SZAKDOLGOZAT

MTLSZ Játékvezetői Bizottság nyilvántartás

Készítette: Csanádi Péter

Programtervező informatikus szak (BSc)

Témavezető: Bende Imre tanársegéd

Eötvös Loránd Tudományegyetem, 2019

<témabejelentő>

**Bevezetés**

Szakdolgozatom témája a Magyar Tollalabda Szövetség Játékvezetői és Versenybírói Bizottságának online nyilvántartása. A weboldal célja, hogy a bizottság tájékoztassa a versenybírókat a versenynaptárról, online felületet biztosítson a versenyekre való jelentkezéshez, ezen jelentkezéseket nyilvántarsa és a kiválasztott versenybírókat erről a tényről tájékoztassa.

Az 1969. óta önállóan létező Magyar Tollaslabda Szövetség Játékvezetői és Versenybírói bizottsága feladatai között szerepel a játékvezetők és versenybírók versenyekre való kijelölése és értesítése, valamint minősítése.

Az elmúlt években ennek megvalósítására a bizottság elnöke egy Google Drive-ban tárolt táblázatot használt, melynek soraiban szerepeltek a versenynaptár azon elemei, melyekre a szervező döntnököt és/vagy játékvezetőt igényelt. A versenybírók – ha jelentkezni kívántak egy versenyre – beírhatták a nevüket a táblázat megfelelő oszlopába. (Szerencsés esetben az addigi bejegyzéseket nem kitörölve.) Átmeneti megoldásként a még jelenleg is létező és használatos táblázatot csak a bizottság elnöke szerkesztheti, a többieknek megtekintési joguk van. Jelentkezni az elnöknek küldött email útján lehetséges, aki a feldolgozás során frissíti a táblázatot. Ezt a gyakorlatot kívánja leváltani az az alkalmazás, melynek jelen szakdolgozat kívánja letenni az alapjait.

Szakdolgozat a Laravel keretrendszerrel készül, amelyet azért választottam, mert egyrészt az authentikációra szinte kész megoldást szállít, másrészt olyan eszközöket és technológiákat használ és szolgáltat, amik helyes használat esetén leveszik a fejlesztő válláról az alacsony szintű programozás terhét, hogy teljes mértékben a megoldandó feladatra koncentrálhasson és logikailag rendszerezni tudja azt.

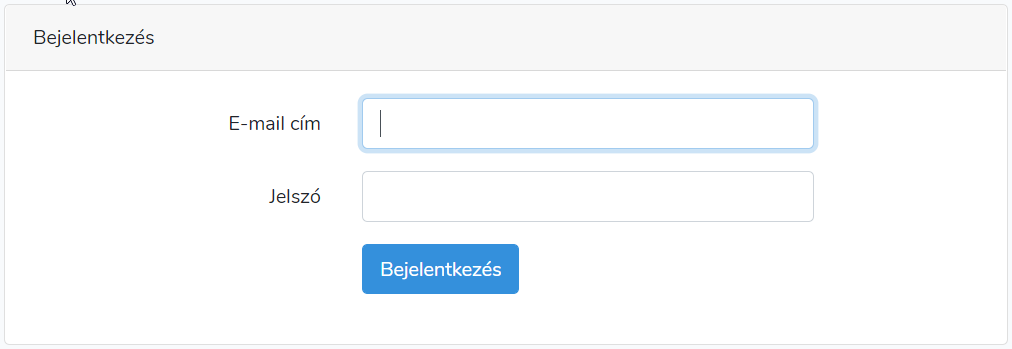
**Felhasználói dokumentáció**

**A feladat**

A weboldal célja, hogy a versenybírók (döntnökök és játékvezetők) jelentkezéseinek nyilvántartását kezelje, online elérést biztosítson minden felhasználónak, aki authentikáció után saját maga tekintheti meg a versenynaptárat és azt, hogy mely versenyekre és milyen szerepre jelentkezett.

**Kezdőoldal**

Mivel a versenynaptár megtekinthető a <http://badminton.hu/versenyek> címen, nincs olyan része a weboldalnak, melyet az authentikáció előtt érdemes megjeleníteni. Ezen megfontolásból a kezdőoldalon csak bejelentkezésre van lehetőség.



**Menü**

Bejelentkezés után minden oldal tetején a menü jelenik meg, ami az oldalon való navigációt segíti.

**Normál felhasználó**

Normál felhasználóként az alábbi kettő menüpont áll rendelkezésre:



**Versenynaptár**

A sikeres authentikációt követően a versenynaptár jelenik meg. A képernyőn az aktuális versenyszezon versenyei látszanak (dátum, elnevezés, helyszín, igényelt játékvezetők száma, eddig jelentkezett játékvezetők száma). Ha a felhasználó jogosult döntnökként közreműködni versenyeken, akkor egy további oszlop jelenik meg ’Döntnök’ fejléccel, mely oszlop segítségével a versenyekre jelentkezni tud. (Amennyiben jelentkezett, ezt visszavonni tudja.) Hasonlóan ha a felhasználó rendelkezik játékvezetői minősítéssel, a jelentkezéseit a ’Játékvezető’ fejlécű oszlopban kezelheti. A jelentkezés (és leiratkozás) csak jövőbeli versenyekkel kapcsolatban lehetséges.



A jelenlegitől eltérő versenyszezon versenyeinek megjelenítésére a táblázat felett látható navigációs gombokkal van lehetőség.

**Jelentkezéseim**

Ezen az oldalon azok a jövőbeli versenyek jelennek meg, ahova a felhasználó akár döntnöki akár játékvezetői szerepre jelentkezett. Az elrendezés a versenynaptáréhoz hasonló, itt azonban jelentkezésre nem, csak a már megtett, de még nem jóváhagyott jelentkezések visszavonására van lehetőség.



**Jelentkezés versenyre**

A versenynaptárban lévő versenyekre a felhasználó akkor tud jelentkezni, ha rendelkezik az adott szerepkörben minősítéssel.

A ’Döntnök’ és ’Játékvezető’ oszlopokban lévő ikonok jelentése:

* jelentkezett (még nem jóváhagyott):

a felhasználó már jelentkezett a versenyre, de a bizottság még nem választotta ki a verseny közreműködőit (a jelentkezés még visszavonható a kisebb ikonra kattintva)

* jelentkezett (jóváhagyott):

a felhasználó jelentkezett a versenyre és a bizottság kiválasztotta közreműködőnek

* jelentkezett (elutasított):

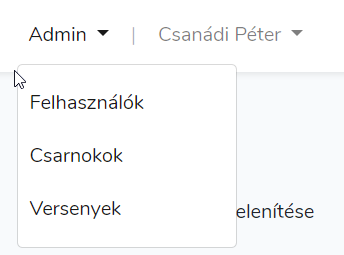
a felhasználó jelentkezett a versenyre, de a bizottság őt nem választotta ki közreműködőnek

* nem jelentkezett:

a felhasználó még nem jelentkezett a versenyre (jelentkezhet a kisebb ikonra kattintva)

**Adminisztrátor felhasználó**

Az adminisztrátor jogosultsággal rendelkező felhasználók további menüpontokat érhetnek el, melyek a fenti menü jobb oldalán az Admin menüpont lenyitásával érhetők el.



**Felhasználók**

Ezen az oldalon listázhatjuk az adatbázisban szereplő felhasználókat. Megjelenített adatok: név, email cím, játékvezetői (Jv) szint, döntnöki (D) szint, adminisztrátor-e a felhasználó. Minden sorban találunk ’Szerkesztés’ és ’Törlés’ (törölt felhasználó esetén ’Visszaállítás’) gombokat.

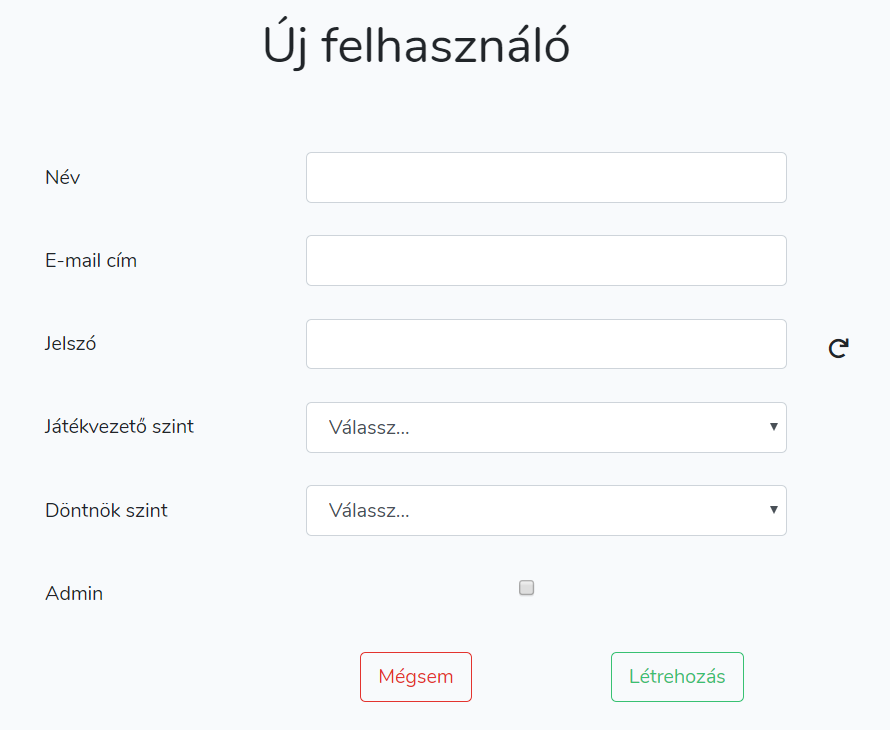


**Felhasználó törlése és törölt felhasználó visszaállítása**

A felhasználó sorában található törlés gomb segítségével törölhetjük. A törölt felhasználók megjelenítése a táblázat felett található jelölőnégyzet segítségével lehetséges. Ha egy felhasználó már törölt, a törlés gomb helyett a visszaállítás gomb jelenik meg, amivel a törlést visszavonhatjuk.

**Felhasználó létrehozása**

Új felhasználót az ’Új felhasználó’ gombra kattintva és a megfelelő adatokat megadva tudunk létrehozni. Kezdeti jelszót kötelező megadni (vagy a mező melletti gombra kattintva generálni).



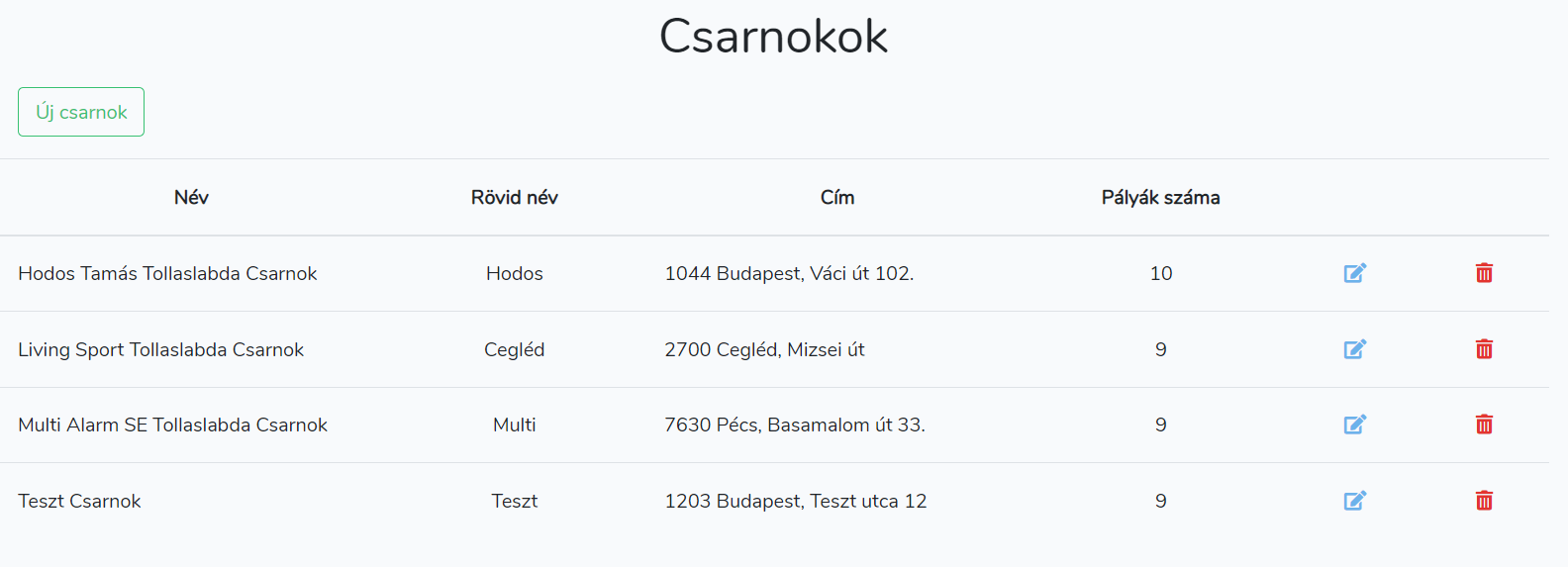
**Felhasználó szerkesztése**

Ezen az oldalon egy meglévő felhasználó adatait módosíthatjuk. A beviteli mezők a felhasználó jelenlegi adataival vannak eredetileg kitöltve. A ’jelszó’ mezőt üresen hagyva a jelenlegi jelszó érvényben marad, a kitöltés ugyanúgy lehetséges, mint új felhasználó létrehozása esetén. A módosítások elvégzése után az oldal alján található gombok közül választhatunk:

* **visszaállítás**: A beviteli mezők eredeti (a módosítások előtti) értékekre való visszaállítása. (A ’jelszó’ mező üres lesz)
* **mégse**: Visszatérés az összes felhasználót listázó oldalra a módosítások mentése nélkül.
* **mentés**: Az adatok mentése és visszatérés az összes felhasználót listázó oldalra. Az oldal tetején (a menü alatt) látható üzenet tájékoztat, hogy a mentés sikeres volt-e.

**Csarnokok**

Ezen az oldalon listázhatjuk az adatbázisban szereplő csarnokokat. Megjelenített adatok: név, rövid név, cím, pályák száma. Minden sorban találunk ’Szerkeszté’ és ’Törlés’ (törölt csarnok esetén ’Visszaállítás’ gombokat. Előbbi a szerkesztés oldalra irányít, az utóbbira való kattintás után ugyanezen az oldalon maradunk. Minden sorban találunk ’Szerkesztés’ és ’Törlés’ (törölt csarnok esetén ’Visszaállítás’) gombokat.

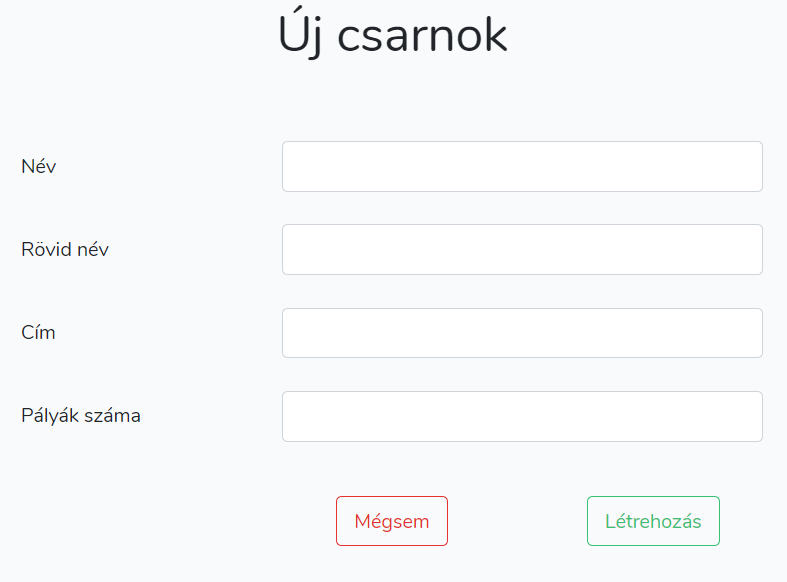


**Csarnok törlése és törölt csarnok visszaállítása**

A csarnok sorában található törlés gomb segítségével törölhetjük. A törölt csarnokok megjelenítése a táblázat felett található jelölőnégyzet segítségével lehetséges. Ha egy csarnok már törölt, a törlés gomb helyett a visszaállítás gomb jelenik meg, amivel a törlést visszavonhatjuk.

**Csarnok felvitele**

Új csarnokot ’Új csarnok’ gombra kattintva és a megfelelő adatokat megadva tudunk létrehozni.



**Csarnok szerkesztése**

Ezen az oldalon egy meglévő csarnok adatait módosíthatjuk. A beviteli mezők a csarnok jelenlegi adataival vannak kitöltve. A módosítások elvégzése után az oldal alján található gombok közül választhatunk.

**Versenyek**

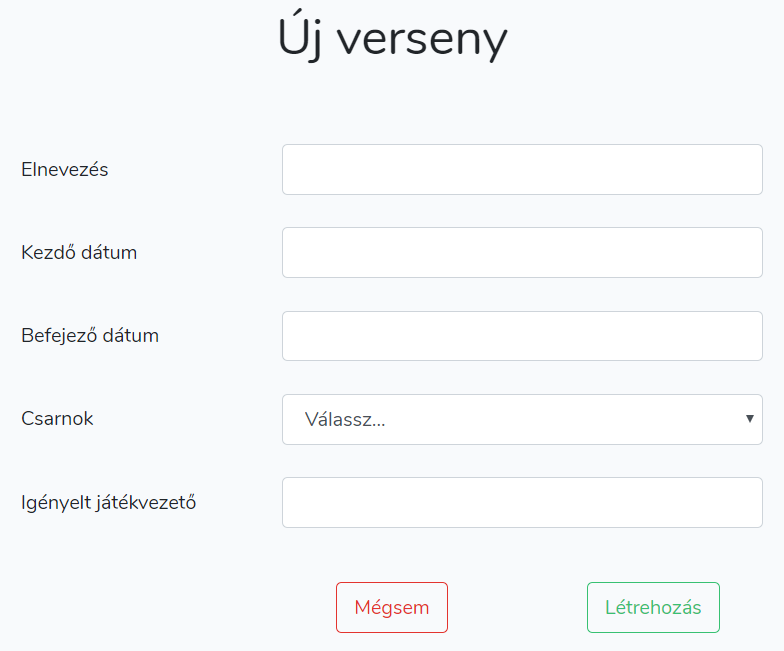
Ezen az oldalon listázhatjuk az adatbázisban található versenyeket. Megjelenített adatok: Név, kezdő dátum, befejező dátum, csarnok, igényelt játékvezetők száma. A ’csarnok’ oszlopban annak a csarnoknak a rövid neve szerepel, ahol a verseny megrendezésre fog kerülni. Ha az egeret egy csarnok rövid neve fölé visszük, megjelenik a csarnok teljes neve és címe.

**Verseny törlése és törölt verseny visszaállítása**

A verseny sorában található törlés gomb segítségével törölhetjük. A törölt versenyek megjelenítése a táblázat felett található jelölőnégyzet segítségével lehetséges. Ha egy verseny már törölt, a törlés gomb helyett a visszaállítás gomb jelenik meg, amivel a törlést visszavonhatjuk.

**Verseny felvitele**

Új csarnokot ’Új csarnok’ gombra kattintva és a megfelelő adatokat megadva tudunk létrehozni.

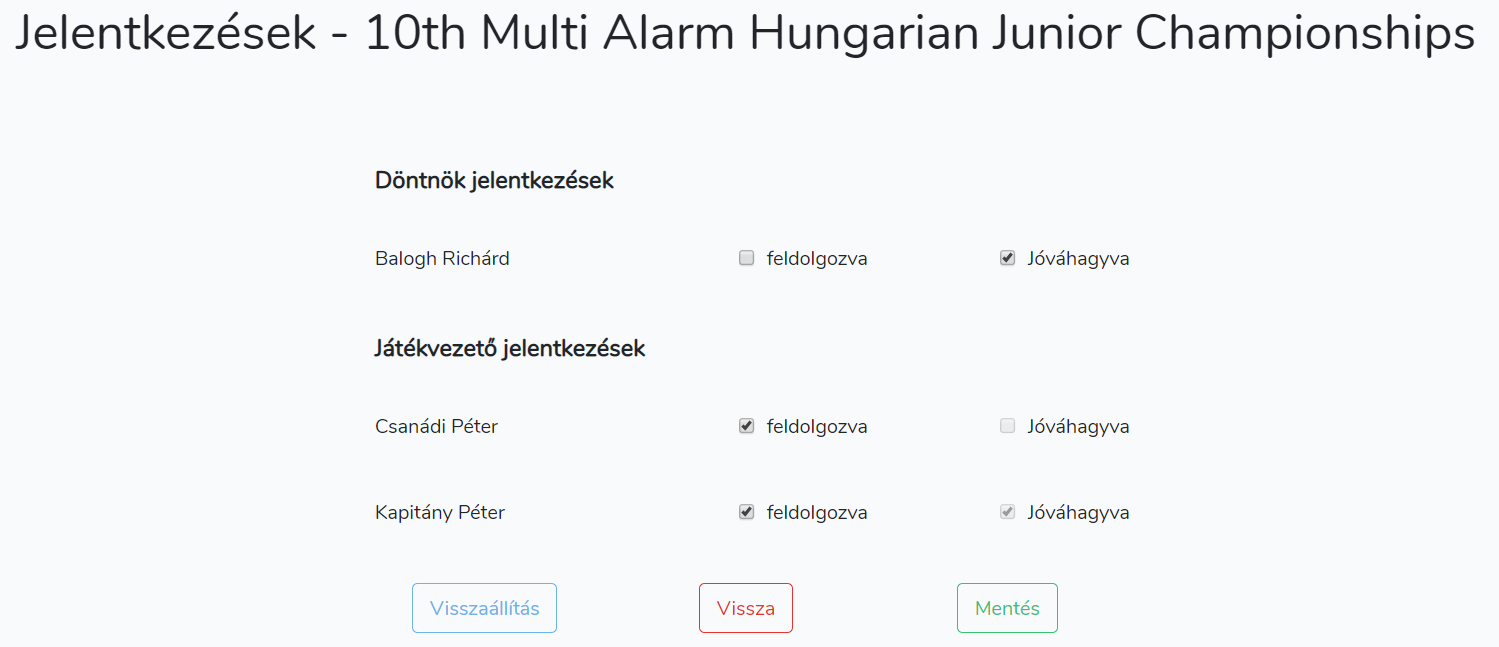


**Verseny szerkesztése**

Ezen az oldalon egy meglévő verseny adatait módosíthatjuk. A beviteli mezők a verseny jelenlegi adataival vannak eredetileg kitöltve. A módosítások elvégzése után az oldal alján található gombok közül választhatunk, melyek működése a felhasználók szerekesztése oldalon szereplő gombokéval megegyezik.

**Jelentkezések kezelése**

Ezen az oldalon felsorolásra kerül azok a felhasználók, akik döntnök vagy játékvezető szerepkörre jelentkeztek a versenyre. Eldönthetjük minden jelentkezésről, hogy jóváhagyjuk vagy sem, ezután pedig azt, hogy a jelentkezést feldolgozottnak jelezzük. Ha egy jelentést feldolgoztunk, nem tudjuk a jóváhagyás állapotát változtatni. (Az értesítő emailt kiküldésekor azok a felhasználók kapják meg, akiknek a jelentkezése feldolgozott és jóváhagyott.)



**Értesítő email kiküldése**

Azoknak a felhasználóknak, akik a versenyre jelentkeztek és a jelentkezésük feldolgozott és jóváhagyott státuszú, értesítő levelet küldhetünk. A levél általános tájékoztatást tartalmaz.

**Fejlesztői dokumentáció**

**A telepítés menete**

**platform**

Windows 10

**szoftverek**

Xampp (Cross platform Apache, MariaDB, PHP, Perl) <https://www.apachefriends.org/index.html>

PHP 7.3.x

Composer <https://getcomposer.org/download/> (needed php)

Laravel (installed by Composer)

node.js (npm – node.js package manager) <https://nodejs.org/en/>

**kiadott parancsok**

composer global require laravel/installer

npm install

npm run dev

c:\xampp\apache\conf\httpd.con 🡪

#

# DocumentRoot: The directory out of which you will serve your

# documents. By default, all requests are taken from this directory, but

# symbolic links and aliases may be used to point to other locations.

#

DocumentRoot "C:\Users\cspca\OneDrive - Eotvos Lorand Tudomanyegyetem\szakdolgozat\YJRB29\Umpires\public"

<Directory "C:\Users\cspca\OneDrive - Eotvos Lorand Tudomanyegyetem\szakdolgozat\YJRB29\Umpires\public">

create dbuser, create database

\.env szerkesztése

php artisan migrate:fresh --seed

**Keretrendszer**

A szakdolgozat a Laravel keretrendszer felhasználásával készült. Készítői hisznek benne, hogy a fejlesztésnek élvezhető és kreatív élménynek kell lennie és ez a keretrendszer leveszi a teher egy részét a fejlesztő válláról azzal, hogy megkönnyíti a legtöbb web projektben előforduló általános tevékenységeket.

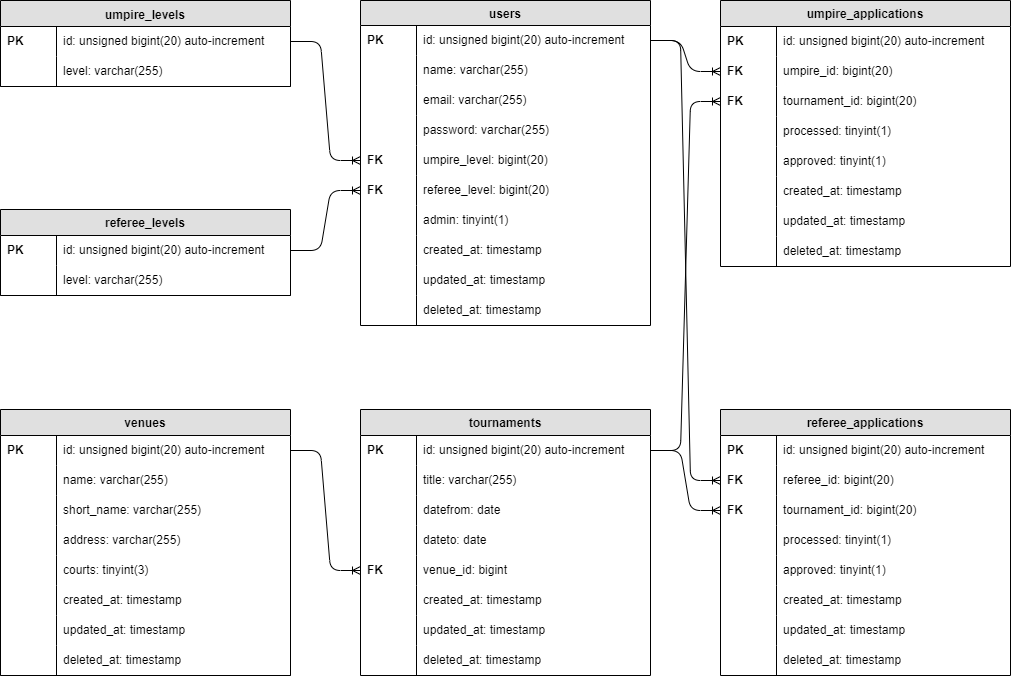
Minden beérkező kérés belépési pontja a public/index.php file, ami nem tartalmaz sok kódot, a feladata mindössze annyi, hogy elindítsa a keretrendszer többi részének betöltését. Szakdolgozatomban a console (api) requestek automatikusan elutasításra kerülnek, a HTTP requesteket pedig a HTTP Kernel dolgozza fel, mely az Illuminate\Foundation\Http\Kernel osztály leszármazottja. Tartalmazza bootstrapperek egy tömbjét, melyek olyan elemekért felelősek, mint pl. a hibakezelés és a naplózás konfigurációja, a futtatási környezet feltérképezése és egyéb feladatok, amiket még a kérés feldolgozása előtt meg kell tenni. Ezt a részt nem módosítottam, legfeljebb a konfigurációs beállításokat tartalmazó <config> mappa file-jain keresztül.

A HTTP kernel definiál ú.n. middleware-eket is, amiken minden kérésnek még a feldolgozás előtt át kell haladnia. Az alapértelmezetten rendelkezésre álló (és az authentikáció hozzáadása után létrejött) middleware-ek mellett egy middleware-t definiáltam (RedirectIfNotAdmin), melynek célja, hogy az adminisztrátor jogosultsággal nem rendelkező felhasználók ne érjenek el nekik nem engedélyezett részeket.

Egyéb felhasznált technológiák:

* Bootstrap: nyílt forráskódú eszközkészlet, HTML, CSS és JS használatával történő fejlesztéshez, reszponzív rácsvonal rendszer, könnyen kiegészíthető előkészített komponensekkel
* jQuery: gyors, kicsi, feature-gazdag JavaScript könyvtár

**Adatbázis**

****

Az alkalmazás felépítéséből adódóan a felhasználókat (potenciális döntnököket és játékvezetőket) tartalmazó *users* és a versenyeket tartalmazó *tournaments* tábláknak központi szerepe van.

Az *umpire\_levels* és *referee\_levels* táblák kivételével – melyek várhatóan sosem fognak változni – minden tábla rendelkezik a következő mezőkkel: ’*created\_at’, ’updated\_at’*, valamint ’*deleted\_at’*. Ezeket a mezőket a Laravel keretrendszer hozza létre és kezeli. Előbbi kettő a változások követésére, utóbbi a – később visszavonható – törlés kezelésére szolgál.

Minden tábla elsődleges kulcsa (PK) az ’*id’,* ezek az egy-több kapcsolatokban külső kulcsként (FK) is előfordulnak

**Migrációk**

A Laravel az adatbázis tábláit ú.n. migrációk segítségével hoza létre (és módosítja). A migrációk a *database/migrations* mappában találhatók. Minden migráció tartalmaz egy *up()* és egy *down()* metódust. Előbbi segítségével létrehozhatunk és módosíthatunk táblákat, utóbbi pedig ezen műveletek megfordítására való. A migrációk sorrendjét a fájlnevekben szereplő időbélyegek jelzik a keretrendszer számára. Szakdolgozatomban csak táblákat létrehozó migrációk vannak (és a táblák eldobására vonatkozó parancsok), az adatbázis felépítésére alkalmasak, egyfajta verziókövetésre nem.

**Seedek**

A *database/seeds* mappában található osztályok a létrehozott adatbázistáblák kezdeti és/vagy tesztadatokkal való feltöltésére szolgálnak. Minden táblához tartozik seed osztály elsősorban tesztelési céllal, bár ha a *users* táblában egy adminisztrátort jogosultsággal rendelkező felhasználó sem lenne, akkor nem tudnánk használni az alkalmazást.

A migrációk és seedek futtatása az alábbi paranccsal történik:

*php artisan migrate:fresh --seed*

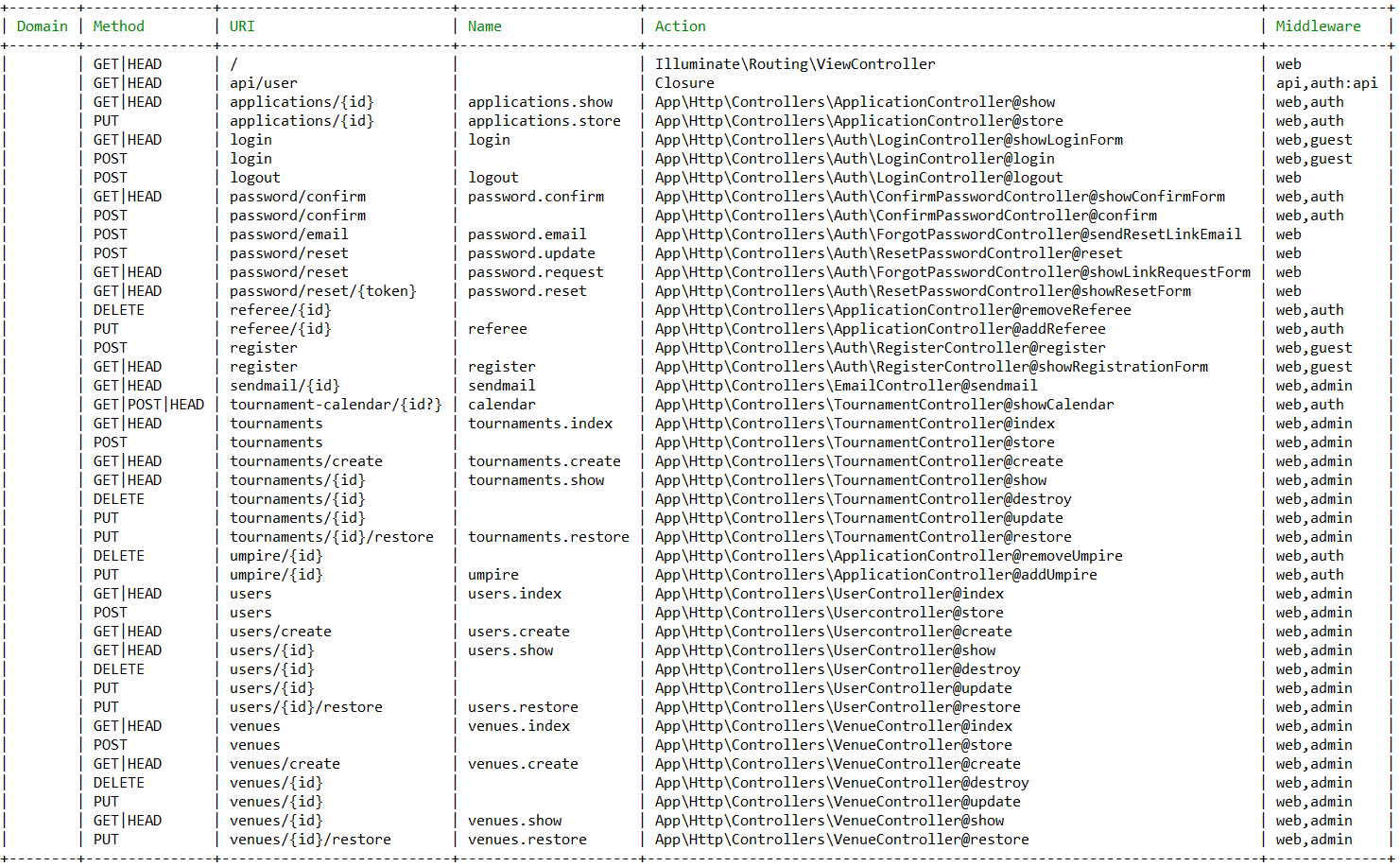
A parancs hatására az összes eddigi adatbázistábla eldobásra kerül (*down()* metódus), majd újra létrejönnek és a *--seed* kapcsoló hatására a kezdeti adatokkal feltöltődnek.

**Admin middleware**

A Laravel által alapértelmezett biztosított (valamint az authentikáció hozzáadásával bekerült) middleware-ek kiegészítéseként egy újabb middleware-t definiáltam (app/Http/Middleware/RedirectIfNotAdmin.php). Ugyan sorrendben az *Authenticate* után áll (ld. app/Http/Kernel.php:$middlewarepriority), az útvonalak definiálásakor mindkettő nem került beállításra, így ez a middleware is ellenőrzi, hogy létezik-e felhasználó (vagyis authentikált-e). Ezután foglalkozik azzal, hogy admin jogosultsággal rendelkezik-e. Ha ezen feltételek egyike nem teljesül, átirányít a ’versenynaptár’ útvonalra. Egyéb esetben a kérést továbbítja (a következő middleware-nek).

**HTTP útvonalak**

A programban a következő útvonalak vannak definiálva:



* Method: az elfogadott HTTP kérés típusok
* URI: az elfogadott elérési út
* Name: az útvonal neve
  + a {} közötti paraméterek kötelezőek (kivéve, ha ? szerepel mögöttük)
  + a keretrendszer *route()* segédfüggvénye a név és az esetleges paraméterek alapján visszaadja az útvonal elérési útját
* Action: a kezelés módja, amikor az útvonalon kérés érkezik be
  + egy kivétellel minden beérkező kérést egy conroller metódus dolgoz fel
* Middleware: azok a middleware-ek, amiken a kérés átmegy, mielőtt feldolgozásra kerülne

Az útvonalak definiálása és tulajdonságaik beállítása a HTTP kérések kapcsán a routes/web.php fájlban történik.

Példa:

*Route::put(’/umpire/{id}’, ’ApplicationController@addUmpire’)->name(’umpire’);*

Ha PUT HTTP kérés érkezik az /umpire/{id} útvonalon (az {id} kötelező paraméter), akkor az Application Controller addUmpire függvénye dolgozza fel. Az útvonal neve ’umpire’.

**Modellek**

**Eloquent**

A Laravel keretrendszerhez tartozó Eloquent ORM (object-relational mapping) az AcitveRecord design pattern egy megvalósítása az adatbázissal való munkához. Minden adatbázistáblához tartozik egy modell, amin keresztül a táblán műveleteket végezhetünk.

**Általános beállítások**

Az alábbi változók szinte az összes modellnél beállításra kerültek, az Eloquent ezek segítségével határozza meg a modell és az adatbázistábla közti kapcsolat jellegét és a kezelés módját.

* $table: a modellhez tartozó adatbázistábla neve
* $primaryKey: az adatbázistábla elsődleges kulcsa
* $incrementing: az elsődleges kulcs automatikusan inkrementálódik-e
* $timestamps: szükség van-e a ’created\_at’ és ’updated\_at’ mezőkre, melyek nyilvántartják, hogy egy rekord mikor jött létre és mikor változott meg utoljára
* $casts: azon mezők listája, amelyeket beolvasáskor valamilyen beépített PHP típusra kell kényszeríteni
* $with: azon egyéb modellek listája, melyeket beolvasáskor mindig be kell tölteni, mert a visszaadott változón kresztül azokat is el akarjuk érni
* $dates: a dátum típusú mezők listája

**Játékvezetői és döntnöki szint (*UmpireLevel, RefereeLevel*)**

A lehetséges játékvezetői és döntnöki szinteket reprezentáló modellek. A hozzájuk tartozó adatbázistáblák: *umpire\_levels* és *referee\_levels*.

Attribútumaik:

* azonosító (*id*)
* szint (*level*)

Az attribútumok és az adatbázistáblában nekik megfeleltethető mezők nevei megegyeznek.

Ezek a modelleki konstansnak tekinthetők, nem várható, hogy a lehetséges játékvezetői és döntnöki szintek a közeljöbőben változnak, ezért nem is tartozik a modellekhez controller. Szerepük mindössze az, hogy a felhasználók játékvezetői és döntnöki szintjei ne szövegként legyenek ’beleégetve’ az adatbázis rekordjaiba. Ha mégis szükséges változtatni rajtuk, az a web api-n keresztül nem, csak közbetlen adatbáziskapcsolattal lehetséges.

**Csarnok (*Venue*)**

A tollaslabda versenyek magyarországi lebonyolítására alkalmas csarnokokat reprezentáló modell. A hozzá tartozó adatbázistábla: *venues*.

Attribútumai:

* azonosító (*id*)
* név (*name*)
* rövid név (*short\_name*)
* cím (*address*)
* pályák száma (*courts*)
* versenyek (*tournaments* – kapcsolat által)

Az attribútumok és az adatbázistáblában nekik megfeleltethető mezők nevei megegyeznek.

**Felhasználó (*User*)**

A felhasználókat reprezentáló modell. A hozzá tartozó adatbázistábla: *users*.

Attribútumai:

* azonosító (*id*)
* név (*name*)
* email cím (*email*)
* jelszó (*password*)
* játékvezetői szint (*umpireLevel* – *umpire\_level*)
* döntnöki szint (*refereeLevel* – *referee\_level*)
* adminisztrátor-e (*admin*)
* játékvezetői jelentkezések (*umpireApplications* – kapcsolat által)
* döntnöki jelentkezések (*refereeApplications* – kapcsolat által)

Az attribútumok és az adatbázistáblában nekik megfeleltethető mezők nevei megegyeznek.

**Verseny (*Tourament*)**

A magyarországi tollaslabda versenyeket reprezentáló modell. A hozzá tartozó adatbázistábla: *tournaments*.

Attribútumai:

* azonosító (*id*)
* megnevezés (*title*)
* kezdő dátum (*datefrom*)
* befejező dátum (*dateto*)
* csarnok (*venue* – *venue\_id)*
* igényelt játékvezetők száma (*requested\_umpires*)
* játékvezetői jelentkezések (*umpireApplications* – kapcsolat által)
* döntnöki jelentkezések (*refereeApplications* – kapcsolat által)

**Játékvezetői és döntnöki jelentkezés (*UmpireApplication, RefereeApplication*)**

A felhasználók jelentkezéseit reprezentáló modellek. A hozzájuk tartozó adatbázistáblák: *umpire\_applications* és *referee\_applications*.

Attribútumaik:

* azonosító (*id*)
* játékvezető (*umpire* – kapcsolat által) és döntnök (*referee* – kapcsolat által)
* verseny (*tournament* – kapcsolat által)
* feldolgozva (*processed*)
* jóváhagyva (*approved*)

Amikor a program email értesítést küld egy versenyre kiválasztott döntnök(ök)nek és játékvezetőknek, azokat a felhasználókat választja ki, melyeknek a jelentkezése jóváhagyott ÉS feldolgozott státuszú.

**Soft delete**

A rekordok valós letörlése mellett az Eloquent része a ’soft delete’ is. Ebben az esetben a modell nem kerül törlésre az adatbázisból, ehelye.tt a ’deleted\_at’ attribútuma kerül beállításra és a táblában tárolásra. Ha a modellnek nem null a ’deleted\_at’ értéke, akkor a modell ’soft deleted’.

A ’soft delete’ használatához a modellben a ’use SoftDeletes’ direktívára van szükség. Szakdolgozatomban a játékvezetői és döntnöki szinteket reprezentáló modelleken kívül mindenhol ezzel a lehetőséggel éltem. Ez teszi lehetővé, hogy a modellek az esetleges törlés után visszaállíthatók legyenek.

Amikor az Eloquent segítségével ki akarjuk listázni egy modell összes rekordját, az *’all()*’ függvényt használjuk. Ezeknél a modelleknél viszont ez a függvény csak a nem törölt rekordokat adja vissza. Ha tényleg minden rekordra szükségünk van, a ’*withTrashed()*’ hívására is szükség van. (Az ’*onlyTrashed()*’ pedig csak a törölt elemeket listázza.)

**Kapcsolatok**

Az adatbázistáblák közötti kapcsolatok leírásához és használatához az Eloquent kényelmes eszközt kínál. Szakdolgozatomban a táblák között egy-több kapcsolatok vannak, melyek a következő módon definiálhatók:

egy: a *belongsTo()* függvénnyel

több:

Példa:  
app/UmpireApplication.php:



app/User.php:



Ezen a módon az *UmpireApplication* modellben elérhető a jelentkezéshez tartozó felhasználó egyszerűen a *user* attribútumon keresztül. (És elérhető a felhasználó minden attribútuma is, pl. *$application->user->name*.) A *User* modellben pedig elérhető lesz a *umpireApplications* attribútum, amin például foreach ciklussal végigiterálhatunk.

A kialakított egy-több kapcsolatok a következők:

|  |  |  |
| --- | --- | --- |
| **egy** | **több** | **kulcs** |
| UmpireLevel | User | umpire\_level |
| RefereeLevel | User | referee\_level |
| Venue | Tournament | venue\_id |
| User | UmpireApplication | umpire\_id |
| User | RefereeApplication | referee\_id |
| Tournament | UmpireApplication | tournament\_id |
| Tournament | RefereeApplication | tournament\_id |

A modellekben szereplő *$with* változó elemei azt jelzik, hogy az adott modell adatbázisból való betöltésekor a kapcsolatban álló modellt is töltse be az Eloquent.

Akár többszörös kapcsolat betöltési igényét is jelezhetjük.

(Pl. az *UmpireApplication* modell ben a ’*tournament.venue’* hatására elérhető ez is: *$application->tournament->venue->address*.)

**Vezérlők**

A vezérlők az eseményeket, felhasználói műveleteket dolgozzák fel és válaszolnak rájuk, az *app/Http/Controllers* mappában találhatók. A HTTP útvonalak definíciója során minden létrehozott útvonalon beérkezett kérést az egyik vezérlő megfelelő metódusa dolgozza fel.

**Általános beállítások**

Szakdolgozatomban minden vezérlő rendelkezik a *\_\_construct()* publikus metódussal, mely a program indulásakor, a vezérlő létrehozásakor fut le. Ebben van lehetőség megadni, hogy a vezérlő metódusai esetlegesen melyik middleware közbeiktatása után dolgozzák fel a beérkezett kérést. A *$this->middleware()* segítségével adhatjuk meg általánosan, melyet később az *only()* és *except()* hívásokkal specializálhatunk. A legtöbb metódus az *admin* middleware-t használja, de vannak, melyeknek az *auth*-ra van csak szükségük.

A vezérlők döntik el azt is, hogy a beérkező kérésnek van-e értelme és a felhasználó, akinek a nevében a kérés érkezett, jogosult-e a kért parancs kezdeményezésére. Ha ez a feltétel nem áll fenn vagy valamilyen más hiba történt (pl. nem sikerült egy modellt az adatbázisból kiolvasni), akkor az *abort()*, *abort\_if()* és *abort\_unless()* függvényhívásokkal a vezérlő kivételt vált ki, melyben beállítja a hibakódot és a hibaüzenetet. Ezeket a kivételeket a keretrendszer elkapja és ha van a HTTP válaszkódnak megfelelő nézet, akkor a hibát azzal jeleníti meg.

A vezérlők metódusai a kérés feldolgozása során szükséges lépések elvégzése után vagy egy nézettel tér vissza (amihez mellékeli a szükséges változókat), vagy átirányít egy másik útvonalra (szintén esetenként addicionális változókkal).

**UserController**

Elsősorban a felhasználó (*User*) modellel kapcsolatban beérkező kérések feldolgozásáért felel.

Metódusai:

* *index*: az összes felhasználó (beleértve a töröltek) listázása
* *show*: megjeleníti a paraméterként megadott azonosítójú felhasználó adatit
* *update*: a paraméterként megadott azonosítójú felhasználó adatait frissíti a kéréshez csatolt értékekre
* *restore*: a paraméterként megadott azonsítójú felhasználót visszaállítja
* *destroy*: a paraméterként megadott azonosítójú felhasználót törli (soft delete)
* *create*: megjeleníti az új felhasználó létrehozásához használatos nézetet
* *store*: a kéréshez csatolt értékek segítségével új felhasználót hoz létre
* *validator*: meghatározza a feltételeket, melyeket a kérésekhez csatolt értékeknek teljesíteniük kell

A vezérlő minden beérkezett kérést csak akkor dolgoz fel, ha az *’admin’* middleware átengedte.

Az összes felhasználó listázásánál a törölt elemeket is listázza (*withTrased()*), mert a nézetben lehetőség van azok megjelenítésére is (visszaállítás céljából). Frissítés és törlés esetén csak a nem törölt felhasználók között keres. A visszaállítás természetesen csak a törölt felhasználók között próbálja a megfelelőt megtalálni.

Ahol a megjelenítéshez szükséges, ott kilistázza és mellékeli a lehetséges döntnöki és játékvezetői szinteket.

A felhasználókat az emailcímük azonosítja, ezért már az adatbázis sem enged új felhasználót létrehozni olyan címmel, ami már szerepel egy rekordban (akár töröltben).

Ha a kérés végrehajtása sikertelen, átirányít a megfelelő útvonalra vagy nézetre és mellékel egy hibaüzenetet (*with(’error’,...)*). Sikeres kezelés esetén erről a tényről ad át üzenetet, ahol erre lehetőség van (*with(’message’,...*).

A jelszavak kezelése a következőképpen történik: a jelszavakat nem sima szövegként tárolja a program az adatbázisban, hanem a keretrendszer által biztosított *Hash* osztály *make()* metódusa által kódolva. Természetesen ebből a kódból nem visszafejthető az eredeti jelszó, csak az ellenőrizhető, hogy a megadott jelszóból kialakuló kód azonos-e a tárolttal. Felhasználó frissítése esetén nem kötelező új jelszót megadni, ebben az eseten az eddig tárolt jelszó érvényben marad.

A validator által ellenőrzött feltételek:

* név: kötelező
* email: kötelező, email formátum, (létrehozás esetén) még nincs ilyen az adatbázisban
* jelszó: (létrehozás esetén) kötelező, létezik a mező
* döntnöki és játékvezetői szint: kötelező

**VenueController**

Elsősorban a csarnok (*Venue*) modellel kapcsolatban beérkező kérések feldolgozásáért felel.

Metódusai:

* *index*: az összes csarnok (beleértve a töröltek) listázása
* *show*: megjeleníti a paraméterként megadott azonosítójú csarnok adatait
* *update*: a paraméterként megadott azonosítójú csarnok adatait frissíti a kéréshez csatolt értékekre
* *restore*: a paraméterként megadott azonosítójú csarnokot visszaállítja
* *destroy*: a paraméterként megadott azonosítójú csarnokot törli (soft delete)
* *create*: megjeleníti az új csarnok létrehozásához használatos nézetet
* *store*: a kéréshez csatolt értékek segítségével új csarnokot hoz létre
* *validator*: meghatározza a feltételeket, melyeket a kérésekhez csatolt értékeknek teljesíteniük kell

A vezérlő minden beérkezett kérést csak akkor dolgoz fel, ha az *’admin’* middleware átengedte.

Az összes csarnok listázásánál a törölt elemeket is listázza (*withTrased()*), mert a nézetben lehetőség van azok megjelenítésére is (visszaállítás céljából). Frissítés és törlés esetén csak a nem törölt csarnokok között keres. A visszaállítás természetesen csak a törölt csarnokok között próbálja a megfelelőt megtalálni.

A csarnokokat a név, rövid név, cím közül bármelyik mező egyértelműen azonosítja, ezért már az adatbázis sem enged új csarnokot azonos adatokkal rendelkező már létező csarnok esetén.

Ha a kérés végrehajtása sikertelen, átirányít a megfelelő útvonalra vagy nézetre és mellékel egy hibaüzenetet (*with(’error’,...)*). Sikeres kezelés esetén erről a tényről ad át üzenetet, ahol erre lehetőség van (*with(’message’,...*).

A validator által ellenőrzött feltételek:

* név: kötelező, (létrehozás esetén) egyedi
* rövid név: kötelező, (létrehozás esetén) egyedi
* cím: kötelező, (létrehozás esetén) egyedi
* pályák száma: egész szám típusú

**TournamentController**

Elsősorban a verseny (*Tournament*) modellel kapcsolatban beérkező kérések feldolgozásáért felel.

Metódusai:

* *index*: az összes verseny (beleértve a töröltek) listázása
* *show*: megjeleníti a paraméterként megadott azonosítójú verseny adatait
* *update*: a paraméterként megadott azonosítójú verseny adatait frissíti a kéréshez csatolt értékekre
* *restore*: a paraméterként megadott azonosítójú versenyt visszaállítja
* *destroy*: a paraméterként megadott azonosítójú versenyt törli (soft delete)
* *create*: megjeleníti az új verseny létrehozásához használatos nézetet
* *store*: a kéréshez csatolt értékek segítségével új versenyt hoz létre
* *validator*: meghatározza a feltételeket, melyeket a kérésekhez csatolt értékeknek teljesíteniük kell
* *intervalDate*: a paraméterként megadott kezdő és befejező dátumot alakítja át intervallum formátumúvá úgy, hogy csak azt az adatot jeleníti meg mindkettőből amelyben már különböznek
* *showCalendar*: a versenynaptár megjelenítése (arra a versenyszezonra vonatkozóan, amit a kérés *’season’* mezője tartalmaz, vagy az aktuálisra, ha nincs ilyen mező)
  + paraméter nélkül: az összes verseny
  + paraméterrel: azok a versenyek, melyekre a megadott paraméterrel rendelkező azonosítójú felhasználó döntnöki vagy játékvezetői szerepben jelentkezett

A vezérlő minden beérkezett kérést – a *showCalendar()* metódust kivéve – csak akkor dolgoz fel, ha az *’admin’* middleware átengedte. A *showCalendar()* metódushoz az *’auth’* middleware szükséges.

Az összes verseny listázásánál a törölt elemeket is listázza (*withTrased()*), mert a nézetben lehetőség van azok megjelenítésére is (visszaállítás céljából). Frissítés és törlés esetén csak a nem törölt versenyek között keres. A visszaállítás természetesen csak a törölt versenyek között próbálja a megfelelőt megtalálni.

Ahol a megjelenítéshez szükséges, ott kilistázza és mellékeli a csarnokok listáját.

A versenyek listázásánál csak azokat a versenyeket adja vissza, amelyek kezdő dátuma a mai napnál nem régebbi.

Az egyetlen metódus, ahol nem egy vagy több modellt ad vissza, a *showCalendar()*. Ez a metódus az, amit a nem adminisztrátor jogosultságú felhasználók leginkább használnak, mert ezen keresztül lehet (felhasználóra szűrve vagy általánosan) a versenynaptárt megjeleníteni. A kért versenyszezon meghatározása után (minden szezon szeptember 1. és augusztus 31. között tart) a szezon összes versenyét feldolgozza, felhasználóra szűrt versenynaptár esetén kigyűjtve hozzájuk a döntnöki és játékvezetői jelentkezéseket. A feldolgozás során új adatszerkezetet állít elő, amely tartalmazza a megjelenítéshez szükséges mezőket. Elegendő lenne itt is modelleket átadni a nézetnek, de igyekeztem a nézetekben a lehetséges legkevesebb logikát elhelyezni és csak a megjelenítésre koncentrálni.

Ha a kérés végrehajtása sikertelen, átirányít a megfelelő útvonalra vagy nézetre és mellékel egy hibaüzenetet (*with(’error’,...)*). Sikeres kezelés esetén erről a tényről ad át üzenetet, ahol erre lehetőség van (*with(’message’,...*).

A validator által ellenőrzött feltételek:

* megnevezés: kötelező
* kezdő dátum: kötelező, dátum formátum, nem későbbi a befejező dátumnál
* befejező dátum: kötelező, dátum formátum, nem korábbi a kezdő dátumnál
* csarnok: kötelező
* igényelt játékvezetők: egész szám típusú

**ApplicationController**

A döntnöki és játékvezetői jelentkezés (*UmprieApplication* és *RefereeApplication*) modellekkel kapcsolatban beérkező kérések feldolgozásáért felel.

Metódusai:

* *addUmpire* és *addReferee*: új jelentkezés
* *removeUmpire* és *removeReferee*: meglévő jelentkezés törlése
* *show*: a paraméterként megadott azonosítójú versenyre való döntnöki és játékvezetői jelentkezések listázása
* *store*: a paraméterként megadott azonosítójú versenyekre való döntnöki és játékvezetői jelentkezések feldolgozott és jóváhagyott státuszainak frissítése
* *add* és *remove*: a kétféle (döntnöki és játékvezetői) művelet olyan kevés eltéréssel renelkezik, hogy a megvalósításuk közös függvényben történik, a *type* paraméter segítségével, melynek lehetséges értékei: *’umpire’* vagy *’referee’*.

A vezérlő minden beérkezett kérést csak akkor dolgoz fel, ha az *’auth’* middleware átengedte.

Ha a kérés végrehajtása sikertelen, átirányít a megfelelő útvonalra vagy nézetre és mellékel egy hibaüzenetet (*with(’error’,...)*). Mivel a jelentkezések hozzáadása vagy törlése esetén is az előző nézet jelenik meg újra, sikeres végrehajtás esetén nincs üzenet.

A validator által ellenőrzött feltételek:

* név: kötelező, (létrehozás esetén) egyedi
* rövid név: kötelező, (létrehozás esetén) egyedi
* cím: kötelező, (létrehozás esetén) egyedi
* pályák száma: egész szám típusú

**EmailController**

Az értesítő email kiküldéséért felel.

Metódusa:

* *sendmail*: a paraméterként megadott azonosítójú versenyre kiküldi a tájékoztató emailt azoknak a felhasználóknak, akiknek a versenyre feldolgozott és jóváhagyott státuszú döntnöki vagy játékvezetői jelentkezésük van

A vezérlő minden beérkezett kérést csak akkor dolgoz fel, ha az *’admin’* middleware átengedte.

Az egyetlen *sendmail()* metódus kigyűjti a szükséges modelleket az adatbázisból és átadja a keretrendszer által biztosított *Mail::send()* metódusnak. Az email sablonja egy nézetben HTML formátumban található meg, amelynek neve a metódus első paramétere.

**Nézetek**

**Blade sablonok**

A Blade egy PHP sablon-motor, amit a Laravel keretrendszer biztosít. Előnye a modularitása mellett az, hogy van benne lehetőség egyszerű PHP kódot is írni a sablonokba.

Minden Blade nézet fájl kiterjesztése*.blade.php*, a *resources/views* mappában vannak.

Az értesítő email sablonja kivételével minden nézet ’szülője’ a *layouts/app* nézet. Ez tartalmazza a HTML oldal head szekcióját és a body alapjait. Itt van az oldal tetején található menü leírása (ami csak akkor jelenik meg, ha bejelentkezett felhasználóként böngészünk, amit az *@auth* Laravel segédfüggvény ellenőriz).

A ’gyerek’ nézetek mindegyike az *@extends(layouts.app)* direktívával kezdődik. Ezután definiálják a *’content’* nevű szekciót a *@yield()* direktívával. Ebből kifolyólag a nézetek nem tartalmazzák egy teljes HTML oldal kódját, csak az *app.blade.php* ’*content’* szekcióját.

A felső menüsor csak bejelentkezett felhasználó számára jelenik meg, mert a weboldal nem tartalmaz olyan részt, ami egy vendég felhasználó számára megjeleníthető lenne.

A nézetek elrendezésének alapja a Bootstrap rácsvonal elrendezése (grid layout). Ennek segítségével definiáltam saját CSS osztályokat, hogy a különböző, de hasonló nézetek egységes külsővel rendelkezzenek. Ezeket a *resources/sass/app.scss* file tartalmazza, melyből az *npm* generálja a *css/app.css* fájlt, amit az *app* Blade nézet header szekciójában beilleszt.

A keretrendszer kényelmesen biztosítja a szokásos programkonstrukciókat, többek között az *@if-@elseif-@else-@endif* és a *@foreach-@endforeach* segédekkel.

A nézetnek átadott változók kiértékelésének módja: {{ $változónév }}.

**Listázó nézetek**

Tartozik listázó nézet a felhasználókhoz, a csarnokokhoz és a versenyekhez.

A nézetek tetején (a menüsor alatt) a bal oldalon egy gomb található, mely a vonatkozó modell új elemének létrehozására használatos útvonalra mutat. A jobb oldalon egy jelölőnégyzet van (*’Töröltek megjelenítése’*), melynek állapota elem törlésekor és visszaállításakor nem változik.

A definiált saját stílusok segítségével ezután először a rácsvonal által megvalósított táblázat fejlécét jeleníti meg, majd a vezérlő által átadott gyűjteményen végigiterálva kitölti a sorokat.

Ha a sorhoz tartozó elem törölt, a sor osztályai között szerepelni fog a *’deleted\_row’*, mely csak akkor jelenik meg, ha a *’Töröltek megjelenítése’* jelölőnégyzet be van jelölve.

Minden sor tartalmaz *’Szerkesztés’* és *’Törlés’* vagy *’Visszaállítás’* gombokat, melyek az adott modell vonatkozó műveletéhez tartozó útvonalra mutatnak. A törlés és a visszaállítás után újra a lista nézet jelenik meg, a szerkesztés az *’Szerkesztés’* nézetre mutat.

**Szerkesztés nézetek**

**Tesztelés**