

ELTE | IK
INFORMATIKAI KAR

Média- és Oktatásinformatikai Tanszék

Autóverseny webapplikáció

Máriás Zsigmond
Külső óraadó (ELTE IK)

Nyíró Levente Gyula
Programtervező informatikus BSc

Budapest, 2023

Tartalom

1. Bevezetés	4
1.1 Témaválasztás.....	5
2. Felhasználói dokumentáció.....	7
2.1. A program általános specifikációja	7
2.2. Felhasznált módszerek.....	7
2.3. Gépigény.....	9
2.4. Oldalak.....	9
2.4.1. Bejelentkezés.....	9
2.4.2. Regisztráció	10
2.4.3. Seasons	11
2.4.4. Season.....	12
2.4.5. Statisztika	15
2.4.6. Beállítások.....	17
3. Fejlesztői dokumentáció	19
3.1. Adatbázis	19
3.1.1. Users.....	21
3.1.2. Seasons	21
3.1.3. Favorites	22
3.1.4. Permissions.....	22
3.1.5. Drivers	23
3.1.6. Teams	23
3.1.7. Races	23
3.1.8. Results	24
3.2. Szerveroldal.....	24
3.2.1. Driver	27
3.2.2. Favorite.....	29
3.2.3. Permission	30
3.2.4. Race	30
3.2.5. Result.....	31
3.2.6. Season.....	31
3.2.7. Team.....	34
3.2.8. User	35

3.3.	Kliensoldal	36
3.4.	Telepítés	39
3.5.	Tesztesetek	40
3.5.1.	Result osztály egységtesztje.....	41
3.5.2.	Result osztály integrációs tesztje	42
3.5.3.	További osztályok tesztjei	44
3.6.	Projektfejlesztési folyamatok	44
3.7.	Továbbfejlesztési lehetőségek.....	46
4.	Összegzés	48
	Forrásjegyzék.....	50
	Ábrajegyzék.....	51

1. Bevezetés

Hároméves programtervező informatikai képzésem utolsó félévében létre kellett hoznom egy szoftvert az általam választott témában. Mindennek a tervezési, illetve implementációs lépéseit kellett véghez vinni. Ezen dokumentáció keretei között részletesen meg lehet ismerkedni a témaválasztás okaival, a tervezésem során kitűzött célokkal, az adatbázis tervezésével, a fejlesztési lépésekkel, a munkafolyamattal, illetve a megvalósított program jellemzőivel. Bemutatásra kerül a felhasználói dokumentáció, amelyben a felület funkcióit tervezem bemutatni, utána a fejlesztői részre áttérve kiderül a program mögöttes tartalma minden egyes funkció implementációjával együtt.

Témakörként a full stack technológiára esett a választásom magában foglalva a szerver- és a kliensoldali programozást. A téma mindig is érdekelt az informatika keretein belül. Hatással volt rám az adatstruktúrák kialakítása, adatkezelések, frontend és a backend közötti kapcsolat, illetve a dizájn kialakítása. A jövőben is ezzel a témakörrel tervezek foglalkozni, így nem volt más választásom, minthogy egy ezzel kapcsolatos szoftvert valósítsak meg.

Egy bekezdés erejéig pillantsunk vissza az időben. A weboldalak szerves részeit képezték az internetnek annak elindulásával és a terjedésével. Amikor a lakosság számára elérhetővé vált az internet, akkor még statikus weblapok léteztek. Magyarországon mindez az 1990-as évek elején kezdődött el. Ezt a korszakot WEB 1.0-nak említik. A lényege volt, hogy információkat szerezzünk egy-egy weboldalról, amit valaki előre megszerkesztett. Később már megjelentek a mögöttes adatbázisok, logikák, amelyben már a felhasználók is lehetőséget kaptak az adatmanipulációra. Szükség volt egy adatátviteli technológiára, amelyre először az XML formátumot használták, azóta a legfőbb technológia erre a JSON, ami kisebb helyigénnyel rendelkezik.

A szakdolgozat is ezt a témát öleli körül. Az adatbázis migrációjára, illetve a háttérben futó kódok implementációjára a .NET technológia által nyújtott lehetőségekre támaszkodtam, mindezt C# programozási nyelven létrehozva. Az adatbázis mind MySQL-en, mind MSSQL-en képes kezelni az adatokat. A kinézeti részhez az Angular keretrendszert használtam TypeScript nyelvben a weboldal dinamikus részeit kialakítva. Mindemellett szükség volt HTML, SCSS és Bootstrap ismeretére is.

A szakdolgozat célja egy olyan webapplikáció felépítése volt, amely az autóverseny bajnokságok pontozási rendszereit bonyolítja le minden kezelhető esettel együtt. Mindebben lehetőség nyílik a felhasználókezelésre, majd létező felhasználóval saját szezonok létrehozására, azokban adatmanipulációra, illetve statisztikák megtekintésére. Fontos volt a weboldal mögött megalkotni a program nem látható részét, azaz a mögöttes logikát, minden controller-rel és service-el együtt, amely összekapcsolja az adatbázis részt a frontend felülettel.

A fejlesztés elején számtalan kitűzött cél volt. Tudtam, hogy ezt a technológiát meg szeretném tanulni olyan szinten, hogy később ebben rugalmasabban tudjak szoftvereket készíteni. Nagy hangsúlyt fektettem mind a strukturált programozásra, mind a clean coding-ra egyaránt, hogy a szoftver később is karbantartható legyen. A sok kutatás és utánajárás növelte az önbizalmamat a .NET terén, ugyanis ez egy viszonylag új technológia volt számomra egy ekkora projekt keretein belül. A szoftver kinézeti részénél is hasonlóan a magabiztosságra törekedtem, ahol szintén úgy érzem, hogy teljesítettem a magamtól elvárt szintet, ugyanis a hibákra, amikre bukkantam a fejlesztés során, kisebb-nagyobb kutatómunka folyamán ki tudtam javítani. Ez segített, hogy legközelebb is bővebb tapasztalattal induljak neki az ilyen és ehhez hasonló technológiákhoz (Vue, React).

1.1 Témaválasztás

Nem kellett sokat gondolkodnom a témán, ugyanis volt a fejemben néhány ötlet, amit az idők során terveztem megvalósítani és az egyik ilyen terv megvalósítására kiváló alkalom volt, hogy mindezt egy szakdolgozat keretei között tegyem meg.

Baráti körben évekkal ezelőtt elkezdünk játszani autóversenyzős játékokkal. Ez egy idő után olyan szintre emelkedett, hogy saját ligákat szerveztünk újabb és újabb versenyzőket bevonva. Egy verseny végén mindig volt egy eredmény, de általában nem álltunk meg egy versenynél, úgyhogy szerveztünk mellé még egy egész szériát tele versenyekkel, külön versenypályákon, hogy az összesített pontszám számítson a végén. Ehhez persze valahol vezetni kellett a pontokat, hogy a végén megszülessen a végső eredmény, hogy megtudjuk a legügyesebb versenyző nevét. Régebben erre az Excel szolgált, de amikor elkezdtem a szoftvertervezés és fejlesztés irányába mozdulni, megígértem a többieknek, hogy csinálok erre egy szoftvert, hogy elég legyen csak

megnyitni a weboldalt ahelyett, hogy egy mindig frissülő, nehezen áttekinthető Excel táblát nézegetnének. Tudtam, hogy ez egy komplex feladat lesz, ugyanis sokféle esetet kell kezelni. Egy-egy szezon külön táblát igényel, ebben kezelni kell a versenyek időpontjait, figyelni kell a csapatbeosztásokra, ki melyik csapatnál szerzett pontot melyik versenyen és hogy közben milyen pozíciót ért el. Ebben a webapplikációban nem csak ezekre kellett figyelni, hanem az áttekinthetőségre is, ugyanis ez adja a végső megoldást, hogy kiváltsuk az egy fokkal bonyolultabb Excel táblát és közben egy valós idejű képet adjon, nem úgy, mint egy chaten átküldött fájl vagy egy képernyőkép.

2. Felhasználói dokumentáció

A következő fejezet célja, hogy iránymutatást adjon az új felhasználónak az elkészített program használatára. Megtudhatja, hogy hogyan juthat el egy végleges liga elkészítéséig, azokon a lépéseken keresztül, mint az felhasználó létrehozása, bejelentkezés, szezon létrehozása, versenyek, csapatok, pilóták hozzáadása, pontozása, illetve egyes beállításokkal is megismerkedhet.

2.1. A program általános specifikációja

Sokan, akik autóversenyeznek, szeretik a pontjainkat összesíteni. Ezt megtegyük akár papíron, akár telefonon. Ha mindezt többször megtegyük jó látni egy végső eredményt, akár statisztikát, hogy hogyan állunk eredmény tekintetében, ki hogyan szokott teljesíteni egy-egy versenyen és az ilyen számokat motiváló érzés elemezni. Ebből derülhet ki, hogy hogyan is kéne teljesítenem, min kéne javítanom.

A kutatásaim során azt vettem észre, hogy ahány liga volt, mind más pontozási rendszert, illetve összesítést használt. Mindenki másképp értelmezte az Excel által nyújtott lehetőségeket és más táblát adott végeredményként. Az egyik motivációja volt így a szakdolgozatomnak, hogy egy olyan egységes rendszert hozzak létre a pontok összesítésére, amelyet bármelyik liga tud használni és egyértelműsíthetjük, hogy melyik ábra mit jelenthet. Ezért is tartalmaz az alkalmazás több szezont egyszerre. Ebből a rendszerből ráadásul olyan adatokat is kinyerhetünk, amely visszaadja a versenyző átlagos teljesítményét. Ez a rendszer túlmutat azon, hogy egy versenyző egy ligába bekerül és csak az ottani eredményéről tudunk.

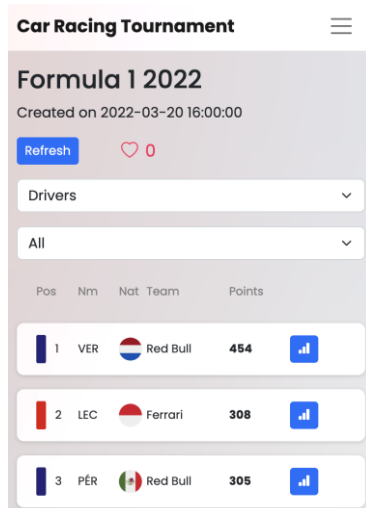
Az is a weboldal mellett szól, hogy dinamikus az adatok tekintetében. Új adatok hozzáadása, vagy meglévő adatok módosítása esetén az eddigi adatok változhatnak anélkül, hogy nekünk bármit is kéne kezelni. Ez kizárja a hibafaktorokat, félreszámolásokat, ami nagy mértékben megkönnyíti a feladatunkat.

2.2. Felhasznált módszerek

Egy weboldalnak mindig vannak általánosan megszokott elemei, amiktől nem szabad vagy legalábbis nem illik eltérni. Fontos volt a regisztrációs, illetve a bejelentkezés weboldal, ugyanis ezek az alap helyek, ahol az autentikációt kezelhetjük.

Ez kulcsfontosságú, hogy a létrehozott szezonokat tudjuk kihez kötni. Meg fog jelenni később egy főoldal, ahol az összes szezon látható kilistázva, illetve azon belül megtalálható majd minden egyes szezon részletei, amiket a felhasználó hozzáadott. Ez adja majd a programnak az egyik lényegi részét. A frontend alapjában véve komponensalapú, így minden komponensnek meg kellett oldani az adatátvitelét és a saját stílusdizájnját. Előnye, hogy minden olyan komponenst, ami többször szerepelt, elég volt csak egyszer implementálni.

A weboldal mobilnézetben is megtekinthető, ugyanis az elemzéseim alapján sokan mobilról fogják használni ezt a webapplikációt. Külön figyelmet szenteltem arra, hogy a legfontosabb adatok áttekinthetőek legyenek ezen az eszközön is. Azonban mobilkészüléken nem jelenik meg minden adat, csak ha elforgatjuk a képernyőt.



1. ábra: webapplikáció mobilnézetben

A backend résznél fontos volt kialakítani a lehető legáttekinthetőbb adatstruktúrát, ahogy szintén figyelni kellett az adatbázis szerkezet kialakítására, illetve a felhasználó- és jelszókezelésre is. Ezek mind-mind kutatómunkát igényeltek, mert .NET keretrendszerben először volt lehetőségem ilyesmit implementálni.

Belépést követően, megjelennek olyan lehetőségek, amelyekkel mi is szerkeszthetjük a weboldal tartalmát szezonok keretein belül. Amikor létrehozok egy szezont, abban én leszek az adminisztrátor. Az adminisztrátor rendelkezik minden joggal a ligán belül, azok adatait feltétlenül szerkesztheti, azonban más szezonban, ahol nem ő az adminisztrátor, mindezt nem teheti meg. Két jog létezik bejelentkezés után egy ligában: adminisztrátor és moderátori. Utóbbit az adminisztrátornak kell hozzáadnia a

ligához és ő csak a versenyzőket, csapatokat, versenyeket, illetve az eredményeket módosíthatja, adhatja hozzá és törölheti.

2.3. Gépigény

Weboldalként elég szubjektív dolog a gépigényét meghatározni a programnak. Nagy mértékben függ a használt böngészőtől és az internetkapcsolat minőségétől. Jelen esetben a Google Chrome minimum gépigényét fogom ismertetni, ugyanis sok esetben ezen a böngészőn teszteltem a webapplikációt a Safari mellett.

Processzor	1 magos, 1500 MHz
Memória	512 MB
Grafika	DirectX 8.1, 64 MB
Merevlemez	1.7 GB

2. ábra: gépigény

2.4. Oldalak

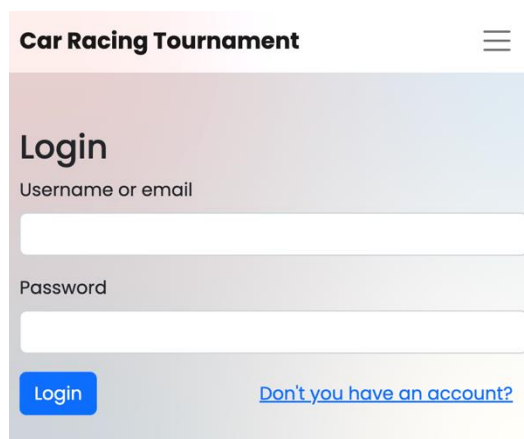
Ahhoz, hogy tudjuk kezelni ezt a webapplikációt, tisztában kell lennünk a minden egyes oldal és kattintás funkciójával. A következő fejezetekben ezt tervezem részletezni.

Kezdetben a mindenhol megjelenő fejléc (header) komponensről kell írnom. Itt megtalálhatjuk az összes olyan menüpontot, ami egy általános, egyáltalán nem specifikus oldalt nyit meg. Amennyiben nem vagyunk még bejelentkezve, akkor itt a szezonok és a statisztikák hivatkozások látszódnak, mellette pedig egy kék bejelentkezés gomb. Amennyiben létrehoztuk a profilt és már regisztráltunk is, akkor megjelenik egy harmadik hivatkozás: beállítások néven. A kék gomb helyett egy piros gomb fog látszódni kijelentkezés felirattal. Amennyiben valamelyik oldal meg van nyitva a három hivatkozás közül, akkor annak az oldalnak a felirata egy fokkal sötétebb árnyalatot fog felvenni.

2.4.1. Bejelentkezés

Ez az oldal a meglévő profil azonosítására hivatott. Ez alapján tudja eldönteni a program, hogy kik is vagyunk valójában és milyen kedvenc, vagy éppen birtokolt, moderátort ligáink vannak. Az oldalt elsősorban a fejlécben található bejelentkezés gombbal tudjuk elérni.

Itt két bemenet látható. Az elsőnél a felhasználónevet, vagy az e-mail címet kell megadni. A második lehetőség a jelszóra hívatott. Amennyiben valamelyik hiányos, vagy az azonosítás volt sikertelen a küldés gomb után, arról a program egy hibaüzenet keretein belül figyelmeztetni fog. Ebben az esetben a megadott jelszó elfog tűnni az űrlapról, egyedül a felső bemenetben megadott adatok fognak megmaradni.



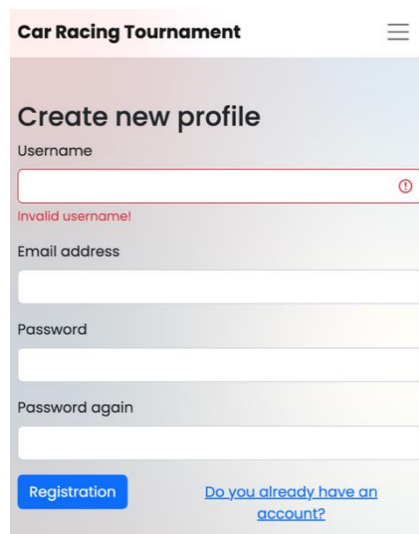
3. ábra: bejelentkezési felület

Az űrlap jobb alsó sarkában találunk egy linket, ami az új felhasználó létrehozására hívatott. Ez át fog irányítani a regisztrációs oldalra, amennyiben nincs profilunk. Amennyiben a bejelentkezés sikeres volt, visszatérünk arra az oldalra, ahonnan a bejelentkezés folyamatát elindítottuk.

2.4.2. Regisztráció

Az oldalt, ahogy az előbb is említve lett, a bejelentkezés oldalról lehet elérni. Itt egy teljesen új profilt tudunk létrehozni, ha úgy döntöttünk, hogy mi is szeretnénk szezonokat létrehozni és menedzselni.

Amennyiben elhatároztuk magunkat, nincs más dolog, mint elsősorban megadni a felhasználónevet. Ennek legalább öt karakter hosszúnak kell lennie. Ez alapján fognak elsősorban azonosítani minket. A következő adat az e-mail cím megadása. Erre később a felhasználó azonosítására lehet szükség. Ennek az adatnak e-mail formátumúnak kell lennie, különben nem kerül elfogadásra. Az utolsó két bemeneti adat a jelszó és annak ismétlése. A rendszer abban az esetben fogadja el a jelszót, ha az tartalmaz legalább egy kisbetűt, legalább egy nagybetűt, legalább egy számot és a hosszúsága meghaladja a nyolc karaktert. Fontos kiemelni, hogy a két jelszónak meg kell egyeznie. Ha valamelyik adat érvénytelen, vagy rendszernek nem megfelelő eset merült fel, abban az



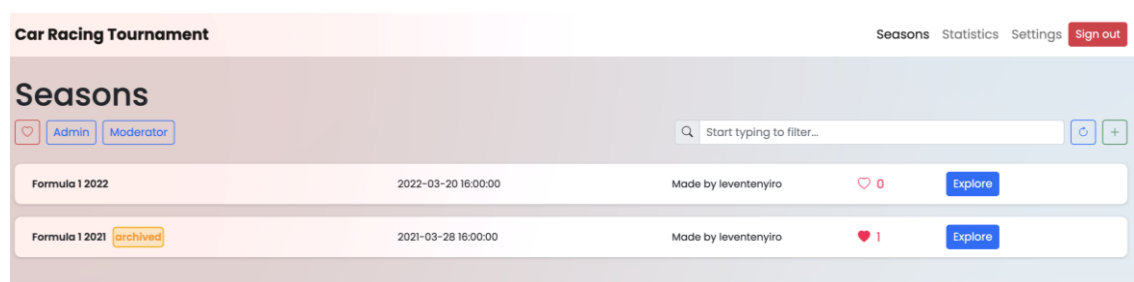
4. ábra: regisztrációs felület

esetben egy hibaüzenet fog megjelenni és az általunk választott jelszót újra meg kell adni mindkét bemenetnél. A többi bemeneti adat megmarad, azonban javítani kell őket, ha valamelyik nem volna helyes.

2.4.3. Seasons

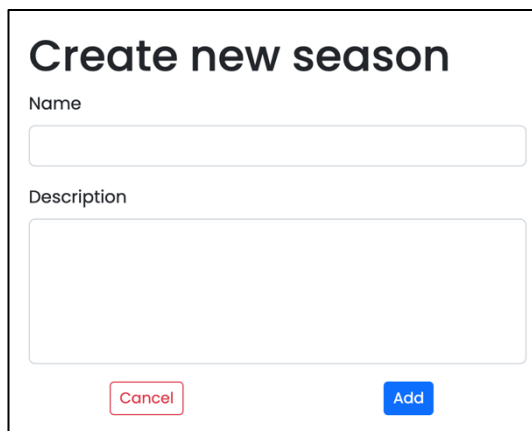
Ha a weboldalra térünk, akkor az első oldal, amivel szembe találjuk magunkat, az a szezonok listája. Ha vannak már szezonok tudunk köztük keresni, frissíteni, illetve bejelentkezett állapot esetén szűrni is. Ha nem jelenik meg adat, azt a weboldal egyértelműen jelezni fogja egy üzenettel. Minden egyes szezonnál meg fog jelenni a neve, a készítés dátuma helyi időben, a létrehozó felhasználó neve, egy kedvencek és egy felfedezés gomb. Mobilnézet esetében azonban csak a név és a szív fog látszódni a reszponzivitás érdekében. Amennyiben a szezon archiválva van, a felhasználó arról is értesül egy jelvény segítségével.

Bejelentkezés után három checkgomb fog megjelenni a jobb felső sarokban. Sorrendben a kedvencek, admin és moderátor gomb. Ha egyet benyomunk, akkor az szűrni fog, de ha kettőt, vagy mind a hármat, akkor hozzáadja azokat a szűrt elemeket, amik mondjuk a kedvencek és az admin uniója lesz. Természetesen szezonok duplikációja nem fordulhat elő. A kedvencek gombbal az általunk kedvelt szezonok fognak látszódni. Az admin gombbal azokat a szezonokat tekinthetjük meg, amelyeket mi hoztunk létre, vagy épp mi vagyunk a vezetői. A moderátor gombbal pedig azokat a szezonokat láthatjuk, amelyekhez hozzá vagyunk adva moderátorként.



5. ábra: szezonok listája

Bejelentkezett állapotban létezik továbbá egy hozzáadás gomb, ahol egy ablak fog megnyílni egy űrlappal. Ez az egyetlen hely az egész programban, ahol új szezont adhatunk hozzá a rendszerhez. Meg kell adnunk a szezon nevét, illetve opcionálisan megadhatunk hozzá egy leírást is, hogy a felhasználók információt kapjanak a liga részleteiről. Amennyiben a nevet nem adtuk meg, vagy hibásan tettük mindezt, egy hibaüzenet figyelmeztetni fog minket erről. Ha sikeresen létrehoztuk a ligát, abban az esetben át leszünk irányítva annak az oldalára, hogy egyből láthassuk annak létezését.

A screenshot of a web form titled "Create new season". It has two input fields: "Name" and "Description". The "Name" field is a single-line text input, and the "Description" field is a multi-line text area. At the bottom of the form, there are two buttons: a red "Cancel" button and a blue "Add" button.

6. ábra: szezon hozzáadása

Ami minden esetben meg fog jelenni bejelentkezés nélkül is, az a keresés, illetve a frissítés. Az előbbivel azonnali időben lehet szűrni a billentyű lenyomásával, míg az utóbbival frissíteni lehet az oldalt, ha esetleg valaki feltöltött az adatbázisba egy újabb szezont. Ha egy szezonra rányomunk át leszünk irányítva annak részletező oldalára.

2.4.4. Season

Ez a program egyik legrészletesebb oldala. Rengeteg funkció található itt, több oldalon keresztül lehet mesélni arról, hogy milyen folyamatok zajlanak le itt egy-egy interakció során.

Kezdetben minden állapotnál megjelenik a szezon neve, létrehozásának időpontja, egy frissítés és egy kedvencek gomb, ami megmutatja, hogy eddig hányan jelölték kedvencnek az imént megtekintett szezont. Asztali nézetben a jobb oldalon, mobilnézetben legalul megjelenik egy moderátor lista az adminnal együtt, ami megmutatja, hogy kinek van jogosultsága a szerkesztéshez, alatta pedig a szezon leírása látható.

Megtalálunk itt továbbá 2 lenyíló ablakot, amin a tábla típusát (drivers, teams, races) és annak tartalmát állíthatjuk be (minden, vagy a típusok nevei felsorolva). Ennek a segítségével hat féle táblát tekinthetünk meg, amit a jogosultság fázisában részletezek. Amennyiben nincs adat valamelyik táblán, úgy egy üzenetet fogunk látni erről.

Ha beléptünk, de nincs jogosultságunk az éppen megtekintett liga szerkesztéséhez, akkor egyedül a „kedvenc” gomb lesz az egyetlen lehetőség, ami használhatóvá válik. Abban az esetben, ha a jogosultságunk érvényes ehhez a ligához, már tudunk adatokat szerkeszteni. Adminisztrátori jogosultsággal módosítani a liga részleteit, archiválni és törölni is képesek vagyunk. Adhatunk a szezonhoz új moderátorokat, elő is lehet léptetni egy moderátort magunk helyett, de elbocsátani is lehetséges. Ezeket moderátor jogosultsági szinten nem tehetjük meg. Minden esetben, amikor valamilyen módosítást tervezünk végrehajtani, megjelenik egy ablak, ahol megerősíthetjük ezt a döntésünket.

Pos	Name	Points
1	Red Bull	789
2	Ferrari	554
3	Mercedes	515

7. ábra: szezon részletező oldala

Az első tábla, ami megjelenik előttünk az az összes pilóta adatai. Asztali nézetben megtekinthetjük a szezonban elfoglalt pozíciókat, a hozzá tartozó felhasználóneveket, valós neveket, nemzetiségeket, rajtszámokat, aktuális csapatot és az eddig elért pontokat a versenyek során. Amennyiben nem vagyunk bejelentkezve, úgy egy gombot is használhatunk, amely átirányít minket az aktuális versenyzőnek a statisztika oldalára. Ha moderátor vagy adminisztrátori jogosultsággal rendelkezünk, akkor képesek leszünk hozzáadni, szerkeszteni a pilótákat, illetve törölni őket. Ehhez fel fog jönni egy ablak. A hozzáadás gomb a tábla tetején található, viszont az adatmanipulációval kapcsolatos gombok minden egyes pilóta mellett megtalálható. Hozzáadásnál vagy módosításnál meg kell adnunk egy nevet és egy rajtszámot. A valós név, nemzetiség és az aktuális csapat megadása nem kötelező (lehetnek tartalékpilóták). Ha két ugyanolyan nevű vagy rajtszámú pilóta van, akkor egy hibaüzenetet fogunk kapni.

Ha az alsó lenyíló menüben kiválasztjuk a pilóta nevét, meg fognak jelenni az általa elért eredmények a versenyek függvényében. Láthatjuk a verseny nevét és annak időpontját helyi időben, az elért pozíciót, pontszámot és hogy melyik csapatban érte ezt el. Fontos kiemelni, hogy egy pilóta két eredményt egy versenyen nem érhet el. Felül egy

hozzáadás gombot találunk. Az eredmények mellett pedig - moderátor és adminisztrátor jogosultság esetében – megjelennek a módosítás és törlés gombok. Amennyiben a hozzáadás gomb lehetőségével élünk, akkor ezt csak abban az esetben tudjuk megtenni, amennyiben létezik legalább egy csapat, illetve szintén legalább egy verseny a versenynaptárban. Ha ez nem teljesül, a program egy hibát fog dobni és nem nyitja meg az ablakot mindaddig, amíg ezeket az adatokat hozzá nem adjuk a ligához. Amennyiben az ablak megnyílik, ki kell választani, hogy az adott versenyt melyik csapat színeiben érte el, melyik versenyen tette meg mindezt, milyen pozíciót ért el és mennyi pontot kap ezért (a pontszámnak oszthatónak kell lennie 0.5-el). Amennyiben ezen a versenyen már ért el eredményt az aktuális pilóta, vagy esetleg negatív pontszámot adtunk meg, akkor egy hibaüzenet fog minket fogadni.

A felső lenyíló menüben kiválasztjuk a pilóták helyett a csapatokat. Alul szintén megjelenik az „összes” lehetőség, ezen kívül pedig a csapatok nevei. Ilyenkor elsőre az összes lehetőség fog tábla formájában megjelenni, ahol láthatjuk felsorolva a liga összes csapatát, a színével, nevével és az eddig elért pontjaival együtt. Megfelelő jogosultsággal létrehozni, módosítani és törölni is képesek vagyunk csapatokat. A megjelenő ablakban meg kell adni ennek a nevét és a színkódját. Amennyiben olyan nevet adtunk meg a csapatnak, ami már létezik, vagy hiányosan adtuk meg a csapatnevet, akkor egy hibaüzenet fog megjelenni.

Amennyiben van már csapatunk, az alsó lenyíló menüben ezt kiválasztva megtekinthetjük, hogy egyes versenyeken milyen eredményeket ért el az aktuális csapat. Ezek az eredmények a versenyek időpontjai szerint vannak rendezve és a helyi időt mutatják. Itt meg fog jelenni a verseny neve, annak helyi időpontja, illetve egy-egy versenyen elért pontok száma az aktuális csapat tekintetében. Ezen a táblán nincsenek funkciók, ugyanis az eredményeket elsősorban pilóta és verseny függvényében kell létrehozni, hogy ezután kiszámolja a program a csapat pontjait.

A harmadik opció a felső lenyíló menüben a „versenyek”. Az „összes” opció ilyenkor megjelenik tábla formájában és láthatóvá válik, hogy milyen versenyek vannak a szezonban. Azoknál a versenyeknél, ahol még nem lett rögzítve eredmény, ott a verseny nevét, helyi időpontját láthatjuk. Abban az esetben, ahol már van eredmény láthatjuk a nyertes nevét és annak a csapatnak a nevét, aminek a színeiben győzött. Megfelelő jogosultság esetén hozzáadhatunk, módosíthatunk és törölhetünk versenyeket is. Ebben

az esetben meg kell adni a verseny nevét, dátumát és a pontos időpontját helyi időben, hogy a részt vevő pilóták tudják, hogy mikor kezdődik egy-egy verseny. Ha valamelyik adat hiányos, vagy létezik már verseny ilyen névvel, akkor a szokásos hibaüzenet fog megjelenni. Törlés esetén, ahogy várnánk, el fognak tűnni a hozzá tartozó eredmények, ezelőtt persze egy ablak fog megjelenni, hogy biztosak vagyunk-e a törlés véglegesítésében.

Az utolsó táblánk egy adott versenyhez tartozó eredményeket tartalmazza, hasonlóan a pilóták eredményeit tartalmazó táblához, csak itt egy versenyre lebontva látjuk ezt pozíció szerint rendezve. Látható itt a megszerzett pozíció, pilóta neve, valószínűség, rajtszám, a csapat, akinek a színeiben versenyzett, illetve a megszerzett pontok számát. Adminisztrátori és moderátori jog esetében megtaláljuk a hozzáadás, módosítás és törlés gombokat. Hozzáadni csak akkor tudunk új eredményt, amennyiben van már legalább egy versenyző és legalább egy csapat a ligában, ezután megjelenik az ablak. Amennyiben negatív pontszámot adunk meg, a program erről figyelmeztetni fog. Továbbá itt verseny szerint csoportosítottunk, így a versenyző nevét kell megadni a verseny neve helyett.

Rengeteg esetben megnyílik egy figyelmeztető ablak, hogy biztosak vagyunk-e egyes műveletekben. Ezeknél két lehetőség van, a „mégse” gombra nyomva nem történik semmi, azonban, ha elfogadjuk ezt, az visszafordíthatatlan adatmanipulációs következményekkel jár. Ezt a döntést fontos mérlegelni ilyen esetekben.

Fontos továbbá kiemelni, ahogy azt a táblák és az űrlapok igazolják, egy pilótának van aktuális csapata, illetve az eredményekben egy külön csapat, akivel a pontot elérte. Ez az esetek többségében ugyanaz lesz, sőt az eredmény megadásánál a program automatikusan kiválasztja ezt indítócsapatként. Előfordulhatnak azonban olyan esetek, amikor a versenyző egy másik csapathoz igazol át szezon közben, azonban ilyenkor a pontokat nem viheti át a másik csapathoz, így ő azt csak egyéniben kapja meg. Ezért volt szükség erre az adattagra a versenyek során.

2.4.5. Statisztika

Fontos dolognak tartottam, hogyha egy olyan szoftverről beszélünk, ahol egyszerre több liga is lehet és egy pilóta több ligában is részt vehet, akkor nyomon tudjuk

követni a teljesítményét. Erre a diagramokat tartottam a legmegfelelőbbnek, illetve a leglátványosabbnak.

Ezt az oldalt többféleképpen is elérhetjük. Egyik lehetőség valamelyik szezon összes vezetőjét felsoroló táblában, ahol megtekinthetjük átirányítva az aktuális versenyző statisztikáit. Ebben az esetben az oldalnak a „name” paramétere automatikusan kitöltődik. Egy másik lehetőség az eléréshez, ha a fejlécből navigálunk el a „Statisztika” linkre kattintva.

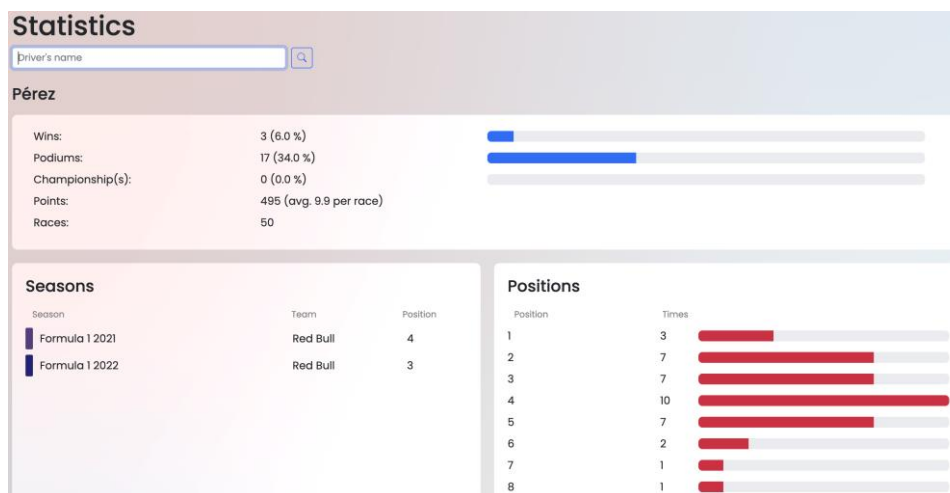
Az utóbbi navigációs esetet alapul véve a keresési mező nem lesz kitöltve és hozzátartozó adatok se fognak megjelenni lentebb. Ha egy versenyző statisztikáit megszeretnénk tekinteni, ahhoz tudnunk kell elsősorban azt a nevet, amivel egy vagy több ligában részt vett. Ezt a nevet a keresőmezőben kell beírni, majd a keresés gombra kattintva a program az adatok lekérése után megjeleníti a statisztikai adatokat.

Az oldalon jelenlévő adatokból olyan dolgokat tudhatunk meg, hogy általánosságban vagy egyes szezonokra lebontva, hogyan teljesít egy versenyző, vagy éppen egyes pozíciókat milyen gyakorisággal ér el.

Az első nagyobb csoportban az általános teljesítményt láthatjuk. A legtöbb esetben egy diagramot is látunk mellette, ami az arányt mutatja meg. Felsorolva az egyes adatokat az első megmutatja, hogy egy versenyző mennyi győzelmet ért el, mellette a százalék pedig megmutatja, hogy milyen gyakran szokott nyerni a részvételei hányadában. A második adat az elért pódiumokat mutatja meg. Ez attól függ, hogy hányszor szerzett első, második vagy harmadik helyezést. Az előző esethez hasonlóan, itt is található egy százalék, ami szintén a részvételei hányadában számol. A következő eset a bajnoki címek száma. Akkor számít egy szezonban elért első pozíció bajnoki címnek, ha a liga archivált állapotban van. A mellette lévő százalék annak az információnak a hányadában számíthat ki, hogy az adott versenyző mennyi ligában vett részt. A következő két információ már diagram nélkül mutatja az adatokat. Az előbbi az eddig elért pontszámokat mutatja meg, mellette megtaláljuk, hogy átlagosan egy versenyen mennyi pontot ér el. Ezt a részt vett versenyek hányadából tudhatjuk meg. Az utolsó adat a csoportban megmutatja, hogy mennyi versenyen vett részt a versenyző. Sok adat ennek az információnak a birtokában számíthat ki.

A következő nagyobb csoport az egyes szezonokon elért pozíciókat mutatja be, attól függően, hogy egy-egy versenyen az elért pontjai alapján hol szerepel. Megtaláljuk

itt a szezon nevét és az elért pozíciót. Az ezután lévő csoport („Pozíciók”) megmutatja, hogy adott pozíciókat hányszor ért el az adott versenyző, mellette egy diagrammal. Itt két oszlop van: az egyik, ami az elért pozíciót mutatja meg, mindet egyszer, a mellette lévő pedig az előfordulás számát. Amennyiben valamelyik pozíciót nem érte el egy versenyző, abban az esetben ez a pozíció nem fog előfordulni a csoportban.



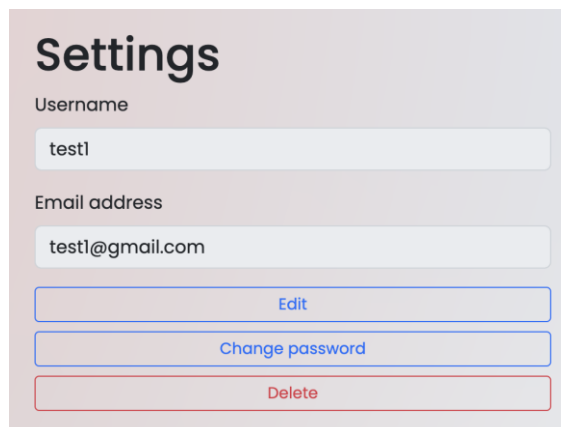
8. ábra: statisztika oldal

2.4.6. Beállítások

Amennyiben egy weboldal rendelkezik felhasználókezeléssel, ahhoz feltétlenül szükséges egy olyan oldal is, ahol a felhasználó módosíthatja a profilja egyes adatait, vagy módosíthatja a jelszavát. Erre az eshetőségre nálam a „Beállítások” oldalon van lehetősége a felhasználónak. Ezt az oldalt a fejlécben tudja elérni a megfelelő linkre kattintva. A megnyitáskor betöltésre kerül minden felhasználói adat, névlegesen a felhasználó név és az e-mail cím. Ez alatt három gomb található más-más funkciókkal.

Az első gomb segítségével a két felső bemenet fog módosíthatóvá válni. Ilyenkor a három alsó gomb helyett megjelennek a „Mégse” és a „Módosítás” gombok. Az előbbire kattintva visszaáll minden adat az eddigi állapotába és újra lezárja a két felső bemenetet és megjelenik a szokásos három gomb, amelyek alaphelyzetben találhatóak meg. A másik gombra kattintva a program ellenőrzi a bevitt adatok helyességét és amennyiben hibát találna, abban az esetben ezt jelzi számunkra egy hibaüzenet formájában és még nem zárja le a szerkesztési folyamatot. Ez akkor fordulhat elő, ha a felhasználónév vagy az e-mail cím formátuma nem megfelelő, illetve előfordulhat, hogy létezik felhasználó ilyen adatokkal. Amennyiben sikeres volt a módosítás, abban az

esetben az új adatok fognak megjelenni az imént lezárt felső bemenetekben és újra láthatóvá válik a három alsó gomb.



The image shows a 'Settings' form with a light gray background. At the top, the word 'Settings' is written in a large, bold, black font. Below it, there are two input fields. The first is labeled 'Username' and contains the text 'test1'. The second is labeled 'Email address' and contains the text 'test1@gmail.com'. Below these fields are three buttons: 'Edit' (blue text on a light blue button), 'Change password' (blue text on a light blue button), and 'Delete' (red text on a light red button).

9. ábra: felhasználói profil szerkesztése

A második gomb a jelszó megváltoztatására szolgál. Kattintáskor megjelenik egy ablak, ahol három bemenetet fogunk látni, alatta két gombbal. Az első bemenet a régi jelszó megadására szolgál, az ezután következő két bemenet az új jelszót és annak ismételt bevitelét kell tartalmaznia. Ha a régi jelszó helytelen, vagy az új jelszó nem a formátumnak megfelelő, vagyis nem tartalmaz legalább egy nagybetűt, legalább egy kisbetűt, illetve legalább egy számot és nem haladja meg a nyolc karakterhosszt, vagy akár az újonnan megadott jelszavak nem egyeznek, abban az esetben a program egy hibaüzenettel fog visszatérni és az inputmezők kitöltését előlről kezdhethetjük, ugyanis azok értékei el fognak tűnni. Amennyiben a jelszóváltoztatás sikeres volt, abban az esetben el fog tűnni a jelszóváltoztatás ablak.

A harmadik gomb a felhasználói fiók végleges törlésére hívatott. Amennyiben emellett a lehetőség mellett döntünk, úgy a program fel fogja tenni a kérdést, hogy biztos kívánjuk-e törölni a profilunkat. Amennyiben a válasz igen, abban az esetben a fiókunk mellett a szezonjaink is végleges törlésre kerülnek.

3. Fejlesztői dokumentáció

Ebben a fejezetben a célom bemutatni a program felépítését fejlesztői szemmel. A fejlesztés menetét szeretném részletesebben leírni a kezdeti tervezésektől, számításoktól egészen a tesztelési szakaszig. Szó lesz itt az adatbázis kialakításáról, bemutatom a hozzá tartozó ER diagramot, illetve bemutatom minden egyes tábla adattagjait és azoknak kulcsfontosságú szerepeit a programon belül. Külön részt szentelek a program logikai részére, hogy a háttérben hogyan is futnak le a folyamatok. Szó esik a metódusokról, osztályokról és végpontokról is. Mindezek után említésre kerül a webapplikáció fejlesztési folyamata, annak metódusai, lekérései, oldalainak szerkezete és az ott zajló folyamatok. A fejezet végén a teszteseteket fogjuk végig nézni.

3.1. Adatbázis

Ha hosszútávú adattárolásra szánjuk el magunkat egy program esetében, ahhoz mindenképpen szükségünk lesz egy adatbázisra, amit bármikor elérünk és adatokat kérhetünk le a nap bármely szakaszában. Erre a célra még mindig toplistás helyen szerepel a MySQL, MSSQL, mint technológia. Ezek mind relációs adatbázisok. Léteznek azonban NoSQL technológiák, amelyek szembe fordulva a szokásos SQL sémáknak JSON-szerűen tárolják az adatokat. A mi esetünkben célszerű az előbbit használni (azaz a relációs adatbázisokat), ugyanis a táblák között számos adatkapcsolattal – így idegenkulcsokkal is – rendelkezünk.

A modernebb szerveroldali programok már tartalmazzák azt a lehetőséget, hogy az objektumokat ORM technológiával hozzuk létre és azokat az adatbázisra migráljuk. Az ORM (object-relational mapping), azaz objektum-relációs leképezés egy olyan technológia, ami segít abban, hogy a szerveroldalon létrehozott objektumokat az adatbázisban is könnyen tudjuk kezelni, tárolni. Erre bizonyos technológiákban más és más könyvtárak használhatóak. NodeJS-ben a népszerűbbek között van a Sequelize, de amit mi most használni fogunk, az a Microsoft által létrehozott Entity Framework Core (továbbiakban EFCore). A .NET technológiában ez a legelterjedtebb ORM könyvtár.

Ahhoz, hogy a programunk egy adatbázist migráljon, el kell készítenünk az adatbázis struktúráját, mindezt C# nyelven. Meg kell adnunk a típusát egy-egy adattagnak és ha igényli ezt a specifikáció, adhatunk hozzá további speciális beállításokat gondolva

itt a kötelező vagy épp egyedi értékekre. Mindezt nekünk létre kell hoznunk egy osztályt, amelyet a DbContext osztály alosztályaként definiálunk. Ezt a módszert „Code First Database”-nek nevezzük. Amennyiben szeretnénk adatokkal feltölteni az adatbázisunkat, úgy egy seed-elést is meg kell írni. Ha ezzel megvagyunk, el kell készítenünk a Program.cs-ben az adatbázis csatlakozásunkat „connection string” segítségével, illetve egy-két további opciót is be lehet állítani. A Program.cs a backend felületünknek a fő motorja. Itt kell meghatározni a backend működésének elengedhetetlen paramétereit.

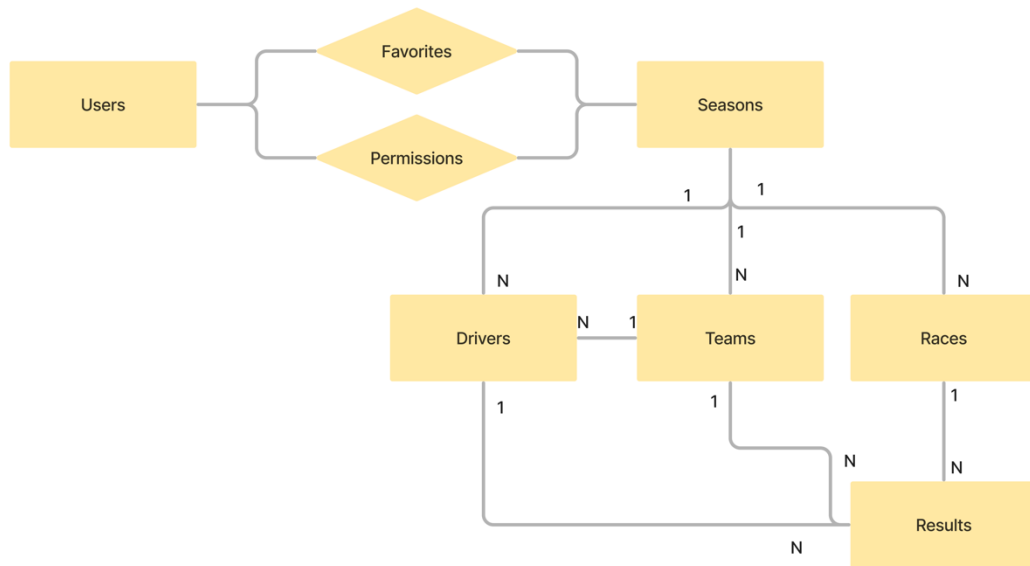
Eleinte MySQL adatbázisra migráltam lokális futtatásnál, ugyanis a Xampp nevű program ezt a fajta adatbázis technológiát tartalmazza és ezt viszonylag könnyen és gyorsan be lehet üzemelni. Később, amikor a „production” módban is futtatásra került a program, akkor mindezt MSSQL-re migráltam. Ahogy látszódni is fog, az egész „Code First” technológiának köszönhetően bárhová el lehet helyezni az adatbázisunkat akadálytalanul és elég ehhez csak egyfajta programnyelv birtokában lenni. Amikor migrálni szeretnénk, abban az esetben két parancsot kell kiadnunk:

```
dotnet ef migrations add init  
  
dotnet ef database update
```

10. ábra: adatbázis migráláshoz szükséges parancsok

A következő alfejezetekben a táblákat és azok felépítéseit fogjuk áttekinteni. Egy felsorolás keretein belül fog fény derülni az adattagokra a táblákon belül. Fontos kiemelni, hogy az azonosítók, azaz az „id”-k mindenhol szerepelnek, azok egyediek, elsődleges kulcsként funkcionálnak és az adattípusuk Guid. Ez egy globálisan egyedi azonosító, aminek közel nulla a valószínűsége, hogy két ugyanolyat generáljunk, ezáltal

kiválóan teljesíti az azonosítás feladatát. Az ER (egyed-kapcsolat) diagramon kiolvasható, hogy melyek lesznek ezek.



11. ábra: Adatbázis ER-diagramja

3.1.1. Users

Ez a tábla szolgál arra, hogy eltároljuk a regisztrált felhasználók alapadatait. Ez alapján tudják magukat azonosítani bejelentkezéskor, illetve szezonokat ezzel tudnak létrehozni, kezelni (akár adminként, akár moderátorként).

- Username – ez egy szöveg típusú adattag, felhasználónév, „unique” tulajdonsággal rendelkezik, ugyanis bejelentkezésnél ez alapján azonosítjuk a felhasználót az e-mail cím mellett.
- Email – szintén szöveg típusú adattag, felhasználó e-mail címe, egyedi tulajdonsággal felruházva, hogy ezzel is be tudjuk azonosítani a felhasználói fiókot.
- Password – szöveg típusú adattag. Célja, hogy a felhasználói fiók illetéktelen személyektől elzárt maradjon, ezen belül ne lehessen adatmanipulációkat végezni.

Amennyiben a felhasználó azonosította ezzel az adattaggal a jogosultságát, már bármit tehet, ami a jogosultsági körébe belefér.

3.1.2. Seasons

Ennek a táblának a célja, hogy a szezonok alapadatait eltároljuk. Egy-egy ilyen szezonban fogjuk megtalálni majd az eredményeket, versenyeket, versenyzőket, csapatokat a webes felületen.

- Name – szöveg típusú adattag. A szezon neve, ami alapján be tudjuk azonosítani, hogy milyen célra lett létrehozva. Felhasználó számára az első felismerhető információ, amikor egy új szezonnal találkozunk.
- Description – szöveg típusú adattag. A szezon bővebb leírását szolgálja, hogy a felhasználó tudja, hogy biztos arról a ligáról van-e szó, amit ő keres. Nem kötelező megadni.
- IsArchived – boolean adattag. Valójában flagként szolgál arra, hogy a szezon maga archivált állapotban van-e vagy sem.
- CreatedAt – dateTime adattag. Egy mellékes azonosítónak szánt adattag, arra a célra, hogy két azonos nevű szezonat biztosan meg tudjunk különböztetni.

3.1.3. Favorites

Ez a kapcsolótábla arra hivatott, hogy eltárolja egyes felhasználók kedvencnek jelölt szezonjait, illetve fordított esetben láthatjuk, hogy egy adott ligát mennyien jelöltek be kedvencnek. Két adattaggal rendelkezik, mindkettő Guid típusú idegenkulcsok. Ezek rendre a UserId, ami a Users tábla egy rekordjának az Id adattagjára hivatkozik, a másik a SeasonId, ami a Seasons tábla egyik rekordjának az azonosítójára – Id-ra – hivatkozik.

3.1.4. Permissions

Ez a tábla hasonló a „Favorites” táblához, azonban rendelkezik plusz egy adattaggal. Ennek a táblának a célja, hogy szezonok alapján kezeljük, hogy ki milyen joggal fér hozzá egy-egy szezonhoz, illetve azt is bemutatja, hogy egy felhasználó, melyik ligákhoz fér hozzá.

Két idegenkulccsal rendelkezik a tábla, hasonlóan a „Favorites” táblához, itt is a UserId-val egy felhasználót azonosítunk az Id-ja alapján és a SeasonId, amivel a szezonat tudjuk beazonosítani, annak Id adattagjának segítségével.

A fentiek mellett megjelenik egy újabb adattag, ami a jogosultság típusát írja le. Ahogy az az előzőekben kiderült kétfajta jogosultság típus van: admin és moderátor. Ez az adattag egy számot tárol el integer típusban. „1”-es számmal jelöli az admin, míg „0”-ás számmal a moderátori jogosultságot.

3.1.5. Drivers

A szezonok tartalmazhatnak versenyzőket, így elengedhetetlen, hogy erre is létrehozzunk egy táblát és definiáljuk a versenyzőket, valamint azok adatait.

- Name – szöveg típusú adattag. Arra szolgál, hogy a játékbeli felhasználónevét a versenyzőnek elmentsük és ez alapján azonosítsuk.
- RealName – szöveg típusú adattag - előfordulhat, hogy a valós nevét is el szeretnénk menteni egy-egy versenyzőnek, hogy egy másodlagos azonosítási lehetőséget adjon. Nem kötelező megadni.
- NationalityAlpha2 – szöveg típusú adattag – célja, hogy az adott nemzetiség két betűs kódját tárolja el az adatbázis
- Number – integer. a versenyző rajtszáma az adott szezonban.
- ActualTeamId – Guid. idegenkulcs, ami a Teams tábla egyik rekordjára hivatkozik. Ez mutatja meg, hogy a versenyző éppen melyik csapatban van. Lehet null is az értéke.
- SeasonId – Guid. idegenkulcs, ami a Season egyik rekordjára hivatkozik. Megmutatja, hogy a versenyző melyik ligában vesz részt.

3.1.6. Teams

A szezonokban többféle csapat előfordulhat. Az autóversenyekben ez egy megszokott felállás, hogy a versenyzők csapatok színeiben indulnak, ezért is van szükség erre a táblára.

- Name – szöveg típusú adattag. Ez alapján tudja a felhasználó azonosítani elsősorban, hogy a ligán belül melyik csapatról van szó.
- Color – szöveg típusú adattag. A formátuma #abcdef, vagyis a színkódot tárolja el HEX formátumban. Ad egy színvilágot a csapatnak még egyedibbé téve azt.
- SeasonId – Guid. Idegenkulcs, ami a Season tábla egyik rekordjára mutat.

3.1.7. Races

A versenyek nélkülözhetetlen tartozéka a ligáknak, hiszen erről szól az egész sport. Egy liga több versenyt is tartalmazhat, ahogy azt látni is fogjuk.

- Name – szöveg típusú adattag. Ennek a névnek segítségével lehet azonosítani a versenyt egy szezonon belül.
- DateTime – dateTime típusú adattag. Megmutatja, hogy a verseny mikor kezdődik vagy kezdődött.
- SeasonId – Guid. Idegenkulcs, ami egy szezonra hivatkozik.

3.1.8. Results

Ez talán az egyik legkomplexebb tábla az egész adatbázisban. Eltárolja az eredményeket egy versenyző, egy csapat és egy verseny függvényében.

- Type – szöveg típusú adattag. A programban 4 féle értéket vehet fel: Finished, DNS, DNF, DSQ. Leírja, hogy az eredménynek mi a végkimenetele. A DNS (did not started) jelentése, hogy nem indult el a versenyen, a DNF (did not finished) esetén nem ért célba, és a DSQ (disqualified) előfordulásánál a versenyzőtől elvették az eredményét, valamilyen szabálytalanság miatt.
- Position – integer. Leírja, hogy milyen pozíciót ért el egy versenyző, amennyiben az eredmény típusa „Finished”. Emiatt lehet az értéke null.
- Point – double. Ezáltal adhatjuk meg, hogy egy versenyző hány pontot szerzett a verseny alkalmával.
- DriverId – Guid. Megadjuk, hogy melyik versenyző szerezte a pontot.
- TeamId – Guid. Leírja, hogy melyik csapat képviselőjében szerezte meg a pontot a versenyző.
- RaceId – Guid. A versenyre hivatkozik, amelyen az eredményszerzés megtörtént.

3.2. Szerveroldal

Két lehetőség adódott a szerveroldal tervezésénél a technológia választásra. Egyik lehetőség volt a NodeJS erre az implementációra, míg a másik a .NET keretrendszer. Mivel utóbbival többet foglalkoztam az elmúlt időben a munkámból adódóan, illetve a fejlesztési környezet és az programstruktúra közel áll az ízlésemhez, így nem volt kérdés, hogy ebben szeretnék tapasztalatot szerezni a szakdolgozatom készítése során.

A program MVC architektúrában készült, ami azt jelenti, hogy a nézetből (View) történnek az interakciók. Jelen esetben erre az Angular technológiában fejlesztett

webapplikáció szolgál. Ez API végpontokat hívva elindítja a vezérlőben (controller) a megfelelő metódust, amely opcionális esetekben adatokat is kap. Itt történnek a válaszok visszaadása, illetve az üzleti logika is itt hívódik meg. A modell az adatbázissal szoros kapcsolatban áll (köszönhetően az ORM technológiának is), ami segít az adatlekérésben. Fontos kiemelni, hogy a programom tartalmaz még egy negyedik szintet, mégpedig a modell és a controller között. Ez a szolgáltatás (service) szint. Itt történnek a számítások, amelyek egyes válaszokhoz elengedhetetlen a program során. Például statisztikákat számol ki, vagy sok esetben csak az adatbázisnak közvetíti, hogy az milyen műveleteket hajtson végre. A szolgáltatás szintjén a folyamatok úgy vannak csoportosítva, hogy a folyamat éppen melyik adatbázist módosítja. Például amennyiben a csapattáblát szeretnénk módosítani, úgy a végrehajtás mindenképpen a TeamService-ben fog történni. A vezérlő szintjén ebben a tekintetben más a helyzet, ott a csoportok inkább adatkapcsolat szintjén vannak kezelve. Ha egy ligán belül új csapatot szeretnénk megadni, abban az esetben mindezt a SeasonController-ben tehetjük meg, ugyanis egy szezonobjektumhoz kapcsolódik az újonnan megadott adat.



12. ábra: UML diagram

A Program.cs fájl nem csak a pár fejezettel ezelőtt említett adatbázis migrációt tartalmazza, hanem sok már tulajdonságot is, ami mellett nem érdemes elmenni szó nélkül. Sorban haladva a kódban itt adhatjuk meg, hogy a szolgáltatás réteg bárholnán könnyedén elérhetőek legyenek a „dependency injection” keretein belül. Ehhez az AddScope generikus metódust kell használnunk, ahol megadjuk az szolgáltatás interfész nevét és mellé a szolgáltatás réteg megfelelő osztályát. Ezután az AutoMapper-t láthatjuk. Ez arra szolgál, hogy a beérkező DTO-kat minden gond nélkül átalakítja az adott osztályra. Ennek a konfigurációját kell megadni paraméterben, amit egy külön fájlban hoztam létre a Profiles/ mappában. AutoMapperConfig.cs-ben le kell írni, hogy miből mibe fordítson és ezt fordítva is megtehesse-e. A DTO-kat (data transfer object) több helyen használtam. Pont azt a célt szolgálja, hogy ne kelljen minden adatot megadni, például egy szezon esetében ne kelljen megadni, hogy ki a tulajdonosa egy adott ligának, elég csak a nevet és a leírást. Továbbá a kódon megtalálhatjuk az autentikációval kapcsolatos beállításokat, ahol feltétlenül be kellett állítani, hogy JWT (JSON Web Token) esetében milyen kulccsal dolgozzon. Az autentikáció után megadtam a Swagger konfigurációit. A Swagger egy felületet szolgáltat, ami segít átláthatóbbá tenni a végpontokat és könnyebbé teszi azok kézi tesztelését. Nagyon megkönnyítette a fejlesztés ezen szakaszát. Ezután fontos volt kezelni azt az esetet, hogy a szerveroldal ne essen DDoS áldozatául, így egy rate limitert beépítve meg lehet határozni, hogy egy másodperc leforgása alatt mindössze csak 100 lekérés történjen. A végén pedig a CORS beállításokat is meg kellett határozni, hogy az adatfolyamat a két domain (szerver- és kliensoldal) között zavartalanul működjön.

A következő fejezetekben végpontcsoportok, ha jobban tetszik vezérlők szerint fogok írni, bemutatva a meghívás módját, a vezérlők folyamatát, adott esetben a szolgáltatási szintet, illetve milyen adatbázismanipuláció történik. A végponthívásoknál csak GET (lekérés), POST (feltöltés), PUT (módosítás), DELETE (törlés) jellegű végponthívások lesznek. Ezeket jelzem is a bekezdések során.

3.2.1. Driver

Ebben a vezérlőben olyan módosítások, vagy lekérések fognak történni, amelyek az adatbázisban már biztosan meglévő versenyzőket fog kezelni. Itt a végpontok hívásánál tisztában kell lennünk a versenyző azonosítójával, de egy esetben – mint látni is fogjuk – elég csak a nevét tudni.

Az első végpont PUT módszerrel lehet meghívni (`/driver/{id}`). Lényege, hogy már egy meglévő versenyző adatait módosítsuk. Tisztában kell lennünk a versenyző azonosítójával, amit az URL-ben kell megadni. A többi adatot, hogy milyen módon szeretnénk módosítani a versenyzőt, a bodyban kell megadni JSON formájában. Ennek magában kell foglalnia a „name”, „realName”, „nationality”, „number” és „actualTeamId” adattagokat. A program először megnézi, hogy a driver objektum létezik-e ezzel az azonosítóval, majd azt, hogy a felhasználónak van-e engedélye ehhez a művelethez, a végén pedig ellenőrzi, hogy a szezon nincs-e archiválva. Ezután végrehajtja a módosítást a DriverService.cs megfelelő objektumának meghívásával, amennyiben a csapat létezik, amit a bodyban megadtunk.

A második végpont DELETE módszerrel rendelkezik és az URL paraméterként átadott azonosítóval rendelkező objektumot fogja törölni (`/driver/{id}`). Nem létező objektum esetén 404-es hibát kapunk. Majd megnézi, hogy van-e engedélyünk, a végén pedig végrehajtja a törlést a DriverService-en keresztül a DbContextben. Amennyiben van a versenyzőhöz tartozó eredmény, akkor azok is törlésre kerülnek.

Ebben a fájlban az utolsó végpont GET módszerrel a statisztikákat adja vissza egy JSON keretben belül. (`/driver/statistics/{name}`). Mint látható, ehhez az URL-ben kell megadni a nevet paraméterként. Amennyiben ilyen nevű versenyző nem létezik, úgy a program ezt jelezni fogja egy 404-es hibakód kíséretében. Sikeres lekérés esetén az alábbi formátumban fogjuk visszakapni az adatokat.

```

{
  "name": "leventenyi",
  "numberOfRace": 2,
  "numberOfWin": 1,
  "numberOfPodium": 1,
  "numberOfChampion": 0,
  "sumPoint": 25,
  "seasonStatistics": [
    {
      "name": "F1 League",
      "teamName": null,
      "teamColor": "#000000",
      "createdAt": "2023-05-14T16:03:43.84265",
      "position": 1
    }
  ],
  "positionStatistics": [
    {
      "position": "1",
      "number": 1
    },
    {
      "position": "DSQ",
      "number": 1
    }
  ]
}

```

13. ábra: statisztika végpont visszatérési értéke

3.2.2. Favorite

Az ebben szereplő vezérlőben nem volt más feladat, csak hogy egy kedvenc ligát – felhasználó párost hozzáadjunk az adatbázishoz, illetve a felhasználó igénye szerint ezt visszavonjuk, így erre elég volt két végpont.

Az előbbi végpontunk POST módszerrel fog rendelkezni. Az URL-je: `(/favorite/{seasonId})`. A szezon azonosítóját feltétlenül meg kell adni az URL-ben. Figyelni kell rá, hogy a szezon létezzen ezzel az id-val, különben hibával fog visszatérni. Fontos, hogy a felhasználó bejelentkezett állapotban legyen, illetve ne létezzen még a kedvencei között az adott liga.

A másik végpontunkat DELETE módszerrel lehet elérni. `(/favorite/{id})`. Fontos, hogy a be legyen jelentkezve hozzá és legyen jogunk törölni az azonosítóval rendelkező kedvenc relációs kapcsolatot, amennyiben az létezik.

3.2.3. Permission

Ahogy az már többször említésre került, a ligákban lehet szerepeket kezelni (moderátor, admin), fontos tehát, hogy ezeket végpontokon keresztül el tudjuk érni. Egy ligához hozzá lehet adni új moderátort, promótálni lehet, illetve törölni is.

Az első ehhez kapcsolódó végpontunk a `PermissionController.cs` fájlban belül a `PUT` metódus (`/permission/{id}`). Mindezt az URL-ben lévő azonosító alapján tudjuk megtenni, ami a liga és a felhasználó közötti relációra mutat rá. Amennyiben ilyen rekord nem létezik, vagy nem rendelkezünk admin jogkörrel, vagy admin role-al rendelkező elemet szeretnénk előléptetni, esetleg épp nincs jogunk a művelet végrehajtásához, abban az esetben a művelet sikertelen lesz.

A másik lehetőségünk törölni a jogot, hogy a felhasználó ne férjen hozzá a ligákban való műveletek végrehajtásához. Erre a következő végpont szolgál `DELETE` metódussal: (`/permission/{id}`). Az azonosítót az URL-ben kell megadni. A helyzet ugyanaz, ami az előző végpontnál is, ezekben az esetekben szintén egy hibaüzenetet várhatunk.

3.2.4. Race

Az előző fejezetekhez hasonlóan itt is fontos, hogy a ligán belül a már létező elemeket kezelni tudjuk végpontokon keresztül. Jelen esetben a versenyeket fogjuk átnézni.

Az első esetben a verseny részleteit tudjuk módosítani `PUT` metódussal. Az URL a következő, ahol az azonosítónak kell szerepelnie: (`/race/{id}`). A bodyban egy JSON-on belül kell megadni az új nevet stringként és a versenyhez tartozó időpontot dátum-idő formátumban. Előfordulhat, hogy a verseny a ligán belül ilyen névvel létezik, illetve fontos, hogy rendelkezünk legalább moderátor joggal a végrehajtáshoz, ahogy az is, hogy a liga létezzen ezzel az azonosítóval, más esetben hibát fogunk kapni.

A törlés a másik lehetőség ennél az elérési útnál és hasonlóan az előzőhöz `DELETE` metódussal rendelkezik és az URL-nek kell tartalmaznia az id-t: (`/race/{id}`). Ezt csak bejelentkezett állapotban, a ligán belül moderátori jogkörrel tudjuk végrehajtani, amennyiben létezik elem ezzel az id-val. Fontos tudni, hogy a végrehajtás során a hozzátartozó összes eredmény törlődni fog.

3.2.5. Result

A ligákon belül a versenyeket is kezelni kell. Ez a csoport fogja tartalmazni a hozzáadást, módosítást, törlést.

POST metódus segítségével elérjük a hozzáadás végpontját az alábbi URL-en: (`/result`). Itt a body-ban kell megadni JSON formátumban a következőket:

- type: string (értékei: Finished, DNS, DNF, DSQ)
- position: integer 0-99
- point: double
- driverId: Guid (versenyző azonosítója)
- teamId: Guid (csapat azonosítója)
- raceId: Guid (verseny azonosítója)

A végpont abban az esetben fog hibát dobni, amennyiben nincs legalább moderátori jogunk az adott szezonban, vagy már létezik ezzel a versenyzővel az adott versenyen legalább egy record, vagy nem létezik valamelyik adat, illetve az is előfordulhat, hogy a csapat, a versenyző vagy a verseny nem tartozik ugyanazon liga alá.

A versenyeken elért eredményeket természetesen módosítani is lehet PUT metódussal (`/result/{id}`). Az URL-ben megadott azonosítójú elemet amennyiben a program megtalálta, akkor hasonló body-t megadva, mint az előző metódusnál, sikeresen tudjuk módosítani az eredményt. Fordított esetben hibára futhatunk.

DELETE metódus itt is működik, amennyiben egy eredményre már nincs szükségünk. A fontos dolog, hogy helyes id-t adjunk meg és legyen megfelelő jogunk ehhez a ligán belül. (`/result/{id}`).

3.2.6. Season

Ebben a csoportban tudjuk végrehajtani a szezonokhoz tartozó adatmanipulációkat, illetve részletesebb lekéréseket tudunk tenni egy-egy ligáról.

Az első ilyen végponttal GET metódussal le tudjuk kérni a létező összes ligát az adatbázisból (`/season`). Ez tartalmazni fogja minden ligának a fontosabb adatait, mint például a név, leírás, létrehozási dátum, archiválva van-e, hányan jelölték kedvencnek, azonosítót is tartalmazni fogja, illetve egy lista keretein belül megmutatja, hogy ki az admin, és kik a moderátorok és ezeknek a regisztrált felhasználóknak a részleteit. Az 1-

es szám az admin-t fogja jelölni, míg a 0, a moderátorra mutat rá. Ezt a végpontot bárki futtathatja akár bejelentkezés nélkül is.

```
{
  "id": "ef87fc1a-aad7-4835-a80d-25178f418cc1",
  "name": "Formula 1 2021",
  "description": "This is the results of 2021 Formula 1 season",
  "isArchived": false,
  "createdAt": "2021-03-28T14:00:00",
  "favorite": 1,
  "permissions": [
    {
      "id": "1d4ad3b9-7d39-4e2e-a3ca-9e417f76d450",
      "userId": "08db26a9-9264-4fb6-88aa-4c547e6326dc",
      "username": "test1",
      "type": 0
    },
    {
      "id": "fb2e3664-587b-4c9a-b080-a33be7f95bf2",
      "userId": "08db26a9-840c-42ee-82c5-ceec14c2a104",
      "username": "leventenyiro",
      "type": 1
    }
  ]
}
```

14. ábra: season végpont visszatérési értéke

Következő végponttal egy teljesen új szezont tudunk hozzáadni az adatbázishoz POST metódus segítségével (`/season`). Ehhez fontos, hogy be legyünk jelentkezve a regisztráció után, mert csak ezáltal lehet valakihez kapcsolni a ligát. Két adatot kell a JSON-nek tartalmaznia: `name` – vagyis a liga neve. Ez az adattag nem lehet üres, azonban a második adattagot hagyhatjuk üresen, ez lesz ugyanis a liga részletesebb leírása, mégpedig a `description`. A többi adatot a végpont a végrehajtás során fogja megadni, így például a `permission`ot, `archive` adattagot, `createdAt` nevezetű timestampet. A bejelentkezett felhasználó adatait a program automatikusan le tudja kérni, így bejelentkezés után csak a fenti adatokra van szükség a létrehozáshoz.

A következő végpontunk segítségével egy liga részletes adatait tekinthetjük meg. Ez a programnak az egyik lényegi részét szedi össze: versenyzőket, versenyeket, csapatokat, a versenyeken belül pedig eredményeket is tartalmaz. GET metódussal érjük el (`/season/{id}`). A versenyzőnek megkapjuk a felhasználó nevét, valós nevét, rajtszámát és az aktuális csapatát. A versenyeknek megkapjuk a nevét és az időpontját, az eredményeken belül pedig az eredmény típusát, elért pozícióját, pontját, illetve

tudomást szerezhethetünk arról is, hogy melyik versenyzőhöz és csapathoz tartozik mindez. Amennyiben a type nem „Finished” értéket vesz fel, abban az esetben a pozíció null értékkel fog visszatérni.

Ha szezon adatait szeretnénk módosítani, azt a PUT metódus segítségével tehetjük meg (`/season/{id}`). Az URL-ben megadott id ellenőrzése után a body-ban megadott JSON által fogja módosítani az adatokat. A „name”, „description” és „isArchived” adattagokat kell itt megadni. Amennyiben nem létezik a szezon, a név hiányzik, JSON helytelen, nem vagyunk bejelentkezve vagy nincs jogunk a módosításra, akkor hibát fogunk kapni. Archivált szezont nem tudunk módosítani, a vezérlőnk ezt is ellenőrizni fogja. Fontos kiemelni, hogy szezon adatokat csak admin jogosultsággal lehet kezelni, moderátornak erre nincs felhatalmazása.

Szezont a fentiekhez hasonlóan a DELETE metódussal és az URL-ben megadott azonosítóval lehetséges (`/season/{id}`). Itt is ellenőrizve lesz a szezon és a megfelelő jogosultság megléte. Amennyiben töröljük a szezont a hozzá tartozó összes adat is törlődni fog, így a versenyzők, csapatok, versenyek és az eredmények is.

A szezont archiválni és feloldani is lehet egy PUT végponttal. Itt csak az azonosítót szükséges megadni az URL-ben (`/season/{id}/archive`). A szezon megléte és az admin jogosultság előfeltétele a végrehajtásnak.

Van egy GET végpont, amely által a bejelentkezett felhasználó által menedzselte ligákat lehet megtekinteni, mind adminisztrátori, mind moderátori jogosultsági szinten. Ehhez csak a végpontot (`/season/user`) kell meghívni, illetve fontos, hogy a felhasználó be legyen jelentkezve, mert így lehet csak lekérni ezt az adatot.

A következő bekezdésekben csak hozzáadásról lesz szó POST metódusban, amellyel ligához tartozó adatokat lehet megadni. Fontos megjegyezni, hogy a ligának nem szabad archivnak lennie a következő metódusok végrehajtásához. Az első ilyen a jogosultság hozzáadása (`/season/{seasonId}/permission`). Itt a szezon azonosítóját feltétlenül meg kell adni és annak egy létező objektumra kell hivatkoznia, amelyre van adminisztrátori jogosultságunk. A body form-jában egy adattagot kell megadni: usernameEmail. Ez tartalmazni fogja vagy a felhasználónevet vagy e-mail címét. Amennyiben ez egy nem létező felhasználóra mutat rá vagy a felhasználó már hozzá van adva a ligához moderátorként, akkor nem fog teljesülni a végrehajtás.

Hozzá lehet természetesen adni versenyzőt is a ligához (`/season/{seasonId}/driver`). Itt arra van szükség, hogy a ligához - ami jó esetben létezik - legalább moderátori joggal rendelkezünk és ez a versenyző még nem létezik ezen a ligán belül. A body-ban JSON formájában kell megadni a szükséges adatokat, így a versenyző módosításánál hasonlóan az induló nevét, teljes nevét (opcionálisan), nemzetiségét (lehet null is), rajtszámát (1-99) között, illetve az aktuális csapatát (lehet null is). Amennyiben ez a név vagy rajtszám már szerepel a ligában, vagy a rajtszám túlmutat a határokon, esetleg a megadott csapat nem létezik, vagy nem ebben a ligában, esetleg nem valós a nemzetiségi kód, úgy a végrehajtás sikertelen lesz.

A következő eset az új csapat hozzáadása (`/season/{seasonId}/team`). Itt is meg kell adni a létező szezon azonosítóját, amelyhez moderátori jogosultsággal rendelkezik a felhasználói fiókunk, illetve JSON formában meg kell adni a nevét a csapatnak és a színét HEX kódban, amennyiben létezik már ilyen nevű csapat, vagy helytelenül adtunk meg valamilyen adatot, úgy hibára fogunk jutni.

Az utolsó végpont a csoportban az új versenyek hozzáadására szolgál (`/season/{seasonId}/race`). Szintén a szezon azonosítójával kell kezdeni, amit a program ellenőriz, hogy létezik-e és a megfelelő (legalább moderátorszintű) jogosultsággal rendelkezünk-e. JSON fájl keretein belül meg kell adni a nevét a versenynek – fontos, hogy ez ne létezzen még a ligában -, illetve a kezdési idejét is meg lehet határozni opcionálisan dátum-idő formátumban.

3.2.7. Team

Ebben a csoportban fog történni a szezonban már létező csapatok kezelése. Hasonlóan a fentiekhez itt is minden végponthoz meg kell adni az URL-ben az azonosítóját a létező csapatnak és fontos, hogy ahhoz a szezonhoz, amelyhez a csapat tartozik, legalább moderátori jogosultságunk legyen, továbbá a szezon ne legyen archiválva.

A csoportban szereplő első metódus PUT-ként érhető el (`/team/{id}`). Itt szerkeszthetjük a már meglévő adatunkat, így a body-nak tartalmaznia kell JSON formátumban a csapat nevét és a színekódját HEX formátumban. Amennyiben valamelyik adat helytelen, arról a program értesíteni fog.

A másik endpoint DELETE metódussal rendelkezik (`/team/{id}`). Arra szolgál, hogy a szezonból az adott csapatot törölje. Amennyiben töröljük, vele együtt az összes hozzá tartozó eredmény is törlődni fog és a versenyzők aktuális csapata is null-ra állítódik, ahol az épp törölt verseny szerepelt.

3.2.8. User

Az imént szereplő fejezetben az utolsó – egyébként jelentős - végpontcsoportról lesz szó, amik mind a felhasználókezelés meghatározó lépéseit tartalmazzák.

Az első végpont POST metódussal rendelkezik és a bejelentkezéshez szolgál (`/user/login`). A végpont a body-ban várja JSON formátumban az „usernameEmail” és a „password” adattagok meglétét. Az előbbi tartalmazhatja a felhasználónevet vagy az e-mail címet. Hiányzó adat esetén nem fog lefutni a végrehajtás. A program megkeresi a felhasználót, majd a jelszóval együtt az objektumot elküldi a szervíznek, ahol összeveti a jelszavakat, majd az alapján, hogy helyes-e a jelszó visszaküld egy tokenet. Mi ezt a kulcsot megkapjuk a végpontban és „Bearer” token formájában végezhetünk el vele belső műveleteket. Amennyiben valami nem sikerült, arról a program világos üzenetet fog küldeni.

A másik POST végpont a regisztrációhoz fűződik (`/user/registration`). Itt négy darab adattagot kell megadni JSON-ben: „username”, azaz a felhasználó neve, „email”, „password”, aminek meg kell felelnie a már említett formátumnak (kisbetű, nagybetű, szám, legalább nyolc karakter), illetve „passwordAgain”, ami a jelszót kéri be újra, hogy összehasonlítva azt a felhasználó biztosan az általa megadott jelszót használhassa. A program itt sok mindent ellenőrizni fog, kezdve a felhasználónév helyességéről (legalább öt karakter), e-mail cím megszokott formátuma, jelszó helyessége, jelszavak összehasonlítása, végül a felhasználónév, illetve a jelszó egyedisége. Ha ez mind megfelelt, a program hozzáadja az adatbázishoz az imént létrehozott felhasználót.

Szerepel egy végpont GET metódussal a csoporton belül (`/user`). Ez szolgál arra, hogy a felhasználó megkapja a saját felhasználói adatait. Ehhez a művelethez elengedhetetlenül szükséges, hogy bejelentkezett állapotban történjen a végrehajtás, különben 401-es hibát kapunk. Normális esetben viszontláthatjuk az azonosítónkat,

felhasználónevünket, e-mail címet és egy listában a kedvenc ligáinkat tartalmazva a relációs azonosítót és a szezon azonosítóját.

A következő végpont PUT metódussal érhető el (`/user`). Ennek a segítségével módosíthatjuk a felhasználói alapadatainkat, így a felhasználónevet és a jelszót. Fontos, hogy a művelethez be kell jelentkezünk, illetve minden adatot hiánytalanul és a formának megfelelően kell megadni a JSON-ben. Megadhatjuk ugyanazokat az adatokat, amik eddig voltak. A program figyel rá, hogy bár ez létezik az adatbázisban, de azok a mi adataink voltak, azonban más adatok megadása esetén szigorúan ellenőrizve lesz, hogy az adatok léteznek-e már.

Nem csak az alapadatokat lehet módosítani, hanem a felhasználói profilunknak a jelszavát is egy külön PUT végpont keretein belül (`/user/password`). Mivel ez egy külön munkafolyamat, így az volt a legcélszerűbb, ha ezt a két folyamatot két metódussá szedem szét. JSON formátumban három adattagot vár a program: „passwordOld”, ami a régi jelszót várja, „password” és a „passwordAgain”, ami az új jelszót és annak ismétlését szeretné elkérni. A program először azonosítja a bejelentkezett felhasználót, majd a megadott régi jelszó alapján, megnézi, hogy biztosan helyes jelszót adott-e meg. Ha minden stimmel, akkor továbbküldi a folyamatot a szervíz rétegre majd ott összehasonlítja az új jelszót és annak ismétlését, illetve a jelszó formátumát is ellenőrzés alá vonja. A végén, ha a program nem dob hibát, lefutott a módosítás és a felhasználó már az új jelszóval jelentkezhet be.

Az utolsó végpont az egész csoportban és a programban is egyaránt a felhasználó végleges törlésére szolgál DELETE metódusban (`/user`). Itt nem kell semmit megadni, az egyetlen fontos elvárás, hogy a felhasználó bejelentkezett állapotban legyen. Fontos kiemelni, hogy amennyiben a felhasználó ezt a lépést megteszi, úgy a hozzátartozó összes ligája törlődni fog, ahol ő korábban admin volt. A szezonnal együtt elvesznek a csapatok, versenyzők, versenyek és eredmények, illetve a „favorite” és „permission” táblából törlődni fognak a hozzá tartozó adatok. A folyamat irreverzibilis, így meg kell gondolni, hogy mikor élünk ezzel a végponttal.

3.3. Kliensoldal

A szoftverfejlesztés piacán számos kliensoldali technológiából tudunk válogatni. Három népszerűt tartunk számon. Angular a Google fejlesztésében, React,

amelyet a Facebook, immáron Meta csapata fejleszt és a VueJS, ami mögött nincs nagyobb cég csak a közösség, amely a fejlesztési folyamatokat diktálja. A szakdolgozatom elkészítéséhez az előbbit választottam, ugyanis van egy olyan pozitív tulajdonsága, hogy a fájlok szeparáltak vannak a projektben, így a html, scss, typescript (logika) külön-külön fájlokban szerkeszthetők. Ennek is megvan a hátulütője, mégpedig sok fájl esetén a fájlstruktúrát átláthatatlanná teheti mindez. Érdemes odafigyelni fejlesztés közben a „clean coding”-ra, azaz az átlátható kódolásra, így elkerülhetővé válhatnak az előbb említett kényelmetlenségek.

A lényegi rész a „src/app” mappában található. Négy fő részből épül fel a webapplikáció: komponensekből (components), modellekből (models), oldalakból (pages) és szolgáltatásokból (services) épül fel. Az ezeken kívüli rész szintén fontos darabjait képzik a webapplikációnak. Ilyen például az „app.module.ts” fájl, ahol a komponensek neveit és a webapplikációba szánt importokat adjuk meg, illetve az „app-routing.module.ts” fájl is, ahol pedig azt határozza meg a program, hogy egyes URL címeknél, melyik komponenseket nyissa meg az oldal – specifikusabban fogalmazva, melyik komponens szerepeljen az „app.component.ts” fájlban. Az „app.component.ts” a magja a látható oldalnak. Ebben a fájlban, amit írunk kódot, mindenhol látszódnia fog. Fontos kiemelni, hogy ez egy komponensalapú technológia, így az egyes komponensek egymásba lehetnek és vannak is ágyazva. Jelen esetben a főfájl egy komponenst vár, amit a „routing” fájl fog meghatározni az URL szerint.

A mappákat tekintve érdekesebb először a „models” mappáról említést tenni. Ez az alapja azoknak a műveleteknek, amiket a felhasználó a weboldalon használni fog. Kivétel nélkül ugyanazok a modellek jelennek itt meg, amiket a szerveroldal is használ modellként vagy akár DTO formájában. Arra szolgál ez, hogy az adatlekérések, illetve adatmanipulációk során objektumokat létrehozva tudjuk a végpontnak átadni a kívánt adatokat.

A következő nagyobb mappa a „pages” névre hallgat. Ezek olyan komponenseket fognak tartalmazni, amelyek teljes terjedelmükben megjelennek az oldalon, illetve elérhetőek a „routing” segítségével. Ide tartozik a „login”, „not-found”, ami abban az esetben jelenik meg, ha rossz oldalra navigáltuk magunkat, a „registration”, „season”, ami egy szezon adatait írja le, „seasons”, ami listázza az eddig létező ligákat,

„settings”, ami a felhasználó beállításaira szolgál, illetve a „statistics”, ami a versenyzők statisztikáit készíti el.

A következő ilyen a komponensek, ezek olyan elemek, amik részben jelennek meg egy oldalon, az oldalaknak úgymond építőkövei. Ilyen a „header”, ami a fejléce az oldalnak és a navigációs lehetőségeit segíti elő a felhasználónak, „modal”, ami egy ablak, ahova bármilyen adatot helyezhetünk a megadott paraméterek segítségével, „season-form”, ahol általánosítani lehetett fejlesztés szempontjából a szokásos szezon input formátumot, illetve azért is volt érdemes külön komponensbe szervezni a többi inputtal szemben, mert ezt az űrlapot több helyen is meghívja a program. Ez utóbbit általában modal keretei közt fogjuk megtalálni a webapplikációban. Itt van még a „verify-form” komponens is, ami a megerősítést szolgálja egy-egy művelet előtt és a „tables” mappán belül a hat darab tábla, ami egyes szezonokban mutatja meg a szezon jelenlegi állását (ezeknek a részletezését [2.3.4.](#)-es pontban találjuk meg).

Az oldalak és a komponensek kódjain belül megtalálhatunk input paramétereket, amelyek azt a célt szolgálják, hogy a komponens azokkal az adatokkal dolgozzon, jelenítse meg, vagy akár végezzen el egy lekérést, ilyen a „modal” komponensnél fordul elő a „components” mappán belül. Ebben az esetben egy feliratot adunk át, hogy az előugró ablak a szülő komponensben átadott értéket jelenítse meg. Ilyen lehetőség az egymásba ágyazás is, amikor megadhatjuk, hogy egy gyermek komponensen belül milyen html-beli elemek jelenjenek meg. Ilyen a megerősítő űrlapoknál fordult elő, amikor mindezt egy előugró ablak kereteiben adtuk meg, ezáltal az előugró ablak komponense többször újrahasználatóvá vált, így megadva a komponensalapú programozás lényegét.

Az utolsó figyelemre méltó mappa a szolgáltatásokat tartalmazó „services” mappa. Itt találjuk meg azokat a metódusokat, amelyek közvetlenül a szerveroldallal kommunikálnak. Két darabot találunk. Az első az „auth.service.ts”. Ez a felhasználókezeléssel foglalkozik magába foglalva a regisztrációt, bejelentkezést, jelszóváltoztatást. A másik lekérésekkel foglalkozó szolgáltatás a „season.service.ts” nevet viseli. Itt történik a programunk lényegi része. Ez kommunikál azokkal a végpontokkal, amelyek a szezonok létrehozására, törlésére, módosítására szolgál, ugyanúgy, ahogy ligákban a versenyzőket, versenyeket, csapatokat és eredményeket is

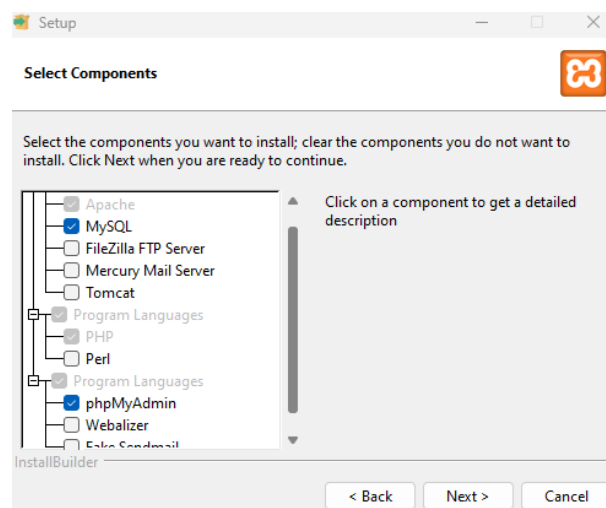
kezeljük. A kedvencek és a jogosultságok manipulációja is ezen osztály keretei között zajlik.

Egy ilyen projektnél fontos, hogy ügyeljünk a rétegek különválasztására, hogy megmaradjon annak logikus felépítése, hogy egy új fejlesztő hamar be tudjon kapcsolódni a munkákba.

3.4. Telepítés

A teljes program három komponensből épül fel. Mindezt egy gépen el lehet futtatni, azonban más-más gépeken is el tudják érni egymást a komponensek.

Ha saját gépre szeretnénk telepíteni, arra elsősorban az adatbázis telepítését javaslom mégpedig MySQL adatbáziskezelővel. Mindezt más technológiák keretein belül is lehet használni, ahogy azt már említettem, akár MSSQL-re is az Entity Framework segítségével. A MySQL-t személy szerint a XAMPP program segítségével telepítettem. A telepítés során elég nekünk a MySQL-t és a phpMyAdmin-t bepipálni. Ha ezzel végeztünk, megjelenik a Control Panel, ahol el kell indítani az Apache-ot és MySQL-t a „Start” gomb segítségével. Ezáltal már van egy működő adatbázisunk.



15. ábra: XAMPP telepítése

A következő lépésnél fontos a .NET 6 megléte. Itt az adatbázist migrálása történik. Ehhez már van egy megírt futtatható állomány „run.sh” néven, amit a következő lépéssel lehet futtatni bash környezetben: `sh run.sh init`. Így az Entity Framework Core is települni fog az adatbázis tábla feltöltése előtt. Amennyiben a szoftver elindítása mellett döntünk, úgy a „car-racing-tournament-api” mappába lépve a

konzolban ki kell adni a `dotnet run` parancsot. Ez a 7157-es porton fogja elindítani a programot.

Az utolsó lépés a Node alapú Angular webapplikáció telepítése és indítása. Fontos ehhez, hogy a gépen telepítve legyen az előbb említett két technológia. A telepítéshez be kell lépünk a „car-racing-tournament-web” mappába a konzolban és ott kiadni a következő két parancsot: `npm install`. A 4200-as porton lehet majd megtalálni a webapplikációt.

3.5. Tesztesetek

Egy szoftver fejlesztése során elengedhetetlen, hogy a program teljes mértékben tesztelve legyen, hogy a kisebb-nagyobb problémák el legyenek kerülve. Ehhez két nagyobb módszer van kezdetben: vagy manuálisan tehetjük meg mindezt vagy automatikusan, előre leprogramozva egyes teszteseteket.

A manuális tesztelést tekintve a szoftver teljeskörűen tesztelve volt. Mind a végpontok, mind a webapplikáció rész is. Mind a kettő stresszteszt keretein belül is. Amikor készen voltam a fejlesztés nagy részével és a webapplikáció már az interneten is elérhetővé vált bárki számára, akkor ismerősi körömet kértem meg, hogy teszteljék a programot minden tekintetben. Ezt meg is tették és a tapasztalataikat, javaslataikat, egyes hibafaktorokat elküldtek nekem egy űrlapon keresztül, hogy azokra minél előbb megoldást találjak.

Fontos volt minden esetben tesztelni manuálisan a mobilnézetet is, ez biztosítja ugyanis a responzív nézet alapjait. Sok oldalon kellett erre figyelni, különös tekintettel a táblázatokkal rendelkező komponenseknél, mint például a szezonon belül. Nem lehetett minden adatot megjeleníteni a táblázatban mobilnézetben, különben veszélyeztette volna a tábla áttekinthetőségét. Ugyanígy a szezonok listájában is eltávolításra került egy-egy oszlop mobilnézetben. Azonban, ha minden információt meg szeretnénk tekinteni, akkor ezt elforgatott képernyővel megtehetjük.

A másik mód volt a leprogramozott tesztesetek a szerveroldalon. Ezzel is sok hibát lehetett kiküszöbölni, ugyanis a tesztek írása során vettem észre, hogy más az elvárt viselkedés és más a valóban megtörtént szituáció. A következő bekezdésekben egy osztályra vonatkozó tesztesetek összességét fogom bemutatni, mind egységteszt, mind integrációs teszt keretein belül. A többi osztályról röviden el fogom mesélni, hogy milyen

érdekességek léteznek egy-egy teszt esetében. Fontos kiemelni, hogy a teszteseteket NUnit könyvtár segítségével alkottam meg.

3.5.1. Result osztály egységtesztje

Az osztály, aminek a részletes teszteseteit szeretném bemutatni, „Result” osztálynak hívják. Az elején az egységtesztelést fogom részletezni. Ez arról szól, hogy amikor az adatbázis „Results” táblájához új rekordot rögzítek, minden egyes eshetőséget kezel-e a szolgáltatás szint mielőtt a sikeres lefutás előtt. Itt fel kellett tölteni a tesztadatbázist fantázia adatokkal, hogy az egyes tesztesetekhez adatokkal rendelkezzek.

Először tesztelni kell, hogy a szolgáltatási szintről sikeresen megkapjuk az elvárt adatokat. „GetResultByIdSuccess” a folyamat neve és ellenőrzi azt, hogy a visszatérés sikeres volt-e, nem kapott a program semmilyen hibaüzenetet és a megszerzett adatok az elvártnak megfelelőek.

Ezután a „GetResultByIdNotFound” következett. Itt egy olyan id-t ad át a program szolgáltatási szintnek, amit a program maga a futtatás pillanatában generál, de ilyen azonosítóval rekord nincs az adatbázisban. Ilyenkor az elvárt az, hogy a futás eredményessége hamis értékkel fog visszatérni.

Amikor a lekéréseket sikeresen tesztelte a program, itt az ideje, hogy az új adat rögzítésére legyenek meg a tesztesetek. Elsősorban a sikerességet nézzük meg, hogy az adat rögzül-e az adatbázisban, erre az „AddResultSuccess” szolgál. Első esetben egy normális pozícióval felparaméterezett entitást adok át a szolgáltatói szintnek, megnézem, hogy a futás sikeres volt-e, nincs hibaüzenet és a rekord bekerült-e az adatbázisba. A második esetben ugyanezt teszem meg, viszont itt már a „Type” adattagot megváltoztattam és null értéket adtam a „Position”-nek.

Mivel az adatbázisban egy versenyzőnek egy versenyen csak egy eredménye (Result) lehet, így ellenőrizni kell, hogy a program hogyan viselkedik, ha még egyszer megadnék neki egy eredményt ugyanazzal a „DriverId”-val és „RaceId”-val, de más eredményekkel. Ellenőrizni fogja a tesztosztály, hogy az eredményessége hamis a lefutásnak és fontos, hogy a helyes hibaüzenetet kapom vissza.

A következő esetben az érvénytelenül megadott pozícióra adott választ ellenőrzi a program „AddResultInvalidPosition” név alatt. Itt a „Type” adattagot Finished-re állítva egyik esetben megadom a számbeli nullát pozíciónak, majd null értékre állítom. Ebben

az esetben is ellenőrizni kell, hogy a program nem futott le, illetve az elvárt hibaüzenetet küldi-e vissza a szolgáltatói szint.

Elengedhetetlen azt is tesztelni, hogy mínuszpont esetén milyen módon reagál a program. „AddResultMinusPoint” néven hoztam ezt a tesztet létre. Megnézem benne, hogy a futás hamis értékkel ért véget, az elvárt hibaüzenetet kaptam és a hozzáadás sikertelenségét vizsgálom meg még úgy, hogy az adatbázisban továbbra is csak egy rekord áll rendelkezésre.

Az előző tesztethez hasonlóan itt is a pontnak az értékét tesztelem le, azonban ebben az esetben nézem, hogy a megadott pontszám eltér-e a 0.5-tel való oszthatóságtól. Mindezt „AddResultNotHalfPoint” hoztam létre és ugyanígy megnézi, hogy biztos nem jött-e létre az új rekord.

A következő tesztet a „AddResultWithAnotherSeason” nevet viseli, lényege, hogy a tranzferobjektumból megtalált verseny, versenyző vagy csapat objektum nincs benne ugyanabban a szezonban. Erre az esetben a metódusban létrehoztam egy tesztversenyzőt egy teszteredménnyel, amelyik egy virtuális szezon azonosítójára hivatkozik. Amint a program visszaadta, hogy ez valóban egy sikertelen végrehajtás, majd beállítom, hogy a csapat legyen idegen a többi objektumtól liga tekintetében és itt is azonosítom, hogy a végrehajtás sikertelen, az elvárt hibaüzenetet kapom vissza, illetve az eredményrekord nem jött létre.

Ugyanezeket, amiket a fentiekben teszteltem, ugyanígy nem csak a létrehozásnál kell tesztelni, hanem az adatmódosításnál is. A tesztsztyál végigtekint minden eshetőséget, amikor a program módosítani akarja az egyes rekordokat és visszaadja, hogy a végrehajtás nem sikerült, illetve az elvárt hibakódot kapjuk.

Az utolsó egységtesztelés a rekord törléséhez kapcsolódik. Itt csak az objektumot kell átadni és a törlés alapesetben mindig végre lesz hajtva, így csak a végrehajtás sikerességét, hibaüzenet hiányát és az adatszerkezet ürességét ellenőrzi a program.

3.5.2. Result osztály integrációs tesztje

Az egységtesztek után kell egy mód, amivel nem csak a szolgáltatási réteget tesztelem, ami direktben módosítja a rekordokat az adatbázisban, hanem kell egy másik típusú teszt, ami összességében teszteli a végpontokat, amelyek szigorúbban ellenőriznek

egyes eshetőségeket (például eredményhez tartozó ligával kapcsolatban). Mivel itt már bejelentkezett felhasználó kezelheti az adatokat, ezért fontos volt, hogy a tesztesztályon belül beprogramozzuk ezt az esetet. Ehhez készítettem el a „SetAuthentication” metódust, ami a beépített Claim osztály és a felhasználó azonosítója által bejelentkezettként menti el az adott felhasználót és ezután gond nélkül lehet tesztelni az egyet vezérlőbeli végpontokat.

Az első ilyen pont segítségével teszteli a program, hogy amennyiben nem létező versenyzonosítót adok meg a tranzferobjektumban, úgy a végpont visszadobja, hogy nem létező objektumról van szó. Ezt az esetet lekezelem mind versenyző, illetve csapat azonosítójánál is.

Fontos, hogy az eredményeket csak olyan felhasználó kezelhesse, akinek erre jogosultsága van, ezért az elején a tesztesetnek olyan felhasználót adtam meg, akinek semmilyen engedélye nincs a ligára nézve és általa adtam meg új eredményt – egyébként helyes adatokkal. Ebben az esetben a program szól, hogy nincs engedélye a felhasználónak.

Volt egy olyan feature a programon belül, hogy egy ligát archiválni lehetett. Amennyiben ezt megtette a felhasználó, úgy a liga adatait nem lehetett befolyásolni, mindaddig, amíg az archiválást fel nem oldotta az admin. Ezt az esetet muszáj tesztelni, hogy archivált ligánál valóban nem lehet-e új adatot megadni és a teszt sikeresen kimutatta, hogy ez egy hibás futásra vezet.

Miután teszteltem a hozzáadással kapcsolatos végpontot, elérkezett az idő a módosítással kapcsolatos metódusok tesztelésére. Amennyiben egy eredményt módosítunk, úgy a bemenő adatok mellett a módosítani kívánt eredmény azonosítóját is feltétlenül meg kell adni. Ehhez le kellett tesztelni, hogy mi történik abban az esetben, amennyiben a felhasználó nem létező eredmény azonosítóját adja meg. Ebben az esetben a program ellenőrzi, hogy az eredmény valóban nem létezik – az objektum nem található. Minden más tesztelési metódus teljes mértékben megegyezik a hozzáadással kapcsolatos tesztesetekkel.

A végén pedig elérkeztem a törléssel kapcsolatos tesztesetekhez. Itt már bemenő adatokkal nem kellett foglalkozni, csakis az eredmény azonosítójával. Azonosító tekintetében tesztelni kellett, hogy nem létező objektumra hivatkozva hogyan viselkedik a program, ilyenkor természetesen visszaadja az elvártak megfelelő nem létező

objektum hibát. Fontos ellenőrizni itt is, hogy a felhasználónak van-e engedélye a törléshez, abban az esetben, ha ez nem valósul meg, úgy a program az elvártak megfelelő „ForbiddenObject” hibát fogja visszaküldeni. A harmadik esetben pedig amint archivált ligáról van szó, abban az esetben a program visszaadja, hogy ez egy rossz lekérés, itt nem lehet törölni semmilyen eredményt.

3.5.3. További osztályok tesztjei

A többi osztályt, ahol valamilyen ligához tartozó adatot, vagy magához a ligához kapcsolódó adatmanipulációt végzünk, hasonló tesztesetekkel kellett ellátni, mint a Result objektumnak a tesztesetei. A jogosultságkezelés, archiválás, sikeres feltöltés mellett ellenőrizni kellett egyes osztályok adattagjainak elvárt vagy épp tiltott értékeit. Ezekről már fentebb írtam, hogy milyen értéket várnak el egyes adatok, a tesztesetekben tulajdonképpen ezek ellenőrzése zajlik. Ahogy integrációs tesztek keretei között is tettem a végpontokkal, úgy egységtesztelésnél is fontos volt a szolgáltatási szint minden egyes metódusát részletesen letesztelni. A felhasználókezelésnél három tesztosztályt is létrehoztam. Míg az első a bejelentkezéssel kapcsolatos eseteket kezeli, addig a másik kettő a regisztrációval, illetve a felhasználó adatainak módosításával kapcsolatos. Erre a jobb áttekinthetőség érdekében volt szükség, hogy a kód jól olvasható legyen a teszteseteknél is.

```
Passed! - Failed: 0, Passed: 183, Skipped: 0, Total: 183, Duration: 2 s
```

16. ábra: tesztesetek sikeres futtatása

3.6. Projektfejlesztési folyamatok

A fejlesztés és menedzselés folyamatai során ügyeltem arra, hogy a munka során minden transzparens és áttekinthető legyen, hogy a későbbiek folyamán bármikor visszakereshetővé váljon egy-egy lépés. Ha ilyen helyzetben volt részem, gond nélkül tudtam megnézni, hogy akkor milyen lépést miért úgy csináltam, ahogy a kódban szerepel. Ehhez nagy segítséget nyújtott a GitHub általi lehetőségek mind repository, mind CI-CD (azaz folyamatos integráció és szállítás), valamint projektmenedzsment formájában. Utóbbi kettő mindig is kedves téma volt számomra, ugyanis szeretem nyomon követni egy projektnek a fejlődését és azt lépésenként meghatározni akár egy ehhez hasonló zöldmezős szoftveren belül is.

A repositoryn belül „car-racing-tournament-api” nevezetű mappában található a szoftver szerveroldali, míg „car-racing-tournament-web” nevű mappán belül a kliensoldali része. Két fontos mappa van még ezen kívül, az egyik a „documentation”, ahol jelen dokumentum is található, illetve minden olyan információ, ami a fejlesztéssel kapcsolatos, míg a másik a CI-CD-t tartalmazza a szoftverhez: ennek a neve „github/workflows”. A következő fejezetben utóbbit fogom részletezni.

A fejlesztés során kell valami, ami ellenőrzi és később élesbe is helyezi az éppen fixen működő szoftverünket. Amikor új feature-t implementálunk egy újonnan létrehozott branch-ben azt később vissza kell vezetni a „main” ágra. Az ezáltal létrehozott ellenőrzési fázist nevezzük „pull request”-nek. Itt fut le a folyamatos integráció (CI) rész, ami ellenőrzi a szoftver helyességét az általunk beállított kritériumoknak megfelelően. Lehet szemantikailag és szintaktikailag is tesztelni a programot ilyenkor. Az ehhez hasonló technológiáknál elég csak a futtatást és a tesztelést kérni a GitHub Actions-tól, utólag láthatjuk, hogy sikeres-e volt a futás és ha nem, akkor mi okozta mindezt. Jelen esetben az előbb említett szolgáltatás lehetőségével éltem, hogy teszteljem a programot. Amikor a „merge” megtörténik, azaz a frissen implementált tulajdonság bekerül a „main” ágon lévő programba, akkor lefut a folyamatos szállítás (CD) is. Ennél a szoftvernél a GitHub Actions a Microsoft Azure-n lévő tárhelyemre helyezi el a szoftvert. Ehhez meg kellett adni a „GitHub Secrets”-ben a csatlakoztató kódokat, aminek a segítségével fel lehet tölteni a programot. A „Secrets” tartalmazza azokat a kódokat, ami egy külső fejlesztő számára el van tüntetve, nehogy visszaélhessen vele. Ezt a folyamatot egyébként nem csak feltöltéskor indíthatjuk el, hanem manuálisan is bármelyik pillanatban.

Van egy másik „pipeline” is, ami az adatbázis migrációra szolgál. Ez nem a főágra való feltöltéskor fog elindulni, hanem bármelyik pillanatban elindíthatjuk. Míg az előző bekezdésben tárgyalt futtatható állomány a „main.yml” fájlban található, ezt a „migration.yml”-ben tudjuk módosítani.

Egy programnál több szakasza van a fejlesztésnek, amelyeket több része kell bontani, hogy átlátható és strukturált legyen a folyamat, így könnyebbé lesz később meghatározni, hogy egyes lépésekre mennyi időre van szükség. Erre a GitHub Projects által nyújtott lehetőségeket használtam. Egy Kanban táblába helyeztem az elvégzendő feladatokat öt darab oszlopon belül. Ezek az oszlopok a következők: „Backlog”, ahova az ötleteket helyeztem el, „Todo”, ahol a fontos teendők találhatóak, „In Progress”-ben

éppen futó feladatokat találhatunk, „Q/A Validation”-ben jelenlévő feladatok éppen jóváhagyásra várnak, hogy a „main” ágba kerülhessenek, míg az utolsó a „Done” oszlop, ahol már a készenlévő feladatokat láthatjuk. Mindegyik „ticket” azaz feladat rendelkezik egy kóddal, amit jelezni kell a „commit”-ok során, hogy a fejlesztési folyamatokkal össze tudjuk kapcsolni a feladatokkal. Külön jeleztem az átláthatóság érdekében, hogy melyik technológiában kell éppen dolgozni egy-egy feladat során, ahogy azt is, hogy feladatról van-e éppen szó vagy esetleg egy hibát kell minél előbb orvosolni. Mindez érezhetően gyorsabbá, élvezhetőbbé és strukturáltabbá tette a fejlesztési folyamatokat.

3.7. Továbbfejlesztési lehetőségek

Rengeteg további kiindulópont van ebben a szoftverben, hogy merre lehetne a továbbiakban haladni. Erre a lehetőségre szerepelnek a jegyek a Kanban tábla „Backlog” oszlopában, ahogy azt az előző fejezetben említettem is.

Az egyik ilyen lehetőség, hogy lényeges volna a felhasználókezelésnél azonosítani a felhasználót, hogy biztos valós emberrel van-e dolgunk, vagy csak egy bottal. Erre általában e-mailt szoktak kiküldeni, aminek a segítségével a felhasználó aktiválni tudja a profilját a későbbi használatra. Amennyiben ezt nem teszi meg, akkor a létrehozott profil egy időkereten belül törlődne (pl.: egy hét).

Ha az e-mail küldés implementálva van, onnantól már lépésekre van a fejlesztés egy másik szakasza, mégpedig az elfelejtett jelszó visszaállítása, amit másképp nem lehet elképzelni ilyen lehetőségek nélkül. Az e-mail cím adott, ugyanis azt regisztrációnál a felhasználónak feltétlenül meg kell adnia.

További lehetőség, hogy képeket megadjunk a szezonokhoz, csapatokhoz, versenyzőkhöz, hogy ezáltal még látványosabb, még inkább érdekesebb összefoglalókat lehessen megtekinteni és az objektumokat is könnyebb így azonosítani.

A többnyelvűség implementálása is érdemes lépés volna egy ilyen szoftver során. Habár a felhasználók által megadott szövegeket (pl.: liga leírását) nem lehet lefordítani, de statikus szövegekkel ezt megtehetjük, ezzel is könnyítve az értelmezést az angol nyelvet nem ismerő felhasználók számára.

Versenyzőkkel kapcsolatos további fejlesztési ötletek is vannak, például a versenyző három betűs kódjának eltárolása vagy éppen a nemzetiségével is megtehetnénk ugyanezt. Ez is a látványosságát segítené elő a szoftvernek.

Felmerült egy olyan lehetőség is, hogy egy admin helyett több is lehessen, ezáltal egy bizalmi jogkört adni egy másik felhasználónak a liga létrehozója által. Mindegyik admin élhetne az eddig megszokott jogkörök tulajdonságaival, így szerkeszthetik a liga részleteit, törölhetik is azt és adhatnak hozzá, ugyanígy rúghatnak ki moderátorokat.

Van egy bonyolultabb és az előbbieknél is érdekesebb továbbfejlesztési lehetőség. Amennyiben valamelyik versenyző egy adott versenyen használ telemetriákat tartalmazó adatot, akkor azt API segítségével közvetítse a program fele, hogy ezáltal még érdekesebbé tegye a statisztikákat. A lépéseket ezzel kapcsolatban még nem dolgoztam ki, de egy későbbi fejlesztés keretében mindenképpen szeretnék foglalkozni a kérdéssel még a szakdolgozat keretein kívül.

4. Összegzés

Ebben a fejezetben össze szeretném gyűjteni az eddig elért sikereket a fejlesztés folyamata során, ahogy az ebben szerzett szakmai tapasztalataimat. Leírni, hogy mit érzek több hónap munka után, amit erre a számomra kedvelt projektbe szántam.

Volt régebben már egy-két nagyobb projekttel dolgom, úgyhogy új célokkal álltam neki ezen szoftver elkészítéséhez. A boldogságot az adta, hogy volt egy új fajta felépítése a programozási folyamatoknak. Ez volt a projektmenedzsment, ahol láttam az eredményét és az előre lépését a munkámnak minden egyes befejezett feladat után, illetve a CI-CD futtatások, amelyek azonnali eredménnyel szolgáltak egy-egy elvégzett feladat után.

Természetesen nem csak a repository által nyújtott zöld pipák nyújtották az elsődleges elégedettséget. Úgy éreztem mindvégig, hogy összehozok egy szoftvert, amivel kamatoztathatom a tudásomat kliens- és szerveroldal szintjén is, így tehát az Angular és .NET keretrendszerekben belül, ami később egy sablonként szolgálhat számomra, hogy egyes lépéseket milyen módon kell megtennem a következő jelentősebb projektemnél.

Sok nehézség és bukkánó akadt a fejlesztés során, ami jelentősebb időt és energiát emésztett fel, de úgy álltam ehhez hozzá, mint egy kutatómunkához. A végére a .NET és az Angular dokumentációs oldalán is otthonosan mozogtam és a korábbihoz képest még nagyobb önbizalmat adott számomra. Segítettek továbbá az internetes fórumok, ahogy az is, hogy egy-egy kisebb feladatot egy külön projektben valósítottam meg és „debugoltam”, hogy úgy hátha működésre bírom a „feature”-t. Amint sikerült emeltem is be a főszoftverbe. Megtanultam, hogy a fejlesztés során fontos a türelem és mellé a szorgalom, valamint a higgadt, józan gondolkodás a struktúrált, tiszta kódok kialakításához.

Összességében hálás vagyok ezért a lehetőségért, hogy utolsó félévemben egy általam választott ötlet fejlesztésének keretein belül bizonyíthatom a magabiztos tudásomat. Nélkülözhetetlen dolognak tartom, hogy egy ilyen feladattal megbirkózzon az ember ebben a szakmában úgy, hogy sokszor magára támaszkodik. Sok mindent köszönhetek viszont a tanárainknak, munkatársainknak, családtagjainknak, barátainknak és

legfőképp a témavezető tanáromnak az építőjellegű kritikákért, az őszinte szavakért, amelyek még inkább vitték előre a fejlesztés menetét és a bennem felmerülő szakmai kérdésekre választ kaphattam.

Forrásjegyzék

- [1] .NET Entity Framework Core Documentation. <https://learn.microsoft.com/en-us/ef/core/>, 2023. 01. 22.
- [2] .NET Documentation. <https://learn.microsoft.com/en-us/aspnet/web-api/overview/data/using-web-api-with-entity-framework/part-5>, 2023. 01. 22.
- [3] Authentication. <https://www.youtube.com/watch?v=HGIdAn2h8BA>, 2023.01.10.
- [4] Bootstrap 5.3 Documentation. <https://getbootstrap.com/docs/5.3>
- [5] W3Schools. <https://www.w3schools.com/>
- [6] Angular Documentation. <https://angular.io/docs>
- [7] Stack Overflow. <https://stackoverflow.com>
- [8] GitHub. <https://github.com/>
- [9] History of web. <https://www.simplilearn.com/what-is-web-1-0-web-2-0-and-web-3-0-with-their-difference-article>, 2023. 04. 19.
- [10] Secrets in GitHub Actions. <https://itbackyard.com/how-to-manage-secrets-in-net-locally-and-on-github/>, 2023. 04. 08.
- [11] Flags API. <https://hatscripts.github.io/circle-flags>, 2023. 05. 11.
- [12] Countries. <https://github.com/luke/ISO-3166-Countries-with-Regional-Codes>, 2023. 05. 11.

Ábrajegyzék

1. ábra: webapplikáció mobilnézetben.....	8
2. ábra: gépigény.....	9
3. ábra: bejelentkezési felület.....	10
4. ábra: regisztrációs felület.....	10
5. ábra: szezonok listája.....	11
6. ábra: szezon hozzáadása	12
7. ábra: szezon részletező oldala.....	13
8. ábra: statisztika oldal	17
9. ábra: felhasználói profil szerkesztése	18
10. ábra: adatbázis migráláshoz szükséges parancsok.....	20
11. ábra: Adatbázis ER-diagramja.....	21
12. ábra: UML diagram	26
13. ábra: statisztika végpont visszatérési értéke	29
14. ábra: season végpont visszatérési értéke.....	32
15. ábra: XAMPP telepítése.....	39
16. ábra: tesztesetek sikeres futtatása	44