

|  |
| --- |
| **EÖTVÖS LORÁND TUDOMÁNYEGYETEM**  **INFORMATIKAI KAR**  **PROGRAMOZÁSELMÉLET ÉS SZOFTVERTECHNOLÓGIAI TANSZÉK** |

**Webalkalmazás fejlesztése Java nyelvben a Spring keretrendszerrel**

Tartalom

**Bevezetés**

Szakdolgozatom témájának megválasztásakor a fő cél az volt, hogy egy olyan alkalmazást készítsek, ami olyan technológiák segítségével van megvalósítva, amit az iparban is használnak. A választásom ezért egy webalkalmazás fejlesztésére esett és mivel ez mellett rajongok a Forma-1-ért, ezért a kettőt ötvöztem és a webalkalmazás témája egy Forma-1 manager játék lett. A játék bárki számára elérhető, aki rendelkezik internet kapcsolattal. A felhasználóknak először regisztrálniuk kell, majd aktiválni az e-mail-ben kapott aktivációs kód segítségével. Ezután a játékos kap egy kezdőösszeget és vásárolhat az elérhető csapatok és pilóták közül, akik képviselni fogják a következő futamon és ez által a játékos pontot és pénzt szerezhet. A játék párhuzamosan fut a valós Forma-1 bajnoksággal. A pilóták és csapatok ára az alapján változik, hogy a futamon milyen helyezést értek el. A felhasználók saját ligákat hozhatnak létre, hogy a ligában szereplők helyezéseit minél gyorsabban láthassák. Minden futam között látható, hogy az előző futamokon a játékos mennyi pontot, pénzt szerzett vagy milyen pilótafelállással szerepelt. A programhoz tartozik egy adminisztrációs felület, ahol a játék adatait lehet beállítani, de ezt csak a megfelelő jogosultsággal rendelkezőek érhetik el.

A programozási nyelv és a szerveroldali keretrendszer kiválasztásakor a legfontosabb szempontok többek között a robosztusság, modularitás, megbízhatóság volt, ezért a Spring Framework mellett döntöttem. A felhasználói felület elkészítéséhez a Bootstrap és jQuery keretrendszereket választottam.

**Felhasználói dokumentáció**

Az alkalmazás futtatásához szükséges programok szerveroldalon:

* Java 1.7
* MySQL 5.5
* Tomcat 7 vagy másik alkalmazásszerver, ami képes Java servletek futtatására

Felhasználók számára

A felhasználóknak rendelkezni kell egy internethez kapcsolódó eszközzel, amelyen tetszőleges internetböngésző futtatható. A weboldal minden böngészőben működik és minden kijelzőn optimális méretben jelenik meg, azaz a weboldal kinézete reszponzív.

A legelterjedtebb böngészők közül a következőket ajánlom:

* Google Chrome
* Mozilla Firefox
* Safari

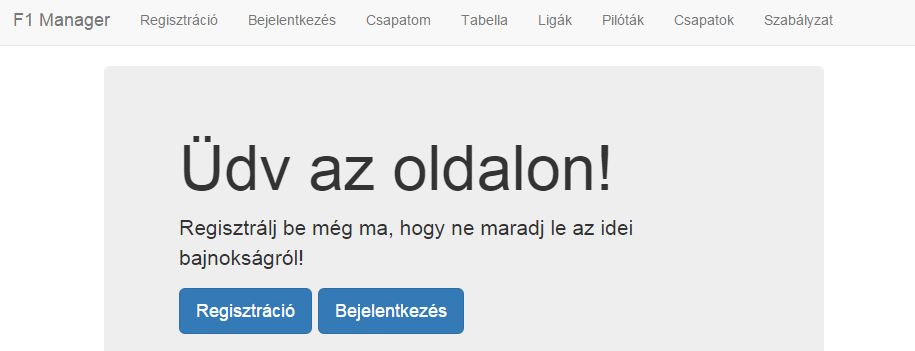
**Használati leírás**

A programot a böngészőben megnyitva a következő címen lehet elérni.

* http://128.199.38.8/Szakdolgozat/

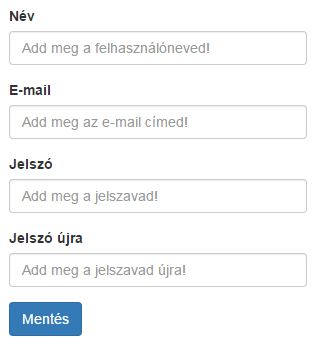
Az oldal betöltése után a felhasználót a következő oldal fogadja:

Bejelentkezés előtt lévő oldal (1. ábra):



1. ábra

**Regisztráció oldal:**

 Az oldal új látogatóinak lehetősége van az oldalra regisztrálnia, amit egy regisztrációs űrlap segítségével lehet megtenni (2. ábra).

2. ábra

A regisztrálás során szükség van egyedi (még nem regisztrált) felhasználónévre, egyedi e-mail címre és jelszóhoz a bejelentkezéshez.

Ha a felhasználónév vagy e-mail cím már szerepel a rendszerben, akkor a felhasználó azonnali visszajelzést kap erről még gépelés közben. Ha a jelszavak nem egyeznek vagy túl rövidek, akkor erről is jelzés érkezik.

Regisztráció sikeressége után szükség van a megadott e-mail cím megerősítésére. Ezt a megadott e-mail címre küldött hivatkozásra kattintva lehet megtenni. Ha az aktivációs e-mail elveszne, akkor a Bejelentkezés oldalon új igényelhető.

**Bejelentkezés oldal:**

Ezen az oldalon tudnak a felhasználók bejelentkezni a fiókjukba a felhasználónevük és jelszavuk megadásával. Ha a bejelentkezés sikeres volt, de a felhasználó még nem erősítette meg az e-mail címet, akkor ezt itt kell megtennie. Ha elvesztette az aktivációs kódot, itt lehet igényelni újat.

Ha a felhasználó elfelejtette a jelszavát, akkor a már regisztrált e-mail cím megadásával lehet újat igényelni. Ekkor a rendszer elküld egy levelet, ami tartalmazza, azt a hivatkozást ahol a felhasználó megváltoztathatja a jelszavát.

Az oldal a sikeres bejelentkezés után (3. ábra)



. ábra

**Csapatom oldal:**

Ennek az oldalnak a tartalmát a felhasználó csak akkor látja, ha már bejelentkezett.

Itt lehet megtekinteni, hogy jelenleg mennyi pont áll a rendelkezésére, a bajnokságban eddig mennyi pontot szerzett és az előző futamokon milyen felállással indult.

Ezen felül itt lehet eladni és vásárolni pilótákat, csapatokat. Összesen két pilótát és három csapatot lehet vásárolni. Nem lehet ugyanazt a pilótát vagy csapatot többször is megvenni. Ha ezt a felhasználó megpróbálja, akkor az oldal kiírja a felhasználó számára a megfelelő üzenetet. Továbbá a két pilóta nem tartozhat ugyanabba a csapatba.

**Tabella oldal:**

Ezen az oldalon tekinthető meg az összes regisztrált felhasználó az aktuális bajnokságban lévő helyezésükkel és további adatokkal.

Az oldal megnyitása után nem jelenik meg rögtön az összes felhasználó, mert ebből nagyon sok lehet. Egy oldalon a megjelent felhasználók száma fix szám. Ha az összes felhasználó száma túllépi az egy oldalon megjeleníthető felhasználók számát, akkor több oldalon keresztül lehet megtekinteni az felhasználókat. A felhasználónév mellett csak kevés adat jelenik meg ilyenkor, de a felhasználónévre kattintva megtekinthető a többi adat is. Ilyenkor látszik, hogy a felhasználó az előző futamok milyen eredménnyel szerepelt.

Lehetőség van felhasználónév alapján szűrni a lista tartalmát. Ilyenkor a lista tartalmában csak azok a felhasználók fognak szerepelni, akik neve tartalmazza a megadott karaktersorozatot.

**Ligák oldal:**

Bejelentkezés előtt csak az összes ligát lehet megtekinteni és egy liga nevére kattintva jön be az adott ligához tartozó oldal.

Bejelentkezés után már egy külön gombra kattintva lehet elérni azokat a ligákat, amikben a bejelentkezett felhasználó szerepel.

Egy harmadik gomb segítségével a felhasználó létrehozhat saját ligát is. Itt a ligának egyedi nevet kell adni, azaz nem lehet a rendszerben másik ugyanilyen nevű liga, ezen felül egy rövid leírást is kell adni. Amelyik felhasználó létrehozta a ligát az lesz a liga adminisztrátora. Ő nem léphet ki a ligából és joga van bármelyik felhasználót kirakni a ligából. A liga minden tagjának lehetősége van meghívni bármelyik felhasználót, aki még nincs benne az adott ligában. Ilyenkor név alapján tud rákeresni a felhasználókra és a talált felhasználókat meg tudja hívni e-mailen keresztül a ligába. Ilyenkor a rendszer elküld egy levelet a meghívott játékosnak, aki a levélben található hivatkozással tud csatlakozni a ligába vagy figyelmen kívül hagyhatja ezt.

**Pilóták és Csapatok oldalak:**

Itt lehet megtekinteni egy táblázatban, hogy az aktuális bajnokságban milyen pilóták és csapatok közül lehet válogatni, illetve itt látszódik, hogy a bajnokságban az adott versenyző vagy csapata mennyi pontot ért el és mennyibe kerül.

**Szabályzat oldal:**

Erre az oldalra kattintva lehet az oldalhoz tartozó részletes szabályzatot elolvasni, hogy az oldalt látogató új tagok is tudják, hogy mi a játék lényege és adott futamok alapján ki mennyi pontot és pénzt kap.

**Kijelentkezés:**

A bejelentkezés után a jobb felső sarokban elérhető kijelentkezés menüpontra kattintva az oldal kijelentkezteti a felhasználót és a kezdőoldalra továbbítja.

**Az oldalhoz tartozó adminisztráció felület:**

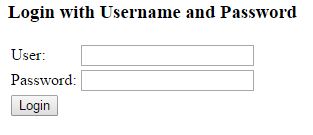
Az oldalhoz tartozó admin felületre csak a megfelelő felhasználónév jelszó párossal rendelkezőek tudnak bejelentkezni.

Egy funkció sem érhető el a bejelentkezés előtt, ha azonban ezt valaki mégis megkíséreli, akkor az oldal továbbít a bejelentkező oldalra.

Az admin felület a következő oldalon érhető el:

* http://128.199.38.8/Szakdolgozat/admin

**Bejelentkező felület:**



4. ábra

Ha a megfelelő felhasználónév és jelszót megadjuk a felületen, akkor az oldal továbbít az admin oldalra, különben hibát jelez és megkéri a felhasználót, hogy próbáljon meg újra bejelentkezni.

Ha a bejelentkezés sikeres volt, akkor a következő felület fogadja az adminisztrátort:

5. ábra

A felületen a következő funkciók érhetőek el:

**A csapatok menüpont esetén két dolog közül lehet választani:**

Az első választási lehetőség az összes csapat listázása. Ilyenkor egy táblázatban látható az összes csapat. A táblázat minden sorában szerepel egy „Szerkeszt” gomb, amire kattintva megnyílik az adott csapathoz tartozó adatokat szerkesztő oldal. Egy csapat esetén négy adatot kell megadni. A csapat nevét, árát, a hozzátartozó kép URL címét és azt,hogy az idei bajnokságban megvásárolható-e. A csapat nevének egyedinek kell lennie. Az árat a bajnokság elején érdemes beállítani, mert minden futam eredményének feltöltése után a rendszer automatikusan növeli vagy csökkenti a megfelelő mennyiséggel. A képhez tartozó mezőt üresen lehet hagyni vagy érvényes URL címet kell megadni, ami egy képhez tartozó elérhetőséget tartalmaz. Egy csapat nem törölhető helyette beállítható, hogy az aktuális bajnokságban a felhasználóak megvásárolhatják-e. Ennek az oka, hogy a rendszer menti a felhasználók régi eredményeit is, és bármikor visszanézhetik, hogy mikor melyik csapattal rendelkeztek attól függetlenül, hogy az aktuális bajnokságban indult a csapat vagy sem.

A második választási lehetőség az új csapat létrehozása. Ebben az esetben egy teljesen új csapatot vesszünk fel a rendszerbe. Ilyenkor a felvett adatokra ugyanazok a szabályok érvényesek, mint módosításkor.

**A pilóták menüpont hasonlóan épül fel, mint a csapatok menüpont:**

Itt is meglehet tekinteni a már meglévő pilótákat vagy újat lehet felvenni. A különbség a pilóták és csapatok között, hogy egy pilótához hozzá kell rendelni egy csapatot is. A csapat, amit szeretnénk beállítani a pilótának az egy listából választható ki.

**A harmadik menüpontban a pályákat lehet beállítani.**

A „Pályák listázása” menüpont alatt a már megszokott táblázatban láthatjuk a rendszerben lévő pályákat és itt választhatjuk ki, hogy melyiket szeretnénk szerkeszteni.

Új pálya felvétele esetén négy adatot lehet megadni. A pálya neve, aminek egyedinek kell lennie és legfeljebb 100 karakterből állhat. Ezen felül az ország és város nevét kell, még megadni ahol fekszik a pálya. Pályák esetén is lehetőség van kép felvételére, ilyenkor érvényes URL címet kell megadni.

**A következő menüpontban a futamokat lehet állítani.**

**Futamok listázásakor a következő információk láthatóak:**

A futam pontos dátuma, helyszíne és státusza, ami azt jelzi, hogy a futamhoz már lett eredmény állítva vagy még nem. Továbbá elérhető egy „Eredmény” gomb is, ami csak akkor jelenik meg, ha már lett állítva eredmény az adott futamhoz. A gombra kattintva lehet megtekinteni, hogy az időmérő edzésen és a versenyen ki milyen helyezést ért el a pilóták közül.

Új futam felvétele esetén három adatot kell megadni. Az első adat a futam időpontja, aminek egyedinek kell lennie, mert egy napon nem szoktak több versenyt szervezni. Két futam között 1-4 hét szokott eltelni egy bajnokságban. Továbbá a bajnokság évszámát és a pálya nevét ahol a futam megrendezésre kerül. Ezt a két adatot a már rendszerben lévő bajnokságokból és pályákból válogathatjuk ki.

**Bajnokság menüpont**

Egy bajnokság felvételéhez csak az adott évet kell megadni, aminek egyedinek kell lennie. A bajnokságok éve később is szerkeszthető, ha rosszul lett megadva.

**Eredmények:**

Ezen belül két menüpont érhető el:

Az első esetén egy adott verseny végeredményét állíthatjuk be. Ilyenkor kell beállítani, hogy melyik pilóta hol végzett az időmérőn és versenyen. A csapatokat nem kell megadni, mert minden pilótának tartoznia kell egy csapatba és a rendszer tudja, hogy melyik pilóta melyik csapatban van, ezért a megadott pilóták alapján beállítja a csapatokhoz tartozó értékeket is.

Ugyanaz a pilóta nem érhet el két helyezést egy időmérő edzésen vagy versenyen, ha mégis ilyen adatot próbálnánk menteni, akkor a rendszer figyelmeztet erre minket.  
Az eredményt figyelmesen kell beállítani, mert ez később már nem módosítható.  
Miután beállítottuk a helyezéseket és a „Mentés” gombra nyomunk akkor a rendszer több adatot is beállít. Először beállítja a helyezések alapján minden csapathoz és pilótához a szerzett pont és pénz mennyiségét. Utána ezen értékek alapján kiosztja a díjakat a felhasználók számára és utána kiszámolja, hogy a futam után melyik felhasználó hányadik a bajnokságban.

Hogy a felhasználók a későbbiekben is megtekinthessék mikor milyen felállással és milyen sikerességgel vettek részt a futamokon, ezért elmentjük ezeket az adatokat is.

**A másik menüpont a „Pontok beállítása”:**

Itt lehet egy táblázatban beállítani, hogy egy pilóta/csapat mennyi pontot szerez egy időmérőn/futamon elért helyezésért. Továbbá beállítható egy „százalék” érték is, ami azt a célt szolgálja, hogy a pilóta/csapat értéke a helyezéshez viszonyítva megfelelő mértékben növekedjen vagy csökkenjen.

**Excel menüpont:**

Ez az oldal arra a célra szolgál, hogy ha egy időben szeretnénk sok csapat tulajdonságát módosítani vagy új csapatokat felvenni, akkor minél gyorsabban és könnyebben lehessen kivitelező. Ehhez egy Excel fájl feltöltése szükséges, aminek megfelelő formátumban kell tartalmaznia a csapatokhoz tartozó adatokat. Az Excel-be ugyanazokat az adatok kell megadni, mint a weboldalon történő szerkesztés esetén. A csapat nevét, értékét, képét és a megszerzett pontok számát. Ehhez a fájlban hat oszlopot kell felvennünk. Az oszlopok első sorai a következőeknek kell lennie tetszőleges sorrendben: „id”, „name”, „price” , „picture”, „point” és „active”. Ha bármelyik oszlop nincs megadva vagy hibásan van gépelve, akkor a rendszer nem fogadja el a fájlt és erről tájékoztat minket. Az oldalról letölthető egy üres fájl, ami már tartalmazza ezt a struktúrát kialakítva, hogy a felhasználónak ne magának kelljen létrehoznia. Ha már a rendszerben szereplő csapatot szeretnénk módosítani, akkor az „id” oszlopban kell megadni a csapathoz tartozó azonosítót, hogy a rendszer tudja, hogy ez már egy meglévő csapat módosítása. Ha egy olyan azonosítót adunk meg, ami nem létezik az adatbázisban, akkor a rendszer erről szintén figyelmeztet. Új csapat esetén ezt a cellát a sorban üresen kell hagyni. Ha a csapathoz tartozó bármelyik érték nem felel meg a validálási szempontoknak (lásd: Csapatok menüpont részletezése), akkor a rendszer erről tájékoztatást ad. Az „active” oszlopba 0 vagy 1 érték írható. A 0 jelentése, hogy a csapat nem indult az idei bajnokságban, míg az 1 jelentése, hogy igen.

A harmadik opció esetében egy Excel fájl tölthető le, ami tartalmazza a rendszerben lévő összes csapatot a hozzátartozó adatokkal együtt.

Az új csapatok vagy már meglévőek módosítása előtt mindenképpen ajánlott letölteni valamelyik előre készített Excel fájlt.

**A kijelentkezés gomb:**

Erre a gombra kattintva a bejelentkezett adminisztrátor kitud jelentkezni a felületről és visszatérni az oldalhoz tartozó bejelentkezés oldalra.

**Fejlesztői dokumentáció**

Az alkalmazás felépítése

Az alkalmazás két fő részből áll, a szerver-kliens modellnek megfelelően

A szerveroldalon a Java nyelv dolgozik, míg kliens oldalon a html ,css és javascript.

A szerveroldalhoz a Spring framework-öt használtam. A keretrendszer 2002-ben jelent meg és jelenleg az iparban az egyik legnépszerűbb keretrendszer Java alkalmazások fejlesztéséhez. A Java EE szabvány hiányosságait próbálták vele helyettesíteni, ami sok esetben sikerült is, ezek később a standard szabványra is hatással voltak.

Az alkalmazás mögött lévő adatbázis manipulációhoz a Hibernate-re esett a választásom. Ez egy objektum-relációs (ORM) leképezést megvalósító programkönyvtár. Az ORM technológia nagy előnye, hogy az adatbázist osztályokon keresztül tudjuk kezelni és nincs szükség SQL utasítások írására, ezért a kód eltérő adatbázis-kezelő rendszerekkel is tud működni és rossz esetben is minimális módosításokra van csak szükség.

A szerver futtatásához szükség van egy alkalmazásszerverre is, amiből több fajta is létezik. A szakdolgozatomhoz a Tomcat alkalmazásszervert használtam, ami ingyenes és egyszerűen konfigurálható. Nagyobb szoftverekhez vannak részletesebben konfigurálhatóak is pl: Weblogic amit az Oracle fejleszt.

Sok funkcióval rendelkező szoftvereknél elengedhetetlen a tesztelés is. Ebben a JUnit és a Spring Test volt a segítségemre.

Továbbá szükség van arra is, hogy naplózik a rendszer hibáit is, ami alapján meg tudjuk keresni a hibákat és javítani tudjuk őket, ehhez a log4j-t használtam.

A dinamikus oldalak előállításához szükség van egy template engine-re is, ami arra a célra szolgál, hogy html kódot tudjunk dinamikusan generálni. Ehhez az Apache Velocity-t választottam, aminek a szintaktikája nagyon letisztult és a bővítése is egyszerű.

Az oldal kinézeténél fontos volt, hogy reszponzív legyen, mert jelenleg egyre több ember internetezik tablet és mobiltelefonról is, amihez az oldal kinézetének alkalmazkodnia kell, ezért a kinézet elkészítéséhez a népszerű Bootstrap keretrendszert alkalmaztam.

Továbbá a Git verziókezelő rendszert is használtam, ami többek között arra a célra is szolgál, hogy a kódnak bármelyik állapota visszanézhető, ami néhány esetben a segítségemre volt.

**Adatbázis**

Az alkalmazásban folyamatosan adatbázisban lévő adatok manipulációja történik.

A következő táblák és a hozzájuk tartozó entitások szerepelnek az alkalmazásban.

Az adatbázis 12 táblából áll, ezek a következőek.

**User tábla:**

Ebben a táblában találhatóak a felhasználó legfontosabb adatai

|  |  |  |
| --- | --- | --- |
| **Mező** | **Típus** | **Leírás** |
| id | bigint(20) | A felhasználó azonosítója |
| activated | bit | A fiók aktiválva lett-e az e-mailben kapott azonosító segítségével |
| activation\_code | varchar(255) | Az e-mailben küldött aktivációs kód |
| actual\_money | bigint(20) | A felhasználó aktuális pénze |
| actual\_point | bigint(20) | A felhasználó aktuális pontjainak száma |
| change\_password\_token | varchar(255) | Az elfelejtett jelszóhoz tárolt token |
| email | varchar(100) | A felhasználó e-mail címe |
| name | varchar(100) | A felhasználónév |
| registration\_date | date | A regisztráció időpontja |
| actual\_driver1\_id | bigint(20) | A felhasználó elsőszámú pilótája |
| actual\_driver2\_id | bigint(20) | A felhasználó másodszámú pilótája |
| actual\_team1\_id | bigint(20) | A felhasználó elsőszámú csapata |
| actual\_team2\_id | bigint(20) | A felhasználó másodszámú csapata |
| actual\_team3\_id | bigint(20) | A felhasználó harmadszámú csapata |
| password | varchar(255) | A felhasználó jelszava titkosítva, hashben tárolva |
| actual\_position | bigint(20) | A felhasználó aktuális pozíciója a bajnokságban |

**Team tábla:**

Ebben a táblában találhatóak a csapatok legfontosabb adatai

|  |  |  |
| --- | --- | --- |
| **Mező** | **Típus** | **Leírás** |
| id | bigint(20) | A csapat azonosítója |
| name | varchar(100) | A csapat neve |
| picture | longtext | A csapathoz tartozó kép URL címe |
| point | int(11) | A csapat aktuális pontszáma |
| price | bigint(20) | A csapat aktuális értéke |
| active | bit | A csapat részt vesz-e az aktuális bajnokságban |

**Driver tábla:**

Itt találhatóak a pilóták legfontosabb adatai.

|  |  |  |
| --- | --- | --- |
| **Mező** | **Típus** | **Leírás** |
| id | bigint(20) | A pilóta azonosítója |
| name | varchar(100) | A pilóta neve |
| picture | longtext | A pilótához tartozó kép URL címe |
| point | int(11) | A pilóta aktuális pontszáma |
| price | bigint(20) | A pilóta aktuális értéke |
| active | bit | A pilóta részt vesz-e az aktuális bajnokságban |
| team\_id | bigint(20) | A pilóta csapatának azonosítója |

**Championship tábla:**

A bajnokság legfontosabb adatai:

|  |  |  |
| --- | --- | --- |
| **Mező** | **Típus** | **Leírás** |
| id | bigint(20) | A bajnokság azonosítója |
| year | int(11) | A bajnokság éve |

**League tábla:**

Egy ligához tartozó legfontosabb adatok:

|  |  |  |
| --- | --- | --- |
| **Mező** | **Típus** | **Leírás** |
| id | bigint(20) | A liga azonosítója |
| name | varchar(100) | A liga neve |
| description | longtext | A liga részletes leírása |
| creator\_user\_id | bigint(20) | A ligát létrehozó felhasználó azonosítója |
| number\_of\_users | smallint(6) | A ligában tartózkodó felhasználók száma |
| date | date | A liga létrehozásának dátuma |

**User\_in\_league tábla:**

Ebben a táblában található azok az információk, hogy egy felhasználónak milyen kapcsolata van egy olyan ligában amiben szerepel.

|  |  |  |
| --- | --- | --- |
| **Mező** | **Típus** | **Leírás** |
| id | bigint(20) | A kapcsolat azonosítója |
| league\_id | bigint(20) | A liga azonosítója |
| user\_id | bigint(20) | A felhasználó azonosítója |
| role | bit | A ligában lévő felhasználó hatásköre |
| join\_date | date | A felhasználó ligába való belépésének dátuma |

**Track tábla:**

Ebben a táblában található egy pályához tartozó információk

|  |  |  |
| --- | --- | --- |
| **Mező** | **Típus** | **Leírás** |
| id | bigint(20) | A pálya azonosítója |
| name | varchar(100) | A pálya neve |
| country | varchar(100) | A pálya melyik országban található |
| city | varchar(100) | A pálya melyik városban található |
| picture | longtext | A pályához tartozó kép URL címe |

**Race tábla:**

Egy versenyhez tartozó adatok:

|  |  |  |
| --- | --- | --- |
| **Mező** | **Típus** | **Leírás** |
| id | bigint(20) | A futam azonosítója |
| date | date | A futam dátuma |
| track\_id | bigint(20) | A pálya azonosítója ahol a futam megrendezésre kerül |
| championship\_id | bigint(20) | A bajnokság azonosítója amelyikben a futam megrendezésre kerül |
| is\_result\_set | bit | A futamhoz lett-e állítva már eredmény |

**Result\_point tábla:**

Ez az a tábla, amiben megtalálhatóak azok az értékek, hogy egy futam után a csapatok és pilóták és ezáltal a felhasználók mennyi pontot kapnak.

|  |  |  |
| --- | --- | --- |
| **Mező** | **Típus** | **Leírás** |
| id | bigint(20) | A pontozás azonosítója |
| result | int(11) | A helyezés a futamon |
| driver\_race\_point | int(11) | A helyezéshez tartozó pontszám a futamon, amit a pilóta kap |
| driver\_qualification\_point | int(11) | A helyezéshez tartozó pontszám az időmérő edzésen, amit a pilóta kap |
| team\_race\_point | int(11) | A helyezéshez tartozó pontszám a futamon, amit a csapat kap |
| team\_qualification\_point | int(11) | A helyezéshez tartozó pontszám az időmérő edzésen, amit a csapat kap |
| rate | int(11) | A helyezéshez tartozó érték ami szerint a csapat/pilóta aktuális értéke csökken/nő |

**Result\_qualification tábla:**

Az időmérő edzéséhez tartozó eredményeket leíró tábla

|  |  |  |
| --- | --- | --- |
| **Mező** | **Típus** | **Leírás** |
| id | bigint(20) | Az eredmény azonosítója |
| result | int(11) | Az eredmény az időmérő edzésen |
| race\_id | bigint(20) | A futam azonosítója |
| driver\_id | bigint(20) | A pilóta azonosítója |
| team\_id | bigint(20) | A csapat azonosítója |

**Result\_race tábla:**

A futamhoz tartozó eredményeket leíró tábla

|  |  |  |
| --- | --- | --- |
| **Mező** | **Típus** | **Leírás** |
| id | bigint(20) | Az eredmény azonosítója |
| result | int(11) | Az eredmény a versenyen |
| race\_id | bigint(20) | A futam azonosítója |
| driver\_id | bigint(20) | A pilóta azonosítója |
| team\_id | bigint(20) | A csapat azonosítója |

**User\_result\_history tábla:**

Ebben a táblában találhatóak meg azok az információk, hogy egy futamon egy felhasználó milyen felállással és milyen eredményekkel vett részt.

|  |  |  |
| --- | --- | --- |
| **Mező** | **Típus** | **Leírás** |
| id | bigint(20) | Az eredmény azonosítója |
| user\_id | bigint(20) | A felhasználó azonosítója |
| driver1\_id | bigint(20) | A felhasználó elsőszámú pilótájának azonosítója |
| driver2\_id | bigint(20) | A felhasználó másodszámú pilótájának azonosítója |
| team1\_id | bigint(20) | A felhasználó elsőszámú csapatának azonosítója |
| team2\_id | bigint(20) | A felhasználó másodszámú csapatának azonosítója |
| team3\_id | bigint(20) | A felhasználó harmadszámú csapatának azonosítója |
| point | bigint(20) | A futamon szerzett pontok száma |
| money | bigint(20) | A futamon szerzett pénz |
| position | bigint(20) | A futam utáni pozíció a bajnokságban |

A kódban minden táblának van egy megfelelő entitás.

Az entitás egy olyan Java osztály, ami az adatbázis egy tábláját reprezentálja.

Az osztály elé kell tenni az @Entity annotációt és a rendszer ebből fogja tudni, hogy az osztály egy táblát reprezentál az adatbázisból.

Rengeteg annotáció áll rendelkezésünkre, hogy az osztályban beállítsuk, hogy melyik adattag pontosan milyen paraméterekkel rendelkezzen az adatbázisban. Ezek közül néhány:

@Table: Az osztály elé kell tenni és paraméterben megadható, hogy az adatbázisban a tábla neve pontosan mi legyen.

@Column: Az annotáció paramétereiben megadhatjuk, hogy a táblázatban az adott oszlopra milyen tulajdonságok legyen érvényesek.

A tényleges adatbázis műveleteket a Hibernate Criteria API segítségével implementáltam, ami lehetővé tette számomra, hogy az SQL utasítások helyett objektumorientáltan függvények segítségével vezéreljem az adatbázist.

Az alkalmazásban használtam a DAO (data access object) design pattern-t, ami arra a célra szolgál, hogy az üzleti logika és az adatbázison végrehajtódó művelet külön rétegbe kerüljenek. Az üzleti logikát leíró függvények a Service rétegben találhatóak meg és ezekből kerül meghívásra a DAO réteg, ami az adatbázis rekordjain végrehajtja a tényleges módosításokat. Ennek többek között nagy előnye, hogy ha a későbbiekben Hibernate helyett másik ORM könyvtárat szeretnék használni, akkor elég csak a DAO rétegben szereplő metódusokat átírnom.

**Konfigurációs fájlok**

Az alkalmazás elkészítése során használtam a Maven-t, ami szoftverprojektek menedzselésére nyújt segítséget.

A pom.xml fájlban találhatóak a buildelési folyamatokat leíró utasítások és a projekt függőségei.

A naplózáshoz található beállítások a log4j.xml-ben találhatóak. A konfigurációs fájlban megadtam, hogy a naplófájlok létrehozása hol történjen meg és milyen gyakran jöjjenek létre új naplófájlok. Minden nap új naplófájl indul, ezzel jobban átláthatóvá teszi a hibák keresését.

A Java-ban készített webalkalmazásoknak szükségük van egy web.xml fájlra is, ami egyfajta telepítési útmutatóként szolgál. Itt beállítottam a kódolást és megadtam a Spring és Spring Security-hez tartozó xml fájlok útvonalát is.

A Springhez tartozó konfigurációk a servlet-context.xml fájlban találhatóak.

Ebben a fájlban található többek között az adatbázishoz való csatlakozást leíró konfiguráció is.

A Spring Security-hez tartozó spring-security.xml fájlban lett beállítva, hogy az adminisztrációs felület csak bejelentkezés után érhető el és itt adtam meg azt a felhasználónév jelszó párost, amivel ellehet érni ezt a felületet is.

**További properties fájlok**

Az application.properties fájlban olyan adatokat adtam meg, amit nem szerettem volna a Java osztályokba égetni, se adatbázisba menteni.

A database.properties fájlban írtam le, hogy az adatbázis, amihez az alkalmazás csatlakozik az hol található és a csatlakozáshoz milyen információk szükségesek. Ezeket az adatokat a servlet-context.xml használja fel.

Továbbá webalkalmazásoknál fontos, hogy a felület több nyelven is elérhető legyen. Ezt úgy tudjuk elérni, ha az alkalmazáshoz tartozó nézetekbe nem direkt írjuk be a szövegeket, hanem kiszervezzük külön fájlokba vagy adatbázisba minden egyes nyelvhez tartozó értékeket. A Spring-hez tartozó xml fájlban beállítottam, hogy jelenleg a különböző nyelvekhez tartozó értékeket melyik properties fájlokban keresse.

A messages\_hu.properties fájl tartalmazza a nézetekben található tokenekhez a magyar nyelvű értékeket.

Jelenleg az oldal csak magyarul használható, de új nyelv felvétele a properties fájlok miatt nagyon egyszerűen és gyorsan tehető meg.

**Spring MVC**

A keretrendszer a Spring MVC modul integrálásával lehetővé teszi az MVC (Model-View-Controller) design pattern szerinti fejlesztést. Ennek a lényege, hogy az alkalmazás három fő részre tagolódik:

1: Model – A modellekben reprezentáljuk az adatbázis táblákat és itt implementáljuk az üzleti logikát.

2: View – Nézetekben szerepelnek az alkalmazáshoz tartozó oldalak kinézetei.

3: Controller: A bejövő adatokat a modellek segítségével feldolgozza, majd az új adatokat a nézet segítségével megjelenít.

A vezérlők felépítése a következőképen néz ki. Az osztály elé kell kitenni a @Controller annotációt, ami megmondja a keretrendszernek, hogy ez egy vezérlő lesz és a beérkezett kéréseket kiszolgáló metódusokat ezekben az annotációval ellátott osztályokban keresse.

Az osztályban szereplő metódusok elé a @RequestMapping annotációt kell tenni.

PL:

@RequestMapping(value=”/getDriver&id={driverId})

A kapcsos zárójel közötti érték változó, reguláris kifejezésekkel szűkíteni lehet, hogy milyen értékeket fogadjon el a rendszer.

Ha Ajax hívásokkal szeretnénk az adatokat feldolgozni, akkor két opció áll a rendelkezésünkre a Spring 4.0-ás verziójától. Az első, hogy az osztály elé a @Controller helyett a @RestController annotációt tesszük, ami azt írja le, hogy az osztályon belül szereplő összes metódus JSON alapú kommunikációt fog végrehajtani.

Ha az osztályon belül csak néhány helyen szeretnénk JSON-t használni, akkor a metódusok elé a @ResponseBody annotációt kell tenni, az osztály elé pedig a @Controller-t.

Ha a bejövő adatokat szeretnénk feldolgozni, akkor azt a következőkép is megtehetjük.

@RequestMapping(value="/saveNewUser")

public String saveNewUser(@Valid @ModelAttribute User user)

Az előző kódban a RequestMapping-ről már tudjuk, hogy milyen célra szolgál.

A @ModelAttribute annotáció azért felel, hogy a bejövő adatokat átalakítsa és innentől a „user” példányon keresztül érhessük el az adatokat, ha a bejövő adatok megfelelnek a User osztályban szereplő adattagoknak.

A @Valid annotáció segítségével azonnal validálni tudjuk a beérkező adatokat, ezt később megnézzük részletesebben.

A függvény visszatérési értéke String, ami valójában már az előállított oldal lesz. Ilyenkor a függvényen belül a return után egy létező oldal nevét kell megadnunk.

A Spring az előzőeken kívül még rengeteg annotációval rendelkezik, ami a fejlesztést hatékonyabbá és gyorsabbá teszi, és ha még nem létező annotációra lenne szükségünk, akkor saját magunk is implementálhatunk.

**Vezérlők az alkalmazásban**

A vezérlőket két külön csomagba helyeztem el. Az egyik csomagban a felhasználó felülethez tartozó vezérlők szerepelnek, míg a másikban az adminisztrátori felületért felelősek. A vezérlők visszatérhetnek nézetekkel vagy JSON adatstruktúrával.

A nézeteket szintén két külön csomagban találhatóak. Az egyik csoport a felhasználói, a másik az admin felülethez tartozó nézeteket tartalmazza.

**A felhasználói felület vezérlői és a hozzátartozó nézetek**

**StaticInformationsController**

A StaticInformationsController osztályban a statikus oldalakért felelős metódusok szerepelnek.

1. viewOtherUser: Ennek a segítségével tekinthetjük meg az egyes felhasználókhoz tartozó oldalakat az adatokkal együtt. A függvény paraméterben vár egy azonosítót, ami egy már létező felhasználóhoz tartozik és visszatér a „myTeam” nézettel.
2. listDrivers: A bajnokságban az aktív pilótákat listázza és visszatér a „drivers” nézettel.
3. listTeams: A bajnokságban az aktív csapatokat listázza és visszatér a „teams” nézettel.
4. results: A tabella megjelenítéséért felelős függvény. Az alkalmazáshoz sok felhasználó is tartozhat, ezért nem célszerű rögtön az összeset megmutatni az oldalon, ezért a függvény vár egy paramétert hogy az oldalon szereplő táblázat hányadik oldalát mutassuk meg. Ez a vezérlő a „users” nézettel tér vissza.
5. searchByUserName: Ennek segítségével kereshetünk rá felhasználókra. Ha a bejött karaktersorozat megtalálható valamelyik felhasználó nevében, akkor ezeket a felhasználókat megmutatjuk az oldalon a „users” nézet segítségével.
6. rules: Ez az akció egy teljesen statikus oldallal tér vissza, amin a játékhoz tartozó szabályok találhatóak.

**UserController**

A UserController osztályban találhatóak meg azok a függvények, amik a felhasználóval kapcsolatos módosításokért felelősek:

1. registration: Visszatér a „registration” nézettel ahol a felhasználó regisztrálni tud az oldalon.
2. saveUser: Ha a felhasználó kitöltötte a regisztrációs űrlapot, akkor az adatok ide érkeznek be. Ha minden adat megfelel a validálási feltételeknek, akkor a felhasználót felvesszük az adatbázisba, majd tájékoztatjuk a sikeres regisztrációról.
3. checkExistUserName: A regisztráció során, azért hogy a felhasználó minél gyorsabban tájékoztatást kapjon arról, hogy a rendszerben már szerepel-e ilyen nevű felhasználó Ajax hívást alkalmazunk, amikor leüt egy újabb karaktert. A függvény egy igaz vagy hamis értékkel tér vissza.
4. checkExistEmail: Az előzőhöz hasonlóan viselkedik annyi különbséggel, hogy a szervertől csak akkor kérdezi meg, hogy létezik-e már az e-mail cím, ha a megadott e-mail cím érvényes.
5. login: Ez a metódus dönti el, hogy a megadott felhasználónév és jelszó páros megtalálható-e a rendszerben. Ha megtalálható, de az e-mail cím még nem került megerősítésre, akkor az „activateAccount” nézettel tér vissza és megkéri a felhasználót, hogy aktiválja az e-mail címét. Ha megtalálható és már aktivált, akkor a „home” nézettel tér vissza és elérhetővé tesz a felhasználónak további funkciókat is az oldalon.
6. logout: Ez az akció jelentkezteti ki a felhasználót az oldalról és a kezdőoldalra továbbítja.

**BuyController**

Ebben a vezérlőben találhatóak a felhasználó vásárlási és eladásáért felelős akciók.

1. myTeam: A metódus lekéri a már bejelentkezett felhasználóhoz tartozó adatokat és visszatér a „myTeam” nézettel.
2. listDrivers: A vásárlás gombra kattintva ez a metódus fut le és listázza az oldalon az elérhető pilótákat, amik közül a felhasználó válogathat.
3. listTeams: Az előzőhez hasonlóan, csak a csapatokkal.
4. buyDriver: Ez a függvény felel a pilóta vásárlásért. Ha a felhasználónak minden feltétele adott ahhoz, hogy megvehesse a kiválasztott pilótát, akkor a vásárlás sikeres volt, ellenkező esetben tájékoztatja a felhasználót.
5. buyTeam: Az előzőhez hasonlóan, csak csapatra vonatkozóan.
6. sellDriver: Ez a függvény adja el a felhasználónak a kiválasztott csapatát.
7. sellTeam: A felhasználó által kiválasztott csapatot adja el a függvény.

**LeagueController**

Ebben a controllerben találhatóak a ligával kapcsolatos eljárások.

1. allLeagues: Lekéri a rendszerben található összes ligát és visszatér a „leagues” nézettel.
2. myLeagues: Lekéri a rendszerben található összes ligát, amiben a bejelentkezett felhasználó szerepel.
3. createLeague: Visszatér a „createLeague” nézettel, mely tartalmazza a liga felvételéhez szükséges űrlapot, hogy a felhasználó új ligát hozhasson létre.
4. saveLeague: A felhasználó által megadott adatokat validálja és ha minden adat megfelelő, akkor a rendszer létrehozza ligát.
5. searchByUserNameToInvite: Ha a paraméterben megadott karaktersorozat megtalálható a rendszerben lévő felhasználók nevében, akkor ezen felhasználók listázásra kerülnek a „users” nézet segítségével.
6. inviteUserToLeagueWithEmail: Ez a függvény egy meghívó levelet küld a kiválasztott felhasználónak, hogy csatlakozzon a ligába.
7. joinToLeague: A bejelentkezett felhasználó ennek a függvénynek a segítségével tud bejelentkezni egy ligába.

**Az adminisztrátori felülethez tartozó vezérlők és nézetek**

**DriverController**

Ebben az osztályban találhatóak azok az akciók, amik a pilóták adatait módosítják.

1. drivers: Az összes pilótát kéri le az adatbázisból és visszatér a „drivers” nézettel.
2. newDriver: Visszatér egy űrlappal, ahol az új pilótára vonatkozó adatokat adhatjuk meg.
3. saveDriver: Az űrlapon megadott adatokat validálja és ha minden megfelelő, akkor felveszi vagy módosítja az űrlapon található pilóta adatait az adatbázisban.
4. modifyDriver: A paraméterben kapott azonosító alapján kikeresi a rendszerből a pilótát és visszatér egy űrlappal, ahol a pilótához tartozó adatokat tudjuk módosítani.

**TeamController**

A csapatok módosításáért felelő metódusokat tartalmazza.

1. teams: Az összes csapatot kéri le az adatbázisból és visszatér a „teams” nézettel.
2. newTeam: Visszatér egy űrlappal, ahol az új csapatra vonatkozó adatokat adhatjuk meg.
3. saveTeam: Az űrlapon megadott adatokat validálja és ha minden megfelelő, akkor felveszi vagy módosítja az űrlapon található csapat adatait az adatbázisban.
4. modifyTeam: A paraméterben kapott azonosító alapján kikeresi az adatbázisból a csapatot és visszatér egy űrlappal, ahol a csapathoz tartozó adatokat tudjuk módosítani.

**TrackController**

A pályák módosításáért felelő metódusokat tartalmazza.

1. tracks: Az összes pályát kéri le az adatbázisból és visszatér a „tracks” nézettel.
2. newTrack: Visszatér egy űrlappal, ahol az új pályára vonatkozó adatokat adhatjuk meg.
3. saveTrack: Az űrlapon megadott adatokat validálja és ha minden megfelelő, akkor felveszi vagy módosítja az űrlapon található pálya adatait az adatbázisban.
4. modifyTrack: A paraméterben kapott azonosító alapján kikeresi az adatbázisból a pályát és visszatér egy űrlappal, ahol a pályához tartozó adatokat tudjuk módosítani.

**ChampionshipController**

A bajnokságok módosításáért felelő metódusokat tartalmazza.

1. championships: Az összes bajnokságot kéri le az adatbázisból és visszatér a „championships” nézettel.
2. newChampionship: Visszatér egy űrlappal, ahol az új bajnokságra vonatkozó adatokat adhatjuk meg.
3. saveChampionship: Az űrlapon megadott adatokat validálja és ha minden megfelelő, akkor felveszi vagy módosítja az űrlapon található bajnokság adatait az adatbázisban.
4. modifyChampionship: A paraméterben kapott azonosító alapján kikeresi az adatbázisból a bajnokságot és visszatér egy űrlappal, ahol a pályához tartozó adatokat tudjuk módosítani.

**RaceController**

A futamok módosításáért felelő metódusokat tartalmazza.

1. races: Az összes futamot kéri le az adatbázisból és visszatér a „races” nézettel.
2. newRace: Visszatér egy űrlappal, ahol az új futamra vonatkozó adatokat adhatjuk meg.
3. saveRace: Az űrlapon megadott adatokat validálja és ha minden megfelelő, akkor felveszi vagy módosítja az űrlapon található futam adatait az adatbázisban.
4. modifyRace: A paraméterben kapott azonosító alapján kikeresi az adatbázisból a futamot és visszatér egy űrlappal, ahol a pályához tartozó adatokat tudjuk módosítani.

**ResultPointController**

A vezérlőben található akciók segítségével tudjuk beállítani, hogy a futamok után milyen mértékű legyen a díjazás.

1. resultPoints: Visszatér egy táblázattal, ami valójában egy űrlap. Itt lehet módosítani a különböző helyezésekhez tartozó adatokat módosítani vagy újat felvenni.
2. saveResultPoints: Az előző táblázatban megadott adatokat validálja és ha minden megfelel, akkor menti a rendszerbe őket.
3. deleteResultPoint: Ez egy Ajax hívás, ami segítségével az előző táblázatból tudunk egy sort kitörölni.

**RaceResultController**

Ennek a vezérlőnek a segítségével lehet felvenni vagy megtekinteni egy futamhoz tartozó eredményt.

1. newRaceResult: Visszatér egy táblázattal, ami tartalmazza az összes bajnokságban szereplő pilótát. Minden helyezéshez csak egy pilóta rendelhető. Csapatokat nem látunk ebben a táblázatban, mert a rendszer tudja, hogy melyik pilóta melyik csapatban szerepel.
2. saveRaceResult: Az előző táblázatban beállított eredményt menti el az adatbázisba, ha nem szerepel egy pilóta kétszer ugyanabban az oszlopban. Frissíti a pilóták és csapatok értékeit, illetve kiosztja a felhasználóknak a díjazásokat és elmenti, hogy a futamon milyen felállással indultak.
3. resultRace: A paraméterben kapott azonosító alapján keresi ki a futamhoz tartozó végeredményt és visszatér a „raceResult” nézettel.

**ExcelController**

A controllerben található metódusok vizsgálják be vagy állítják elő a megfelelő excel fájlokat.

1. excelFiles: Visszatér az „excel” nézettel, ahol letölthetőek vagy feltölthetőek lesznek a fájlok.
2. downloadExcelTemplateTeams: Ez a metódus állítja elő azt az excel fájlt és teszi letölthetővé a felhasználó számára, ahol csak az oszlop nevek vannak kitöltve.
3. downloadExcelTemplateWithTeams: Ez az akció állítja elő azt az excel fájlt, amiben szerepel minden csapat az adatbázisból.
4. uploadExcelTeam: Ez a metódus vizsgálja be a feltöltött excel fájlt. Ha a fájlban minden megfelel a feltételeknek, akkor frissíti az adatbázisban a csapatok táblát. Ha a fájlban legalább egy adat nem felel meg, akkor erről tájékoztatja a felhasználót és az adatbázison nem hajt végre semmilyen módosítást.

**Beérkező adatok vizsgálata**

Az alkalmazásba beérkező adatoknak a validálása a szerver oldalon kötelező, a kliensoldalon pedig erősen ajánlott.

Szerveroldalon az adatokat azért kell ellenőrizni, hogy ne hajthassunk végre olyan műveleteket, ami hibákat okozhatna vagy érzékenyen érinthetné az üzleti logikát.

A kliensoldali validáció növeli a felhasználói élményt és csökkenti a szerver terhelését.

A beérkező adatok gyors és hatékony validációjának érdekében a Hibernate Validatort használtam szerveroldalon. Ennek a könyvtárnak a segítségével az osztályon belül lévő adattagokra tudunk annotációkat tenni és megadni különböző feltételeket.

Ilyen annotációk lehetnek többek között a következőek:

@NotNull: Az adott adattag értéke nem lehet nulla.

@Size(min=5,max=10): Az adattag hosszára ad megszorítást.

@URL: az adattagnak érvényes URL-nek kell lennie.

@Future: A dátumnak a jövőben kell lennie.

Az előző annotációkon kívül még van több is és sajátokat is lehet létrehozni.

A korábban említett Controller-nél szereplő @Valid annotáció azt írja le, hogy a beérkezett adatokra történjen-e validálás vagy sem. Ha az annotációt használjuk és valamelyik feltétel nem teljesül, akkor a hibát azonnal tudjuk kezelni. Ha nem használjuk, akkor a feltételeknek a kiértékelődése nem történik meg.