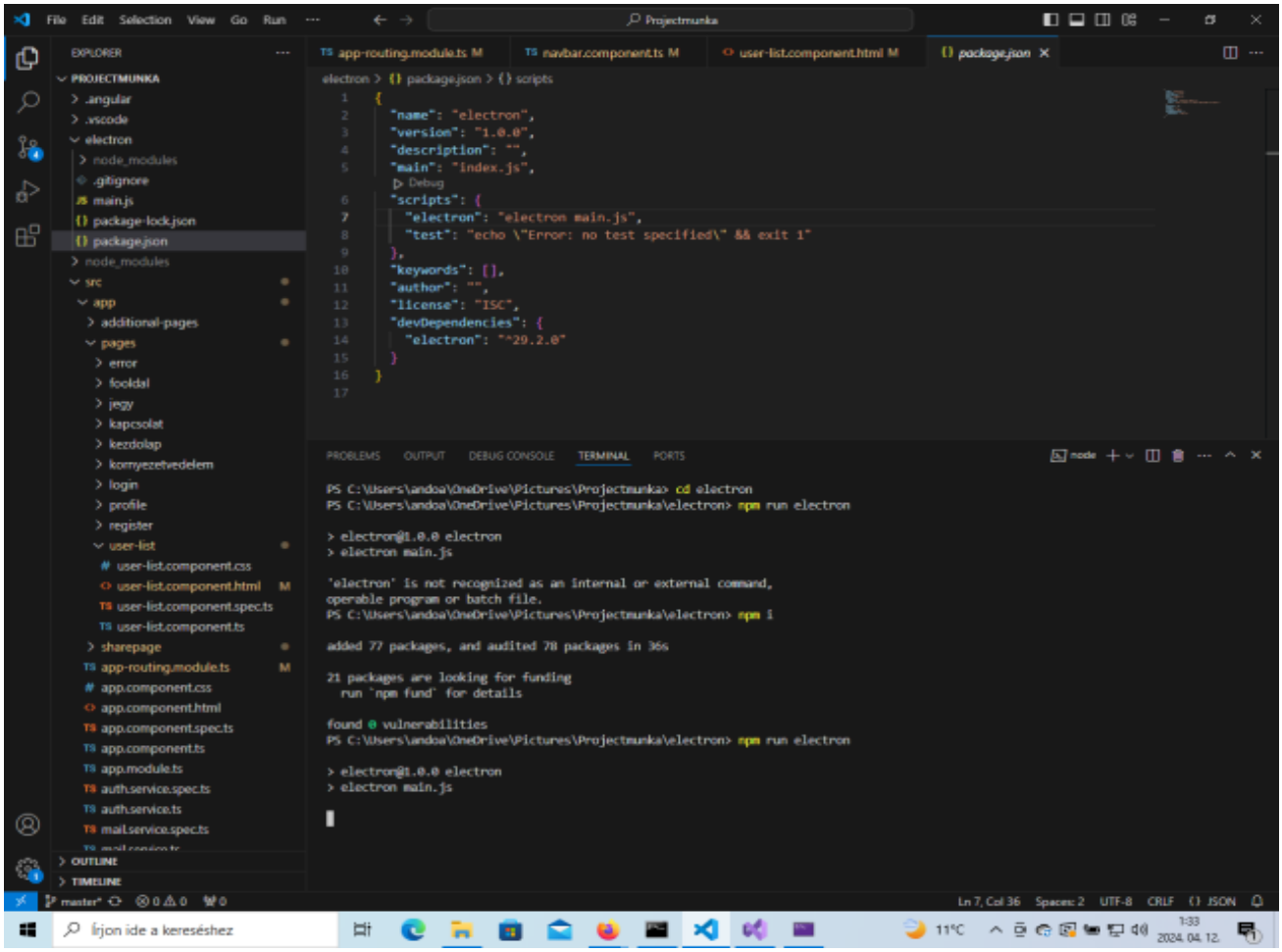
RolesModel	{
	id string
	nullable: true
	roles
	[
	nullable: true
	string]
	}

UpdateModel	{
	id string
	nullable: true
	username string
	nullable: true
	firstName string
	nullable: true
	lastName string
	nullable: true
	email string
	nullable: true
	address string
	nullable: true
	}

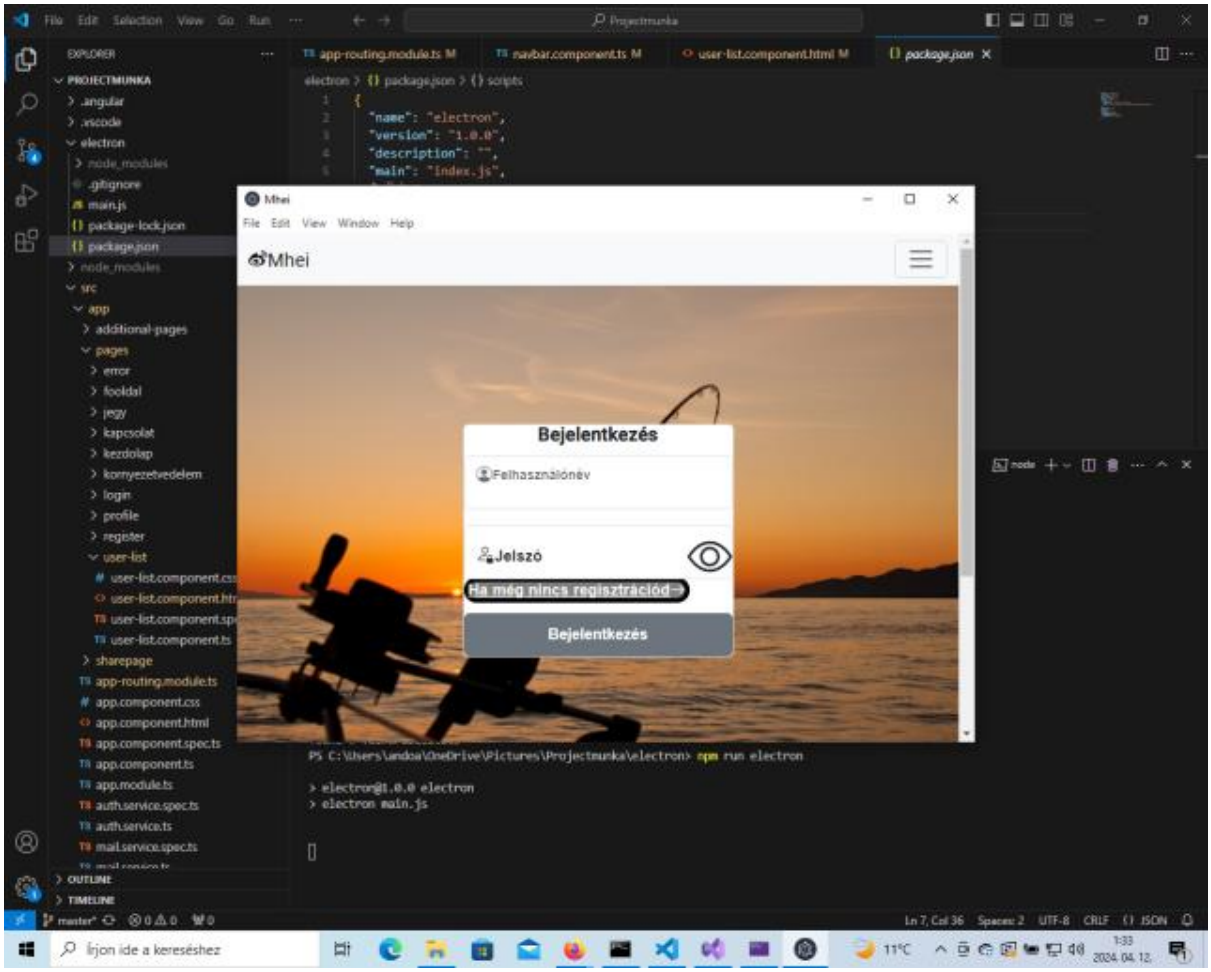
Electron

Megvalósítottuk a célt, hogy meglévő Angular projektünket Electron asztali alkalmazássá alakítsuk át. Ezt a törekvést az a vágy motiválta, hogy kiterjesszük alkalmazásunk hatókörét az asztali felhasználókra, miközben kihasználjuk a megszokott webes fejlesztési stacket.

A folyamat összefoglalása:
Projekt inicializálás: A folyamatot úgy indítottuk el, hogy navigáljunk az Angular project könyvtárunkba, és a terminálban végrehajtottuk a következő parancsokat:
npm init -y
npm install electron --save-dev



A main.js létrehozása: fájl létrehozása a projekt könyvtárunkban. Ez a fájl az Electron alkalmazásunk belépési pontjaként szolgál, meghatározva annak viselkedését és életciklus eseményeit.
A package.json szerkesztése: Ahhoz, hogy projektünket az Elektronhoz konfiguráljuk, elvégeztük a szükséges módosításokat a package.json fájlban. Konkrétan Electron-specifikus konfigurációkat adunk hozzá a scriptek szakaszhoz, hogy megkönnyítsük alkalmazásunk építését és végrehajtását.
Angular projekt építése: Az Angular-projektünkkel a helyére került, folytattuk annak építését az Angular CLI segítségével, hogy biztosítsuk, hogy készen áll az Electron integrációra. A projekt építéséhez a következő parancsot futtattuk le:
npm run build



Angular projekt kiszolgálása: A fejlesztési munkafolyamat részeként úgy döntöttünk, hogy az Angular projektet az Angular CLI segítségével szolgáljuk ki. Ez lehetővé tette számunkra, hogy teszteljük és iteráljuk az alkalmazásunk funkcionalitását, mielőtt integrálnánk azt az Electronba. Lefuttattuk a parancsot:
npm start
Electron alkalmazás futtatása: A következő paranccsal futtattuk az Electron alkalmazást:
npm run electron

A fenti lépésekkel sikeresen átalakítottuk a meglévő Angular projektünket egy Electron asztali alkalmazássá. Ez a folyamat lehetővé tette számunkra, hogy kihasználjuk az Electron erejét, hogy zökkenőmentes asztali felhasználói élményt nyújtsunk felhasználóinknak, miközben megőriztük a webes fejlesztési technológiák rugalmasságát és ismertségét.

Munka felosztása

A munkát előzetes hosszas megbeszélés alapján osztottuk fel, hármunk között. Amihez a Gantt Diagram nevű programot választotuk segítségül. Elosztás így nézett ki:

Andó Attila - Beckend / API,
Fekete Gergely Zoltán - Frontend

A dokumentációban lejjebb Gantt diagram formátumba próbálom bemutatni ,hogy hogy haladtunk a projectel. Meg lehet benne mutatni ,hogy ki mikor mit csinált a projectbe. Le lehet mérni ,hogy mennyi idő kellet a cél eléréséhez azaz a projekt leadásához illetve bemutatásához.

Gantt diagram értelmezése

Gantt diagram segítségével segítségével szemléltetjük, a projekt megvalósításának a lépéseit. Amit akkor használunk, ha a tervezési szakaszon túl lépünk, majd GanttProject-be felviszünk a tevet. A program segítségével, gyorsabban kevesebb időveszteséggel, elkészülhet a projekt terv.

A fenti ábrán látható, hogy a GanttProject-be, a projekt szempontjából minden fontos részletet fel lehet tüntetni, mint például, az idő szükségletet, lebontva napokra. Meg tudjuk nézni pontosan, hogy egyes tevékenységek lebontva mennyi anyagi ráfordítást igényel, mint a projekt mint a szakember akit megbíznak a feladattal.

A Projekt abban is segítséget tud nyújtani, ha esetleg az alptervbe eszközölnünk kell, egy vagy több változtatást, mert a program kiszámolja, hogy mekkora eltérés az a alaptervhez képest. Amit az alapterv beállításoknál találunk. Alaptervet a programon belül többet is elkészíthetünk. Az alaptervhez tudunk rendelni kritikus utat is, amit a program szintén számol, és ezek alapján megfelelő információkat kaphatunk vissza a projekt állapotával kapcsolatban.

Mheiprojektmunka 2024. ápr. 15.

Feladatok

2

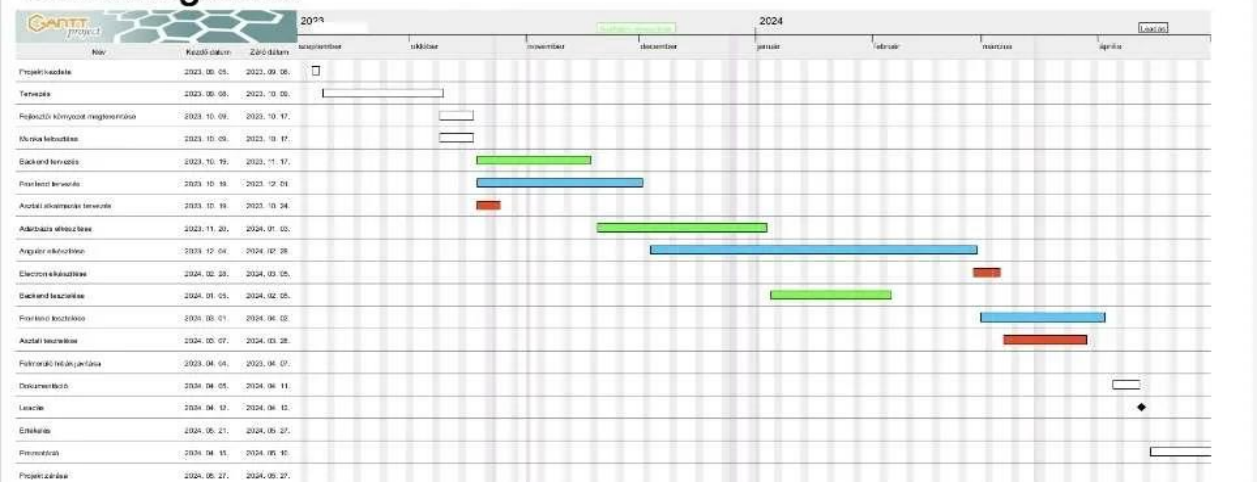
Név	Kezdő dátum	Záró dátum	Megjegyzés
Projekt kezdete	2023. 09. 05.	2023. 09. 06.	
Tervezés	2023. 09. 08.	2023. 10. 09.	
Fejlesztői környezet megteremtése	2023. 10. 09.	2023. 10. 17.	
Munka felosztása	2023. 10. 09.	2023. 10. 17.	
Backend tervezés	2023. 10. 19.	2023. 11. 17.	
Frontend tervezés	2023. 10. 19.	2023. 12. 01.	
Asztali alkalmazás tervezés	2023. 10. 19.	2023. 10. 24.	
Adatbázis elkészítése	2023. 11. 20.	2024. 01. 03.	
Angular elkészítése	2023. 12. 04.	2024. 02. 28.	
Electron elkészítése	2024. 02. 28.	2024. 03. 05.	
Backend tesztelése	2024. 01. 05.	2024. 02. 05.	
Frontend tesztelése	2024. 03. 01.	2024. 04. 02.	
Asztali tesztelése	2024. 03. 07.	2024. 03. 28.	
Felmerülő hibák javítása	2023. 04. 04.	2023. 04. 07.	
Dokumentáció	2024. 04. 05.	2024. 04. 11.	
Leadás	2024. 04. 12.	2024. 04. 12.	
Értékelés	2024. 05. 21.	2024. 05. 27.	
Prezentáció	2024. 04. 15.	2024. 05. 10.	
Projekt zárása	2024. 05. 27.	2024. 05. 27.	

Ezen az ábrán jól láthatóan fel vannak tüntetve .hogy melyik napokon milyen feladat lett végrehajtva. Természetesen az értékelés meg a projekt zárása még fiktív mivel még a jövőbe fog megtörténni. Ezeket szemléltetésképpen vettük fel. Lejebb ábrákkal is ábrázoljuk szintén Gantt diagramos formába az egyes munkafolyamatokat.

Mheiprojektmunka 2024. ápr. 15.

Gantt diagramm

4



Erőforrás diagram értelmezése

Ebben a diagramba tudjuk nyomon követni, hogy kik dolgoznak a projekten.

Nyomon lehet követni, a tevékenységek időszükségletét a diagramon.

Tevékenységek munka napokra lebontva, hogy egyes tevékenység mennyi időt vesz igénybe:

Tevékenység hozzárendelése:

Itt írom le részletesen, hogy melyik személy melyik tevékenységnél dolgozik, mennyi napot.

Fekete Gergely - Frontend, a munkalapon kék színnel feltüntetett területen dolgozik.

Andó Attila - Beckend ,a munkalapon zöld színnel feltüntetett területen dolgozik.

Illetve van a fehér színnel eltüntetett rész ami a projekt közös része, mint pl. a dokumentáció, hogy mindenki a saját részén dolgozik de mint itt megosztott dokumentumba. Igy bele tudunk nyúlni egymás dolgaiba. Látjuk ki hogy halad ha esetleg elakad akkor komolyabb megbeszélés nélkül tudunk javítani.

Frontend

Az inicalizálása projekt létrehozás, amit úgy hozzunk létre hogy kiadjuk az ng new parancsot s utána írjuk a projekt nevét. Kiválasztjuk hogy css file-ok legyenek a html-ben. Utána be lépünk cd-vel a projektbe.

Componensek:

src	app
additional-pages	additional fish fish shop event versenyek tickets
pages	login profile register userlist kapcsolat kornyezetvedelem kezdolap fooldal error jegy
share pages	footer navbar
guards	sadmin
service	auth mail

Authservice: Ez biztosítja az elérést a backendnek.

1. Importálások: Az Angular HttpClient-et használva kommunikál a szerverrel HTTP kérések révén, és az RxJS BehaviorSubject és Subject osztályokat használja az adatok állapotának kezelésére.
2. Konstruktor: Injektálja az HttpClient-et, amely lehetővé teszi a HTTP kérések küldését a szerver felé.
3. Metódusok:
    - logout(): Kijelentkezteti a felhasználót, törli a tokent és a felhasználói adatokat.
    - getCurrentUser(): Visszaadja a jelenlegi felhasználó BehaviorSubject-jét.
    - getUser(id): Lekéri egy felhasználó adatait az azonosító alapján.
    - getUsers(): Lekéri az összes felhasználó adatait.
    - getClaims(id): Lekéri egy felhasználó jogosultságait az azonosító alapján.
    - setClaims(id, claims): Beállítja egy felhasználó jogosultságait.
    - register(user): Regisztrál egy új felhasználót.
    - update(user): Frissíti egy felhasználó adatait.
    - delete(user): Törli egy felhasználót.
    - eventdelete(event): Törli egy eseményt.
    - getEvents(): Lekéri az összes eseményt.
    - getEvent(id): Lekéri egy esemény adatait az azonosító alapján.
    - putEvent(event): Frissíti egy esemény adatait.
    - eventadd(event): Hozzáad egy új eseményt.
    - changePassword(newPassword): Megváltoztatja a jelszót.
    - changeMyPassword(newPassword): Megváltoztatja a saját jelszót.
    - login(user): Bejelentkezteti a felhasználót.

Ezek a metódusok a felhasználói műveleteket kezelik, és kommunikálnak a szerverrel az Angular HttpClient segítségével.

Sadmin Guardok:

egy függvényt használ az ellenőrzés végrehajtására. Ebben az esetben a CanActivateFn típusú függvény egy függvényt vár paraméterként, ami a guard logikáját tartalmazza.

A guard célja, hogy ellenőrizze, hogy a felhasználó rendelkezik-e "SAdmin" jogosultsággal. Ehhez az AuthService szolgáltatást használja, amelynek SadminSub tulajdonsága egy BehaviorSubject, ami jelzi, hogy a felhasználó rendelkezik-e "SAdmin" jogosultsággal.

A sadminGuard függvényben egy console.log parancs található azzal a céllal, hogy nyomon kövesse a guard működését. Ezután visszatér a SadminSub értékével, amely egy BehaviorSubject, ami a guard döntését határozza meg: ha true, akkor a guard engedi a navigációt, ha false, akkor pedig megakadályozza azt.

Login:

Az ngModel direktívák használatával két bemeneti mező (username és password) kapcsolódik a loginModel adattaghoz, ami valószínűleg az űrlapmodellt jelenti.

A translate csővezetéket használják a szövegek fordításához, amelyek valószínűleg a title3, usernamePlaceholder, passwordPlaceholder, registrationLink és loginButton kulcsokkal vannak definiálva a fordítási fájlban.

Az eyeicon osztályhoz tartozó szem ikonra kattintva a viewpass() függvény fut le, ami valószínűleg a jelszó megtekintésének vagy elrejtésének funkcióját valósítja meg.

Az "Észlelés" és "Elrejtés" funkció megvalósításához a visible változóval valószínűleg nyomon követik, hogy a jelszó mező éppen látható-e vagy sem.

A "Regisztráció" linkre kattintva a felhasználó a "register" útvonalra lesz átirányítva, valószínűleg az új felhasználói regisztráció oldalára.

A "Bejelentkezés" gombra kattintva a login() függvény fut le, ami valószínűleg az űrlap elküldését és a felhasználó bejelentkeztetését végzi.

Profile:

- Az ngModel direktívák használatával a bemeneti mezők kapcsolódnak a regModel adattaghoz, ami valószínűleg a regisztrációs vagy felhasználói adatok modelljét jelenti.
- A szövegek fordítását a translate csővezetékkel végzik, amelyek valószínűleg a title4, usernamePlaceholder2, firstNamePlaceholder, lastNamePlaceholder, emailPlaceholder, addressPlaceholder, updateButton, passwordPlaceholder2, passwordPlaceholder3, és changePasswordButton kulcsokkal vannak definiálva a fordítási fájlban.
- Az "Észlelés" és "Elrejtés" funkció megvalósításához a visible változóval valószínűleg nyomon követik, hogy a jelszó mező éppen látható-e vagy sem.
- A "Frissítés" gombra kattintva a update() függvény fut le, ami valószínűleg az űrlap elküldését és a felhasználó adatainak frissítését végzi.
- A "Jelszó módosítása" gombra kattintva a changeMyPassword() függvény fut le, ami valószínűleg a felhasználó jelszavának módosítását végzi.
- Minden jelszó beviteli mezőhöz tartozik egy "szem" ikon, amelyre kattintva a felhasználó megtekintheti vagy elrejtheti a jelszót, és ehhez használják a viewpass() függvényt.

Register:

- Az ngModel direktívák segítségével a bemeneti mezők kapcsolódnak a regModel adattaghoz, ami valószínűleg a regisztrációs adatok modelljét jelenti.
- A szövegek fordítását a translate csővezetékkel végzik, amelyek valószínűleg a title4, usernamePlaceholder2, firstNamePlaceholder, lastNamePlaceholder, emailPlaceholder, addressPlaceholder, passwordPlaceholder2, és register2 kulcsokkal vannak definiálva a fordítási fájlban.
- Az "Észlelés" és "Elrejtés" funkció megvalósításához a visible változóval valószínűleg nyomon követik, hogy a jelszó mező éppen látható-e vagy sem.
- A "Regisztráció" gombra kattintva a register() függvény fut le, ami valószínűleg az űrlap elküldését és a felhasználó regisztrációját végzi.

Ez az űrlap lehetővé teszi a felhasználók számára, hogy kitöltsék a regisztrációs adataikat és regisztráljanak az alkalmazásban. Az adatokat a regModel objektumban tárolják, és az űrlap elküldésekor az alkalmazás valószínűleg ezt az objektumot használja a regisztráció feldolgozására.

Userlist:

- Az *ngFor strukturális direktívával az alkalmazás a users tömb elemein iterál végig, és minden egyes felhasználóhoz egy táblazatsort (tr) jelenít meg.
- A táblázat fejlécében (thead) felsorolja a megjelenítendő oszlopok címeit.
- A táblázat sorai (tr) a felhasználók adatait jelenítik meg az oszlopokban (td).
- A felhasználók jogaikat (claims) egy checkbox formájában jeleníti meg, amelyeket az adott felhasználó rendelkezésére álló jogok alapján állít be vagy töröl. A jogokat dinamikusan betölti a jogok tömbből.
- A felhasználókhoz tartozó műveleteket is megjeleníti, például a "Profil" és "Törlés" gombokat, amelyeket a profile() és delete() függvényekkel kezel az alkalmazás. A szövegek fordítását a translate csővezetékkel végzi.

Ez az HTML kód tehát egy dinamikusan generált felhasználói lista megjelenítését valósítja meg, amely lehetővé teszi a felhasználók adatainak megjelenítését, jogosultságainak kezelését és műveletek végrehajtását az alkalmazásban.

Navbar:

- A navigációs sáv (<nav>) tartalmazza az alkalmazás logóját, navigációs linkeket az egyes oldalakra, valamint egy nyelvválasztó legördülő menüt.
- Az *ngIf strukturális direktívákkal vannak feltételek beállítva, hogy bizonyos navigációs elemek csak akkor jelenjenek meg, ha a felhasználó be van jelentkezve (currentUser értéke igaz), vagy ha a felhasználónak bizonyos jogosultságai vannak, például "SAdmin".
- Az egyes navigációs elemekhez a routerLink attribútumokon keresztül vannak hozzárendelve az útvonalak.
- A nyelvválasztó legördülő menü (<select>) lehetővé teszi a felhasználó számára, hogy válasszon a rendelkezésre álló nyelvek közül, és az ChangeLanguage() eseménykezelőt használják az új nyelv beállításához.
- Az ngx-spinner komponens egy betöltő ikont jelenít meg a felhasználói élmény javítása érdekében. A bdColor, size, color és type attribútumok segítségével testre szabható a betöltő ikon stílusa.
- A <p> elem tartalmazza a betöltő ikon alatt megjelenítendő szöveget, amelyet a fordítási fájlban található loadingSpinnerText kulcs segítségével fordítanak.

Ez a navigációs sáv és betöltő ikon segít a felhasználók számára könnyen navigálni az alkalmazás különböző részei között, valamint jelzi nekik, hogy a tartalom betöltése folyamatban van.