Tartalom

[Bevezetés 3](#_Toc133482565)

[Fejlesztői dokumentáció 4](#_Toc133482566)

[Felhasznált technológiák 4](#_Toc133482567)

[Backend 4](#_Toc133482568)

[Készítéshez használt programok 4](#_Toc133482569)

[Frontend 4](#_Toc133482570)

[Alapkönyvtárak 4](#_Toc133482571)

[Felületterv 5](#_Toc133482572)

[Végpontok 6](#_Toc133482573)

[Általános működés: 7](#_Toc133482574)

[Osztályok 8](#_Toc133482575)

[BaseController 8](#_Toc133482576)

[AuthController 9](#_Toc133482577)

[DiscountController 10](#_Toc133482578)

[PassController 12](#_Toc133482579)

[PasstypeController 13](#_Toc133482580)

[UserController 15](#_Toc133482581)

[Modellek 17](#_Toc133482582)

[CityRegion 17](#_Toc133482583)

[Discount 17](#_Toc133482584)

[Pass 18](#_Toc133482585)

[Passtype 18](#_Toc133482586)

[Region 18](#_Toc133482587)

[User 19](#_Toc133482588)

[Webes része 20](#_Toc133482589)

[JavaScriptek és főbb függvényeik 20](#_Toc133482590)

[Tesztelés 22](#_Toc133482591)

[Továbbfejlesztési lehetőségek 25](#_Toc133482592)

[Felhasználói dokumentáció 26](#_Toc133482593)

[Ismertető 26](#_Toc133482594)

[GYMSTER 27](#_Toc133482595)

[Főoldal 27](#_Toc133482596)

[Áraink menüpont 28](#_Toc133482597)

[Regisztráció menüpont 29](#_Toc133482598)

[Bejelentkezés menüpont 30](#_Toc133482599)

[Profil menüpont 31](#_Toc133482600)

[Kijelentkezés menüpont 35](#_Toc133482601)

[Ami az adminisztrátornál más: 35](#_Toc133482602)

[Összefoglalás 39](#_Toc133482603)

# Bevezetés

A GYMSTER egy (kitalált) interneten elérhető edzőtermi weblap, amely egy nyilvántartó rendszert biztosít, emellett plusz funkciókat kínál az adminisztrátorok számára.

A weboldal célja, hogy a látogatók könnyedén megnézhessék az adott hétre tervezett edzéstervet és az edzőket.

Az oldal egyszerű felülettel rendelkezik, amelyet egy menüsor segítségével lehet navigálni. A látogatóknak lehetősége van regisztrálni és belépni a weboldalra, ahol bérletet vehetnek és személyes adataikat személyre szabhatják.

A GYMSTER fő célja az egészséges életmód elősegítése, a helybeni regisztráció elkerülése, valamint a bérletvásárlás folyamatának egyszerűsítése. Bár az oldal jelenleg egy korai verzióban áll rendelkezésre, a cél az, hogy a jövőben még több funkcióval bővüljön.

Az edzőtermeknek egyre nagyobb kihívásokkal kell szembenézniük a fennmaradás és a növekedés érdekében.

Az internetes elérhetőség egyre fontosabbá válik, az ügyfélkör bővítése és az ügyfélelégedettség növelése érdekében. Az edzőtermeknek olyan online megoldásokat kell kínálniuk, amelyek egyszerűvé teszik a fontosabb ügyintézéseket anélkül, hogy személyesen meg kelljen jelenni az edzőteremben.

# Fejlesztői dokumentáció

## Felhasznált technológiák

### Backend

* Laravel API (PHP keretrendszer)

-laravel sanctum (Autentikációs kiegészítő csomag)

* MariaDB szerver (Adatbázis kiszolgáló)

### Készítéshez használt programok

* dia
* Visual studio code 1.77.3
* Insomnia 2023. 1.0
* Mariadb server
* phpmyadmin

### Frontend

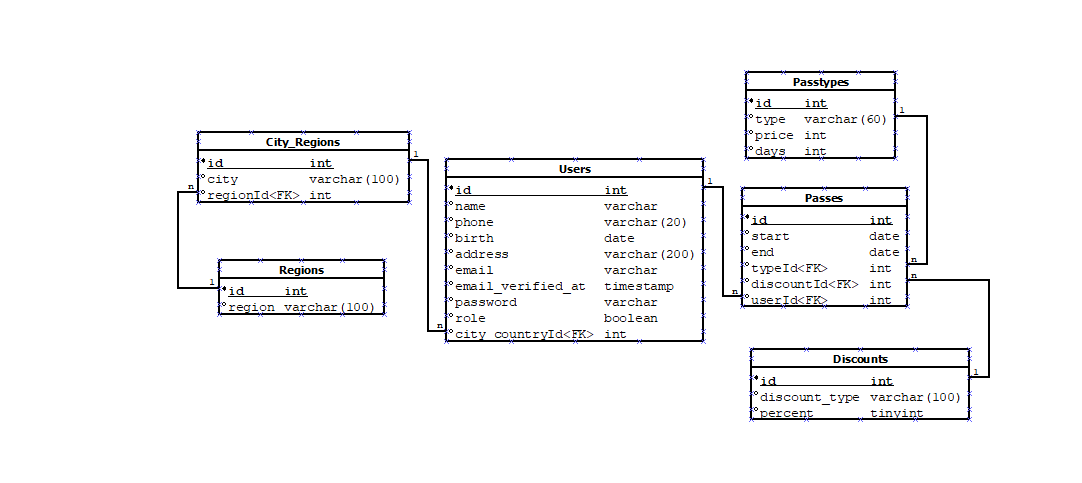
* HTML
* JavaScript
* CSS

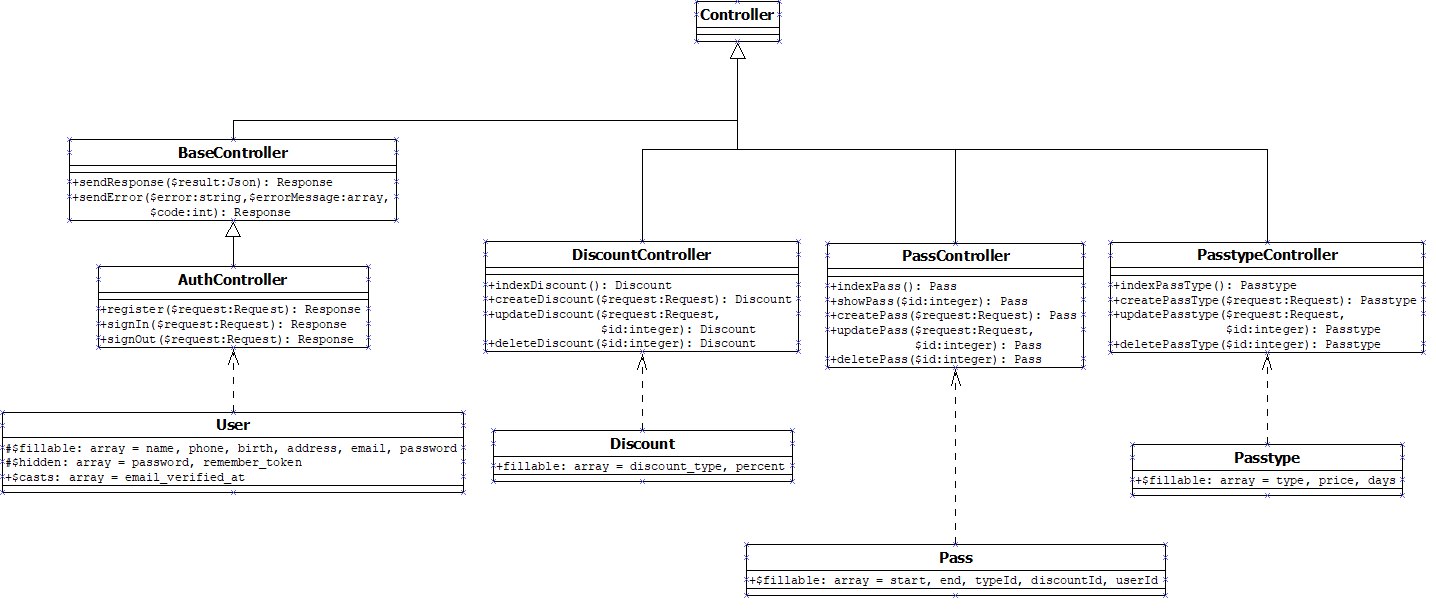
## Alapkönyvtárak

* api
* database
* docs

Az api a Laraveles backend, a database-ben van a terv fájlok, a docs-ban a dokumentáció, és a frontend a Vizsgaremek2 mappában van mert, nincsen mappákba elrendezve.

## Felületterv

Adatmodell

UML terv

## Végpontok

|  |  |  |  |
| --- | --- | --- | --- |
| Végpont | Metódus | Azonosítás | Leírás |
| /register | post | nem | Felhasználó regisztrációja |
| /login | post | nem | Felhasználó bejelentkezése |
| /logout | post | igen | Felhasználó kijelentkezése |
| /users | get | igen | Felhasználók lekérdezése |
| /user/{id} | get | igen | Felhasználóra keres id szerint |
| /passtypes | get | nem | Bérlettípusok lekérdezése |
| /user/{id} | put | igen | Felhasználó módosítása id alapján |
| /admin/user/{id} | delete | igen | Felhasználó törlése id alapján |
| /admin/user | post | igen | Admin felvétele |
| /discounts | get | nem | Kedvezmények lekérdezése |
| /discount | post | igen | Kedvezmény létrehozása |
| /discount/{id} | put | igen | Kedvezmény módosítása |
| /discount/{id} | delete | igen | Kedvezmény törlése |
| /passes | get | igen | Bérletek lekérdezése |
| /pass | post | igen | Bérletek felvétele |
| /passtype | post | igen | Bérlettípus felvétele |
| /passtype/{id} | put | igen | Bérlettípus módosítása |
| /passtype/{id} | delete | igen | Bérlettípus törlése |

### Általános működés:

A REST API http kéréseket fogad, melyek tartalmazzák a műveletekhez szükséges megfelelő adatokat.

A kényes műveletek végpontjai védettek, autentikáció, és admin bejelentkezés szükséges a használatukhoz. Ilyenek az új admin felvétele, felhasználók bérletének lekérdezése, felhasználó törlése, kedvezmények létrehozása, módosítása és törlése, bérletek lekérése, létrehozása, módosítása, illetve törlése, bérlettípus létrehozása, módosítása és bérlettípus törlése.

A bejelentkezést igénylő védett végpontok is autentikációval vannak védve, de nem igényel adminisztrátori szintű jogokat. Ilyen a kijelentkezés, felhasználó saját adatainak frissítése és a bérlet lekérdezése.

A regisztráció, bejelentkezés, bérlettípusok lekérdezése, kedvezmények lekérése és a felhasználók keresésének, és lekérdezésének végpontjai publikusak, ezek használatához autentikáció nem szükséges.

Az adatokat a rendszer Json formátumban fogadja és dolgozza fel. A vezérlést kontrollerek valósítják meg, minden adatkezelési csoportnak (autentikáció, kedvezmény, bérlet, bérlettípusok, felhasználó) külön kontrollere van, itt történik az adatfeldolgozás.

A kontrollerek különböző modellekkel vannak kapcsolatban, amelyek az adatkezelésért felelősek. Minden adatkezelési csoportnak (megye, városmegye, kedvezmény, bérlet, bérlettípusok, felhasználó) külön modellje van, itt történik az adatok adatbázisból kiolvasása, illetve az adatok kiírása adatbázisba. A modellek az adatbázis táblákkal vannak kapcsolatban, melyek az adatok tárolásáért felelősek. Az adatbázis táblák adatait a modellek kezelik.

## Osztályok

### BaseController

Feladata a bejelentkezési műveletek válaszainak küldése. Sikeres művelet esetén a nevet, a szerepet, és a generált azonosító tokent visszaküldi, sikertelen művelet esetén hibaüzenetet küld.

#### Metódusok:

##### sendResponse($result)

A metódus JSON választ küld vissza, amelyben az adatokat tartalmazó ’result’ paraméter lesz a válasz.

Bejövő paraméterek:

* $result (kötelező): Az adatokat tartalmazó paraméter.

Kimenő adatok:

* A metódus egy JSON választ küld vissza, amely az adatokat tartalmazza a result paraméterben, a HTTP válaszkóddal 200.

##### sendError($error, $errorMessage = [], $code = 404)

A metódus egy hibás JSON választ küld vissza, amely az ’error’ és az opcionális ’errorMessage’ paramétereket tartalmazza. A válasz kódja az opcionális ’code’ paraméterrel adható meg, ha az nincs megadva, akkor az alapértelmezett 404-es HTTP státuszkódot veszi használatba.

Bejövő paraméterek:

* $error (kötelező): Az általános hibaüzenet.
* $errorMessage (opcionális): A hiba részletező üzenete.
* $code (opcionális): Az HTTP válaszkód.

Kimenő adatok:

* A metódus egy hibás JSON választ küld vissza, amely az ’error’ és az opcionális ’errorMessage’ paramétereket tartalmazza, az opcionális ’code’ HTTP válaszkóddal.

### AuthController

Feladata egy új felhasználó felvétele, felhasználók autentikációja, felhasználók kijelentkeztetése.

#### Metódusok:

##### register()

Ezzel a metódussal a felhasználó regisztrálhat az alkalmazásba.

Bejövő paraméterek:

* $request: A Laravel Request objektum, amely tartalmazza a beérkező kérést.
  + "name": A felhasználó neve,
  + "email": A felhasználó e-mail címe,
  + "password": A felhasználó jelszava,
  + "confirm\_password": A felhasználó jelszavának megerősítése.

Kimenő adatok:

* $success: A regisztrált felhasználó neve.
* "Sikeres regisztráció" üzenet.

##### signIn()

Feladata a felhasználó azonosítása email és jelszó alapján. Sikeres autentikáció esetén generál egy tokent a felhasználó számára és bejegyzi a personal\_access\_tokens adatbázis tábla megfelelő mezőjébe, majd átadja a BaseController sendResponse() metódusának a saját üzenettel együtt.

Bejövő paraméterek:

* $request: A Laravel Request objektum, amely tartalmazza a beérkező kérést.
  + "email": A felhasználó e-mail címe.
  + "password": A felhasználó jelszava.

Kimenő adatok:

* $success: Az autentikált felhasználó adatai:
  + "id": Az autentikált felhasználó azonosítója.
  + "name": Az autentikált felhasználó neve.
  + "email": Az autentikált felhasználó e-mail címe.
  + "role": Az autentikált felhasználó szerepköre.
  + "token": Az autentikált felhasználó tokenje.

##### signOut()

Feladata a felhasználó kijelentkeztetése és a token törlése az adatbázis táblából.

Bejövő paraméterek:

* $request (a kijelentkezéshez szükségek adatok a kérésben, token)

Kimenő adatok:

* "Sikeres kijelentkezés" üzenet.

### DiscountController

Feladatai egy új kedvezmény felvétele, összes kedvezmény lekérdezése és kedvezmény frissítése és törlése id alapján.

#### Metódusok:

##### indexDiscount()

Visszaadja az adatbázisban szereplő kedvezményeket egybe.

Bejövő paraméterek:

* nincsenek

Kimenő adatok:

* a lekérdezett kedvezmények listája

##### createDiscount()

Eltárolja az adatbázisban az új kedvezményt.

Bejövő paraméterek:

* a frissítendő kedvezmény azonosítója, és a frissítendő kedvezmény adatai, JSON formátumban

Kimenő adatok:

* a frissített kedvezmény adatai JSON formátumban, és egy sikerüzenet

##### updateDiscount()

Frissíti az adatbázisban tárolt egy kedvezmény adatait.

Bejövő paraméterek:

* a frissítendő kedvezmény azonosítója, és a frissítendő kedvezmény adatai, JSON formátumban.

Kimenő adatok:

* a frissített kedvezmény adatai JSON formátumban, és egy sikerüzenet.

##### deleteDiscount()

Feladata a kiválasztott kedvezmény törlése az adatbázisból.

Bejövő paraméterek:

* a törölni kívánt kedvezmény azonosítója

Kimenő adatok:

* egy sikerüzenet

### PassController

Feladata a bérletek lekérése, új bérlet felvétele, meglévő bérlet módosítása és törlése.

#### Metódusok

##### indexPass()

A metódus visszaadja az összes bérletet az adatbázisból.

Bejövő paraméterek:

* Nincsenek

Kimenő adatok:

* Az összes bérletet tartalmazó modell

##### createPass(Request $request)

A metódus létrehoz egy új bérletet az adatbázisban.

Bejövő paraméterek:

* $request - A kérés, amely tartalmazza az új bérlet adatait

Kimenő adatok:

* Az újonnan létrehozott bérletet tartalmazó modell, vagy hibaüzenet, ha a bemenet érvénytelen.

##### updatePass(Request $request, $id)

A metódus frissíti a kiválasztott bérlet adatait az adatbázisban.

Bejövő paraméterek:

* $request - A kérés, amely tartalmazza a frissítendő bérlet adatait.
* $id - a kiválasztott bérlet azonosítója

Kimenő adatok:

* Az frissített bérletet tartalmazó modell, vagy hibaüzenet, ha a bemenet érvénytelen.

##### deletePass($id)

A metódus törli a kiválasztott bérletet az adatbázisból.

Bejövő paraméterek:

* $id - a törölni kívánt bérlet azonosítója

Kimenő adatok:

* A törölt bérlet azonosítója, vagy hibaüzenet, ha nem létezik a bérlet.

### PasstypeController

Feladata a bérlettípusok lekérése, új bérlettípus felvétele, típus módosítása és bérlettípus törlése.

#### Metódusok:

##### indexPassType()

Visszatér a Passtype model összes típusával JSON formátumban.

Bejövő paraméterek:

* Nincsenek.

Kimenő adatok:

* A Passtype model összes típusa JSON formátumban.

##### createPassType(Request $request)

Létrehoz egy új típust a Passtype model-ben a megadott adatok alapján.

Bejövő paraméterek:

* $request: Az HTTP kérés objektum.

Kimenő adatok:

* Ha a típus létrehozása sikeres, akkor visszatér a létrehozott típus adataival és a "Típus hozzáadva" szöveggel JSON formátumban.
* Ha a típus létrehozása nem sikerült, akkor hibás validációs adatokkal és az "Érvénytelen bemenet" szöveggel tér vissza.

##### updatePassType(Request $request, $id)

Frissíti a Passtype model-ben a megszabott azonosítójú típus adatait a megadott adatok alapján.

Bejövő paraméterek:

* $request: Az HTTP kérés objektum.
* $id: Az azonosító a típus frissítéséhez.

Kimenő adatok:

* Ha a típus frissítése sikeres, akkor visszatér a frissített típus adataival és a "Típus sikeresen frissítve" szöveggel JSON formátumban.
* Ha a típus frissítése nem sikerült, akkor hibás validációs adatokkal tér vissza.

##### deletePassType($id)

* Törli a Passtype model-ben a megadott azonosítójú típust.

Bejövő paraméterek:

* $id: Az azonosító a típus törléséhez.

Kimenő adatok:

* A "Típus törölve" szöveggel tér vissza JSON formátumban, ha a típus sikeresen törölve lett.

### UserController

Felhasználóknak kezelésére szolgáló vezérlő. Az alapértelmezett Laravel Auth rendszert használja.

#### Metódusok:

##### index()

Visszaadja az összes felhasználót JSON formátumban.

Bejövő paraméterek:

* Nincsenek.

Kimenő adatok:

* Az összes felhasználó adatai JSON formátumban.

##### addAdmin(Request $request)

Létrehoz egy új felhasználót az admin jogosultsággal a megadott adatok alapján.

Bejövő paraméterek:

* $request: Az HTTP kérés objektum.

Kimenő adatok:

* Ha a felhasználó létrehozása sikeres, akkor visszatér a létrehozott felhasználó adataival és az "Felhasználó hozzáadva" szöveggel JSON formátumban.
* Ha a felhasználó létrehozása nem sikerült, akkor hibás validációs adatokkal és az "Érvénytelen bemenet" szöveggel tér vissza.

##### show($id)

Betölti a megadott azonosítójú felhasználó adatait JSON formátumban.

Bejövő paraméterek:

* $id: Az azonosító a felhasználó betöltéséhez.

Kimenő adatok:

* Ha a felhasználó azonosítója létezik, akkor visszatér a felhasználó adataival és a "Felhasználó betöltve" szöveggel JSON formátumban.
* Ha a felhasználó azonosítója nem létezik, akkor az "Felhasználó nem létezik" szöveggel tér vissza.

##### update(Request $request, $id)

Frissíti a felhasználói profil adatait a megadott adatok alapján.

Bejövő paraméterek:

* $request: Az HTTP kérés objektum.
* $id: Az azonosító a felhasználó frissítéséhez.

Kimenő adatok:

* Ha a felhasználói profil frissítése sikeres, akkor a megváltoztatott felhasználót az új adatokkal írja vissza.

## Modellek

### CityRegion

A CityRegion osztály egy Eloquent modell, amely a "city\_regions" adatbázis táblát reprezentálja.

Az osztály $fillable tömbjében meghatározott mezőkkel (jelen esetben a "city" és "regionId" mezőkkel) lehet közvetlenül feltölteni az adatbázis táblát a modell új példányának létrehozása során.

Az osztályban az időbélyegek kikapcsolt állapotban vannak a $timestamps változó beállításával.

Az osztály a User és a Region osztályokkal áll kapcsolatban. A "users" metódus visszaadja az összes olyan felhasználót, akiknek van kapcsolatuk a CityRegion példánnyal, míg a "regions" metódus visszaadja az adott CityRegion-hez tartozó Region példányt.

### Discount

A Discount osztály egy Eloquent modell, amely a "discounts" adatbázis táblát reprezentálja.

Az osztály $fillable tömbjében meghatározott mezőkkel (jelen esetben a "discount\_type" és "percent" mezőkkel) lehet közvetlenül feltölteni az adatbázis táblát a modell új példányának létrehozása során.

Az osztályban az időbélyegek kikapcsolt állapotban vannak a $timestamps változó beállításával.

Az "discount\_type" mező azt határozza meg, hogy milyen típusú kedvezményt kínálunk (például állandó vagy időszakos kedvezmény), míg a "percent" mező az adott kedvezmény százalékos mértékét adja meg.

### Pass

A Pass osztály egy Eloquent modell, amely a "passes" adatbázis táblát reprezentálja.

Az osztály $fillable tömbjében meghatározott mezőkkel (jelen esetben a "start", "end", "typeId", "discountId" és "userId" mezőkkel) lehet közvetlenül feltölteni az adatbázis táblát a modell új példányának létrehozása során.

Az osztályban az időbélyegek kikapcsolt állapotban vannak a $timestamps változó beállításával.

A "start" és "end" mezők az adott bérlet érvényességi idejét határozzák meg, míg a "typeId" mező az adott bérlet típusát, a "discountId" mező az adott bérletre vonatkozó kedvezményt és az "userId" mező pedig az adott bérlet tulajdonosát tartalmazza.

### Passtype

A Passtype osztály egy Eloquent modell, amely a "passtypes" adatbázis táblát reprezentálja

Az osztály $fillable tömbjében meghatározott mezőkkel (jelen esetben a "type" és "price" mezőkkel) lehet közvetlenül feltölteni az adatbázis táblát a modell új példányának létrehozása során.

Az osztályban az időbélyegek kikapcsolt állapotban vannak a $timestamps változó beállításával.

### Region

A Region osztály a "regions" adatbázis táblát nézi. Kikapcsolja az időbélyegeket a $timestamps változóval.

Az osztály $fillable tömbjében meghatározott mezőkkel (jelen esetben csak a "region" mezővel) lehet közvetlenül feltölteni az adatbázis táblát a modell új példányának létrehozása során.

A "cityregions" függvény egy "hasMany" kapcsolatot definiál a CityRegions modellhez.

### User

Az User osztály egy Eloquent modell, amely a "users" adatbázis táblát reprezentálja. Az osztály a Laravel-ben található, Authenticatable osztályból származik, ami lehetővé teszi a felhasználók autentikációját.

Az osztályban a $fillable tömbben meghatározott mezőkkel (jelen esetben a "name", "phone", "birth", "address", "email" és "password" mezőkkel) lehet közvetlenül feltölteni az adatbázis táblát a modell új példányának létrehozása során.

Az osztályban található roles() függvény segítségével lekérhetjük az adott felhasználó szerepeit, míg a cityregions() függvénnyel az adott felhasználóhoz tartozó városi régiókat kérhetjük le.

Az osztályban az időbélyegek kikapcsolt állapotban vannak a $timestamps változó beállításával.

## Webes része

### JavaScriptek és főbb függvényeik

#### add\_admin

API-ról lekéri az összes felhasználót és a táblázatban azokat jeleníti meg, ahol a felhasználó email címe "@mod.com"-mal végződik. Ha a felhasználó email címe végén "@admin.com" található, akkor az összes felhasználót megjeleníti a táblázatban, beleértve a törlés gombot is. Az eredményt az oldalon a ’renderTable()’ függvény segítségével jeleníti meg.

#### admin\_discount

A kód célja a kedvezmények átfogó kezelése:

* Kedvezmények megjelenítése egy táblázatban ( getDiscounts() )
* Kedvezmények törlése ( deleteDiscount(id) )
* Kedvezmények hozzáadása ( addDiscount() )
* Kedvezmények szerkesztése (updateDiscount() )

#### admin\_price

A kód elején lévő ’localStorage.getItem’ funkció a ’token’ változóban tárolja az aktuális felhasználó tokenjét. Ha a felhasználó nincs bejelentkezve, akkor a kód automatikusan átirányítja a felhasználót a bejelentkezési oldalra.

Ezután egy táblázat jelenik meg az oldalon, amelyben megjelennek az eddig létrehozott jegytípusok. Az oldal lehetővé teszi új jegytípusok hozzáadását, meglévő jegytípusok törlését és szerkesztését.

#### login

A ’loginUser’ felelős a felhasználói adatok elküldéséért (a szervernek), emellett az eredmények alapján átirányítja a felhasználót (’role’ szerint) a megfelelő weboldalakra. Az „emailLog” és „passwordLog” változók tárolják az email és jelszó beviteli mezők elemét későbbi azonosítások érdekében.

#### logout

A logout() függvény felelős a felhasználórendszerből történő kijelentkeztetését. Az autentikációhoz a ’localStorage’-ban eltárolt ’token’-t használja. Amennyiben a kérés sikeres volt, akkor törli az eltárolt ’token’-t az adatbázisból, majd átirányítja a felhasználót a login.html oldalra.

#### pass

A felhasználók bérleteinek kezelését végzi.

Dátumok kezelése: meghatározza a mai dátumot, majd lekéri az aktuális hónap nevét. Ezután összeállít egy dátumot, amelyet később egy másik funkciónál használ a bérletek kezeléséhez. A ’getPasses()’ függvény lekérés után megvizsgálja hogy van-e aktuális bérlete a felhasználónak.

Az ’inputType()’ függvény bekéri egy listába a lehetséges bérlettípusokat, és változtatás esetén a hozzá tartozó árat is kiírja. Az ’inputDiscount()’ hasonlóan működik, viszont, ha a felhasználó választja a kedvezményt, akkor a rendszer az árat is újraszámolja.

#### profile

#### ’ UserProfile() ’: Ez a függvény egy API hívást indít, amely a felhasználói profil adatokat tölti be az adatbázisból. Az adatokat a kódban található input mezőkbe írja be.

#### ’ UpdateProfile()’ : Ez a függvény egy másik API hívást indít, amely a felhasználói profil adatokat frissíti az adatbázisban. Az adatokat a kódban található input mezőkből veszi.

#### register

Űrlap segítségével felhasználói regisztrációt valósít meg. A ’nameReg’, ’emailReg’, ’passwordReg’ és ’confirm\_passwordReg’ változók segítségével meghatározzuk az űrlapban szereplő mezőket, amelyek tartalmát a felhasználó adja meg, majd a ’registerUser()’ függvény segítségével JSON formátumban elküldi azt a szerverre, ha sikeres volt.

#### user\_prices

Kiírja a felhasználóknak a létező bérlet és kedvezmény típusokat. A ’getPasstypes()’ és a ’getDiscounts()’ függvények meghívásra kerülnek az oldal betöltésekor, amelyek visszatérnek a szerver által adott adatokkal. A renderPasstypeTable() és a renderDiscountTable() függvények ezeket az adatokat használják a megjelenítendő táblázatok létrehozásához.

## Tesztelés

Teszteléshez 2 fiókot használtunk, ami bele van égetve a Laravel egyik seederébe.

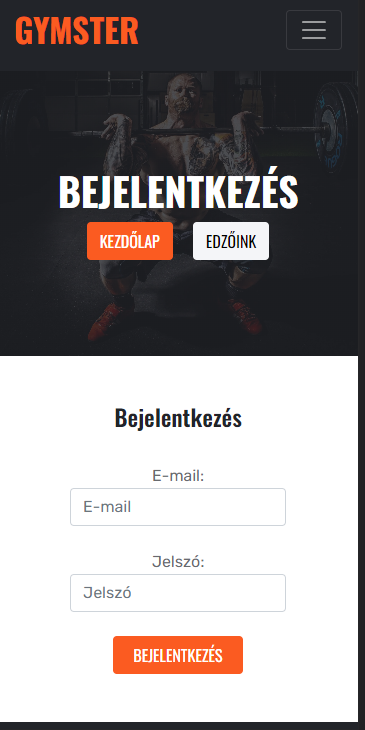
Admin: Felhasználó:

név: admin név: user

email: [admin@admin.com](mailto:admin@admin.com) email: user@user.com

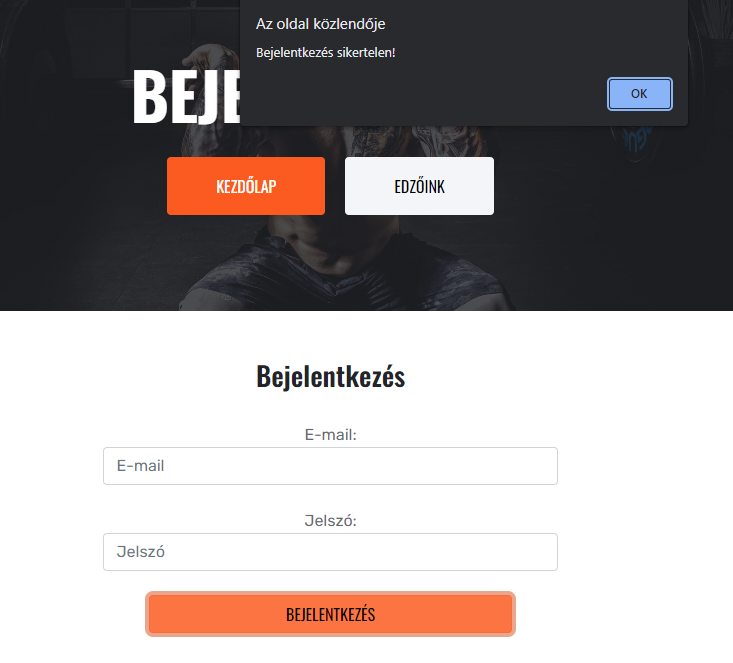
jelszó: admin jelszó: user

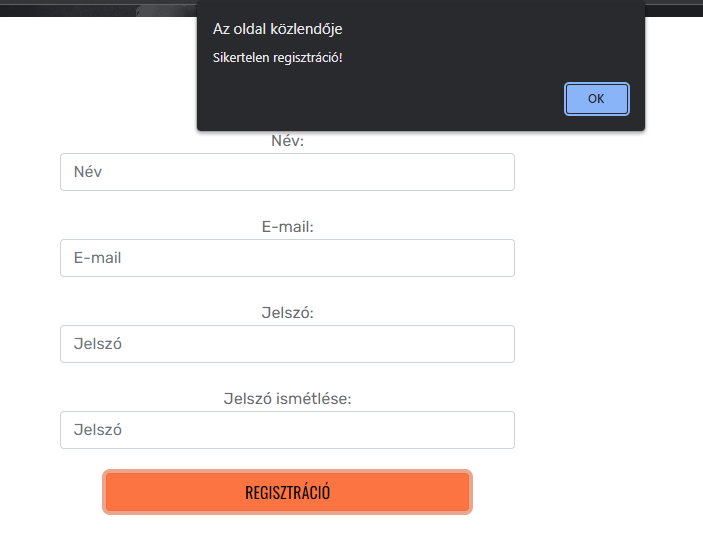
A projekten való dolgozást Google Chrome-n és Microsoft Edge-n végeztük. Legtöbbet számítógépen végeztük a teszteléseket (Windows 10 és 11-en), és a funkciók működését, de az oldal reszponzivitása miatt, telefonon is meg tudtuk nézni, hogy hogy néz ki a végső projekt.



(360x740 felbontásban)

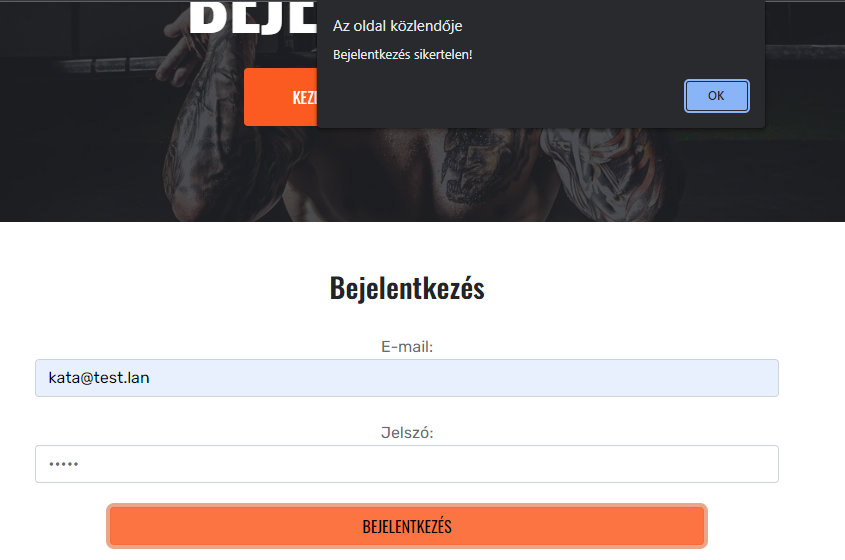
Ha üres adatokkal akarnánk belépni, vagy regisztrálni, kidob az oldal tetejére egy alert üzenetet, ezzel is jelezve, hogy sikertelen volt.





Ugyan ez igaz a félig-meddig kitöltött mezőkre is. Csak a teljesen kitöltötteket fogadja el.

Belépésnél megnézi, hogy van e olyan felhasználó és csak akkor engedi tovább:

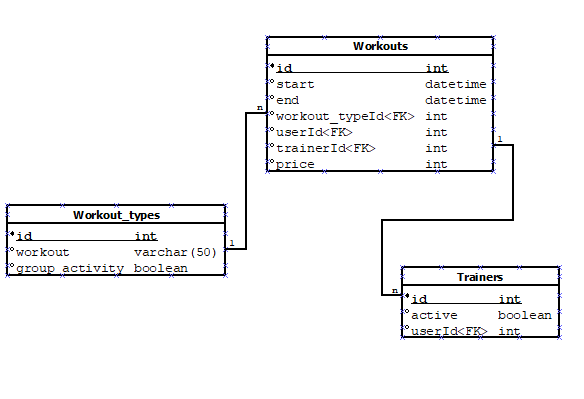




…

## Továbbfejlesztési lehetőségek

Adatmodell:



Későbbiekben lehetne bővíteni, hogy könnyebben testreszabhatóbb legyen. Négy táblával lehetne bővíteni: Edzések, Edzés típusok, Edzők és Szerepek.

Adminisztrátorok kezelnék a főbb műveleteket, mint például az Edzők felvételét, hasonlóan, mint a moderátor/admin felvételét a jelenlegi verziónál. Esetlegesen, ha abba akarja hagyni egy ideig az edzések tartását, átrakhatná magát „inaktívba”, az „active” mezőben.

Ilyen még a „Edzések” tábla, ahol lehetne bővíteni a jelenlegi minta edzéseket, akár részletesebben is. Emellé jönne az „Edzés típusok”, ahol bővíteni lehetne kedvünk szerint az edzéseket és hogy csoportos foglalkozás e, vagy csak pl.: személyi edzés.

Mindez az „Users” táblával lenne összekötve, mégpedig a „Workouts” táblában lévő userId-vel ami az id-re mutat.

# Felhasználói dokumentáció

## Ismertető

A GYMSTER egy edzőtermi nyilvántartó rendszer és weboldal. Regisztráció nem szükséges a böngészéséhez, de lehetőség van további funkciók elérésére regisztrációval, illetve bejelentkezéssel.

Bejelentkezés után személyre szabható profilt és bérletet találunk.

#### Rendszerkövetelmények

* Internetes hozzáférés
* Internetes böngésző (pl. Chrome, Opera, Edge)

#### Futtatása

Amire szükség van:

* XAMPP Control Panel
  + phpMyAdmin
* Valamilyen kódszerkesztő, ahol tudunk parancsot futtatni
* Composer
* Github linkje: <https://github.com/tailorbrigit/Vizsgaremek2.git>

Tetszőleges helyre letöltjük a fájlt a „Download ZIP” gombbal (a „Code” zöld fül alatt)

Kicsomagoljuk a ZIP-et ahova szeretnénk,

majd megnyitjuk benne az ’api/gym\_registry’ mappát egy kódszerkesztővel, és terminálba írjuk hogy:

* composer install

majd készítunk egy másolatot az ’.env.example’ fájlról, és elnevezzük úgy, hogy ’.env’ majd a ’DB\_DATABASE=’ után írjuk, hogy gym\_registry.

ezután elindítjuk a XAMPPban az Apache-t és a MySql-t.

Majd megnyitjuk a ’ <http://localhost/phpmyadmin/’-t> és létrehozunk egy gym\_registry táblát,utf-8\_hungarian\_ci-vel.  
  
ha elindult mind a kettő, és táblánk is van, az api/gym\_registry termináljába beírjuk az migráció seeder, és indító parancsot:

* php artisan migrate
* php artisan db:seed
* php artisan serve

ezek után a szerver már fut, és csak annyi a dolgunk, hogy megnyissuk az index.html-t a fő mappában.

Admin bejelentkezési adatok:

Email: [admin@admin.com](mailto:admin@admin.com)

Jelszó: admin

User bejelentkezési adatok:

Email: [user@user.com](mailto:user@user.com)

Jelszó: user

## GYMSTER

### Főoldal

A főoldal fogad minket vendégként amint megnyitjuk az index.html-t.



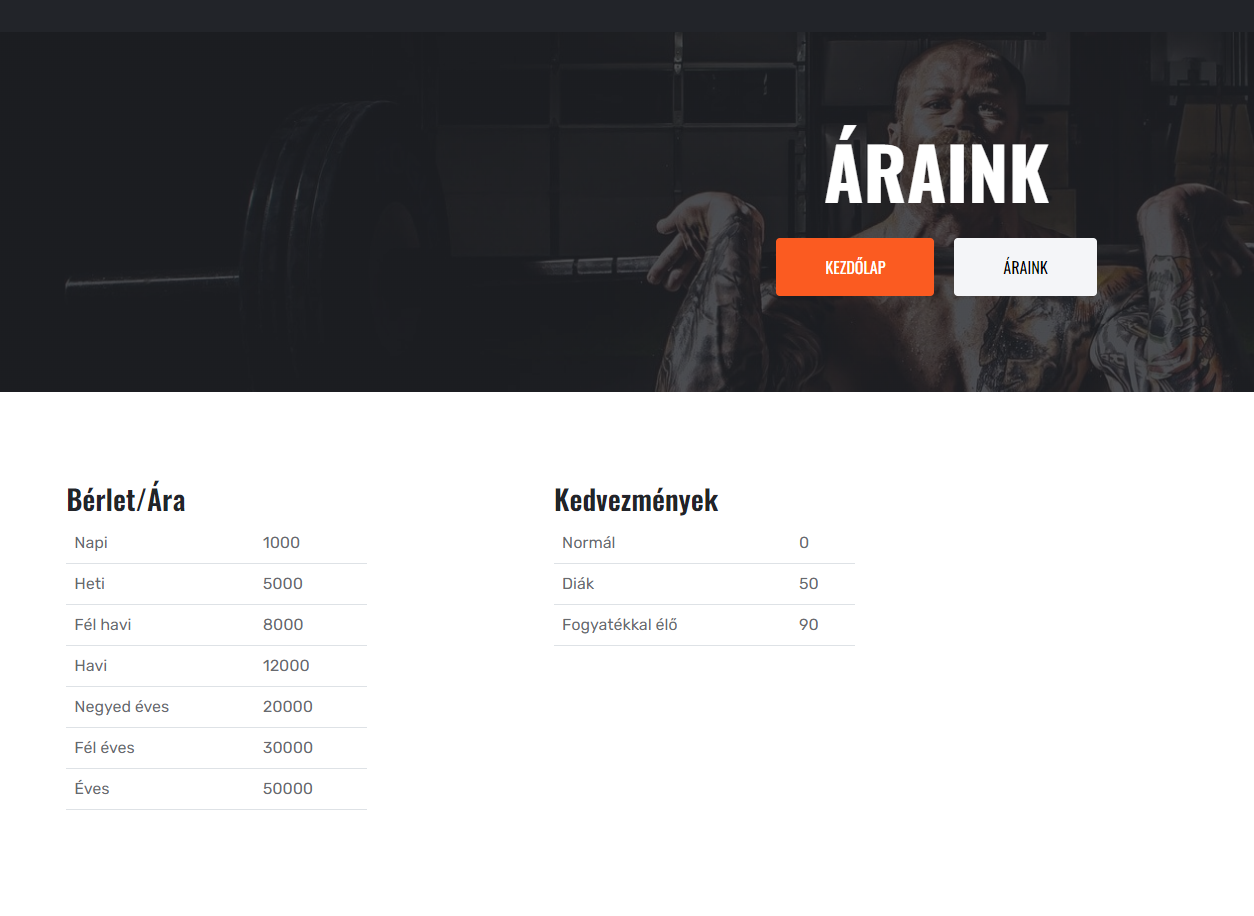
Lejjebb találhatóak még nem funkcionális részek is, viszont továbbfejlesztésnél célunk ezeknek a működtetése is, mint például az edzések fejlesztése.

Felül található egy menüsáv, ahol a weboldal rövidítése látható, és pár menüpont:

* + Kezdőlap
  + Rólunk
  + Osztályok
  + Edzőink
  + Oldalak
    - Blog
    - Blog
    - Ajánlólevél
  + Kapcsolat
  + Áraink (Funkcionális)
  + Regisztráció (Funkcionális)
  + Bejelentkezés (Funkcionális)

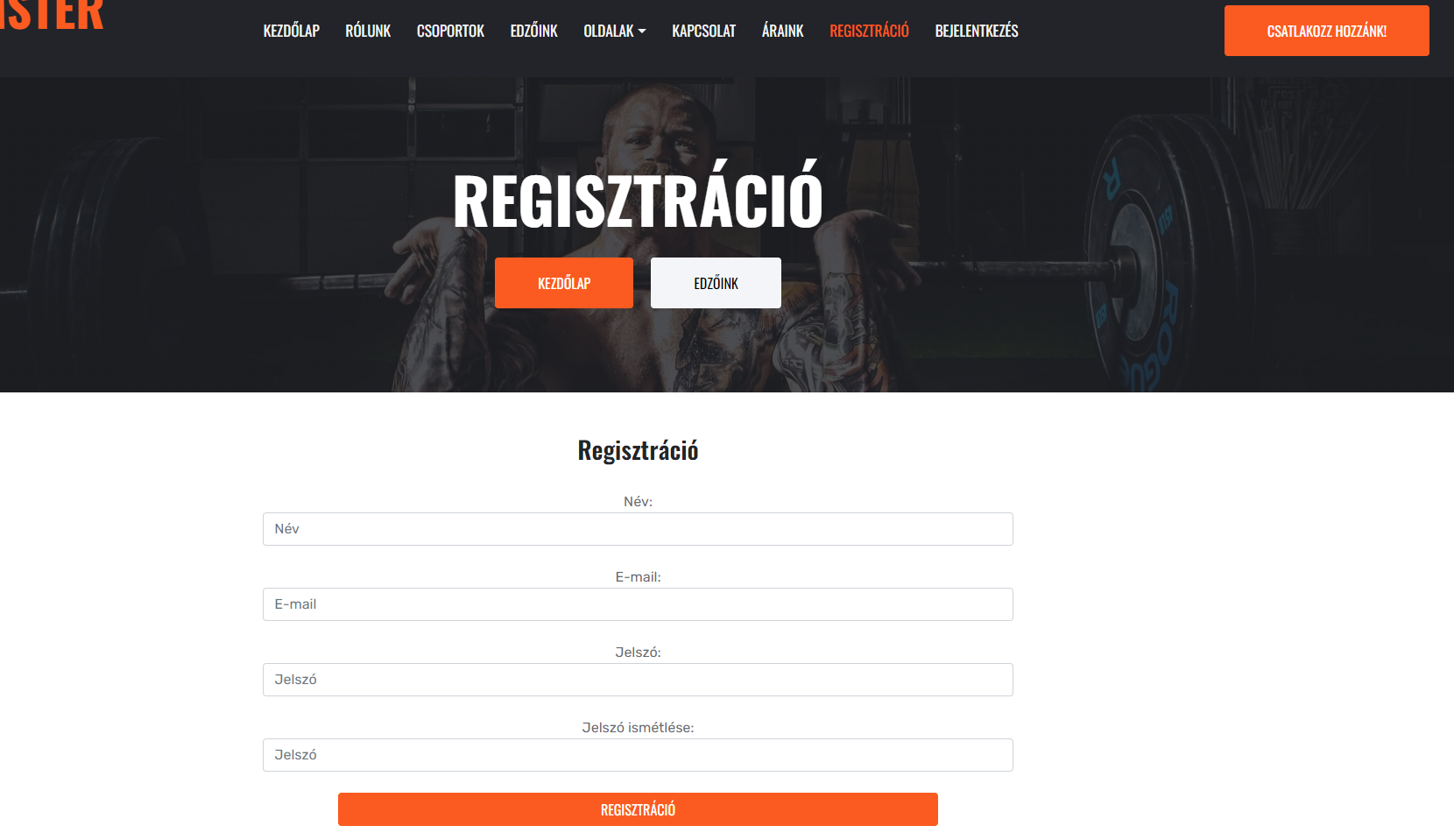
### Áraink menüpont

Ezen a menüponton találjuk az összes bérletet az árral együtt, és a kedvezményeket is, amik jelenleg az edzőteremnél elérhetőek.



### Regisztráció menüpont

A regisztráció oldalon tud a vendég regisztrálni, mellyel a felhasználói jogok elérhetővé válnak számára.



A regisztrációhoz hat adatot kell kötelezően megadni:

* Név
* Email
* Jelszó
* Jelszó ismétlése

Az utóbbira a jelszó megerősítése miatt van szükség. A regisztrációnál az alábbi

hibalehetőségek vannak:

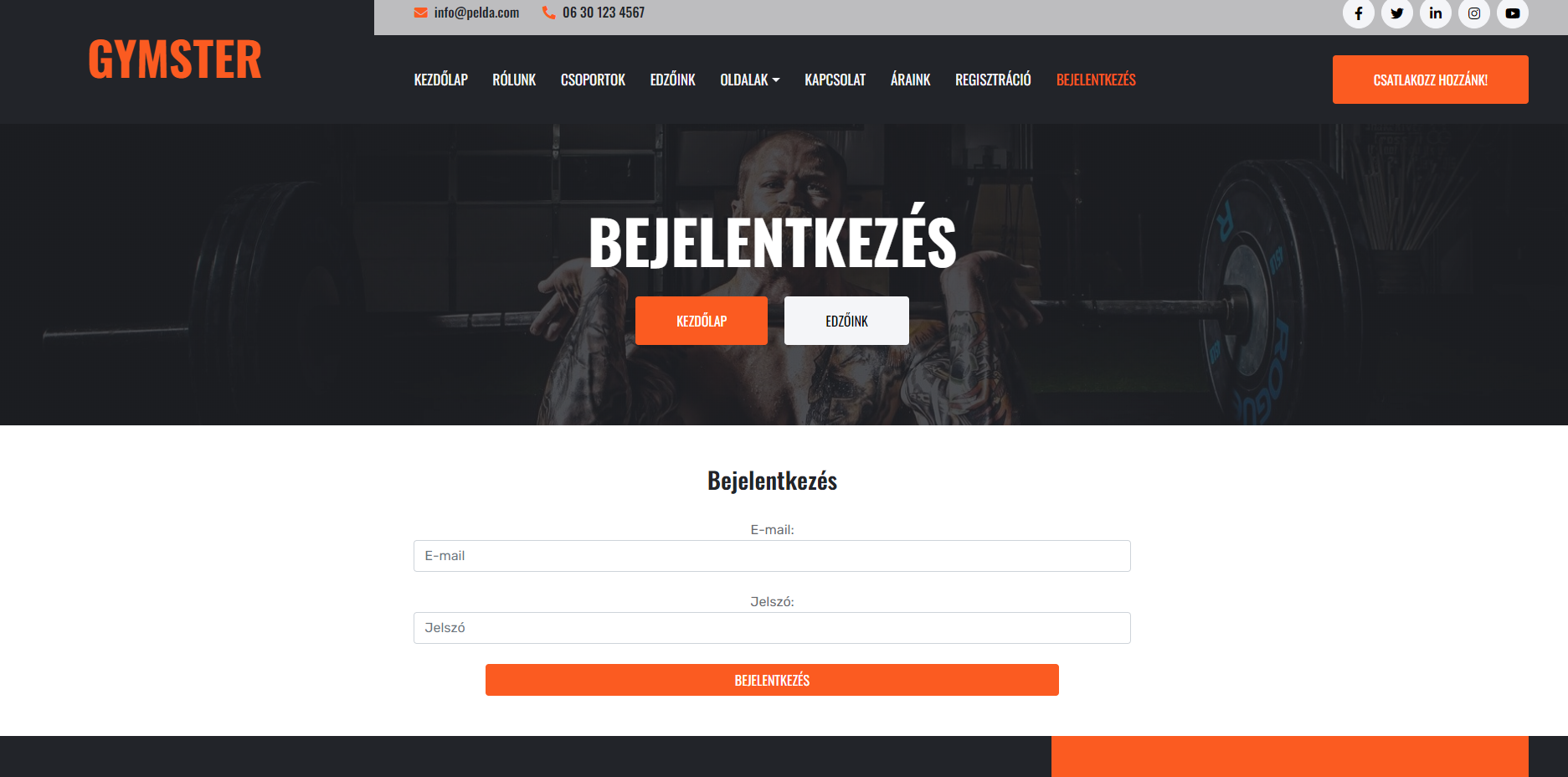
* Mindegyik mezőt ki kell tölteni
* A két jelszó nem egyezik meg
* Ez az E-mail cím már létezik

Ha sikeresen kitöltöttük az adatokat, akkor regisztrálhatunk is az oldalra, ami ezek után visszadob minket a főoldalra, ahonnan a bejelentkezés menüre kattintva beléphetünk.

### Bejelentkezés menüpont

A belépés menüpontot, és a regisztráció menüpontot csak vendégként bírjuk elérni.

Itt tud a már regisztrált felhasználó belépni a fiókjába.



A belépéshez kettő adatot kell megadni, amik a következők:

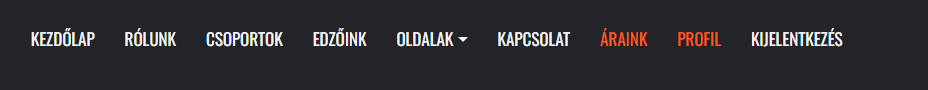
* Email
* Jelszó

Hibalehetőségek:

* Hibás email
* Hibás jelszó
* Ez a felhasználó törölve lett

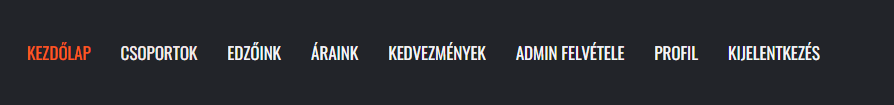
#### Felhasználóknál:

Felhasználói bejelentkezés után a bejelentkezett személy profiljára megy, ahol további menüpontok jelennek meg.



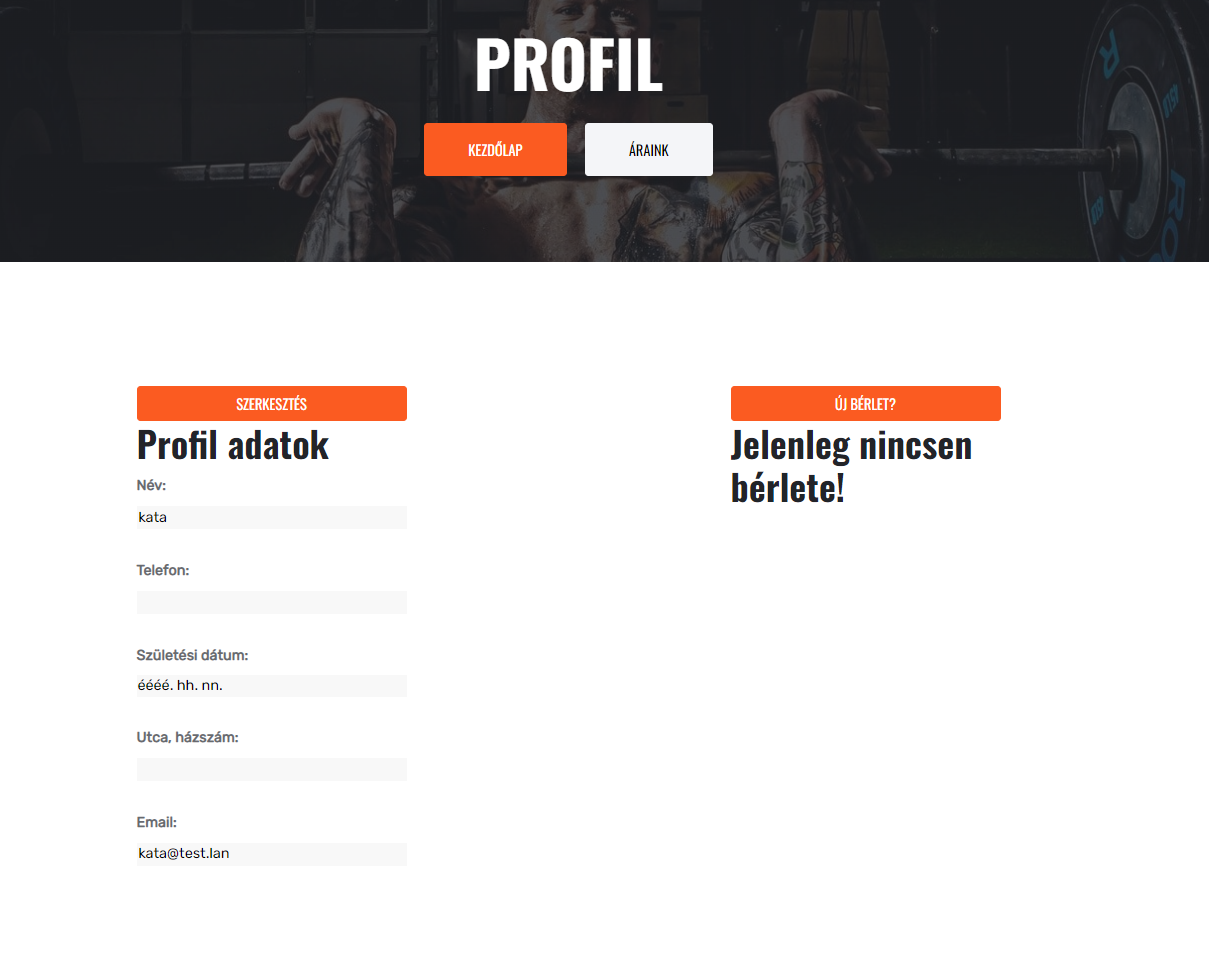
#### Adminisztrátoroknál:

Adminisztrátori bejelentkezés után a személy profiljára megy, ahol további menüpontok jelennek meg.



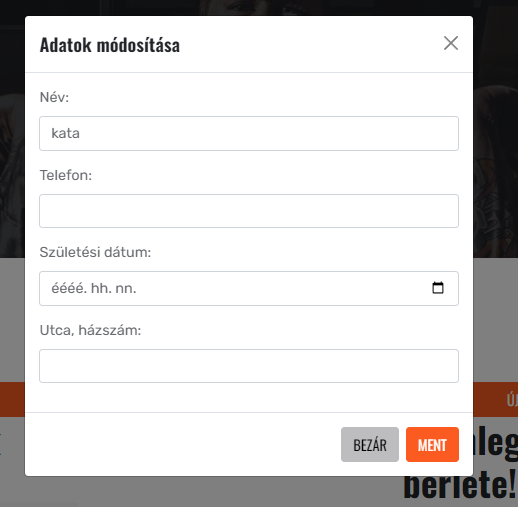
### Profil menüpont

A profil menüponton láthatjuk a felhasználónak az személyes adatait.



Kettő funkció közül lehet választani, Profil szerkesztése, és új bérlet vétele, ha még nincs bérletünk.

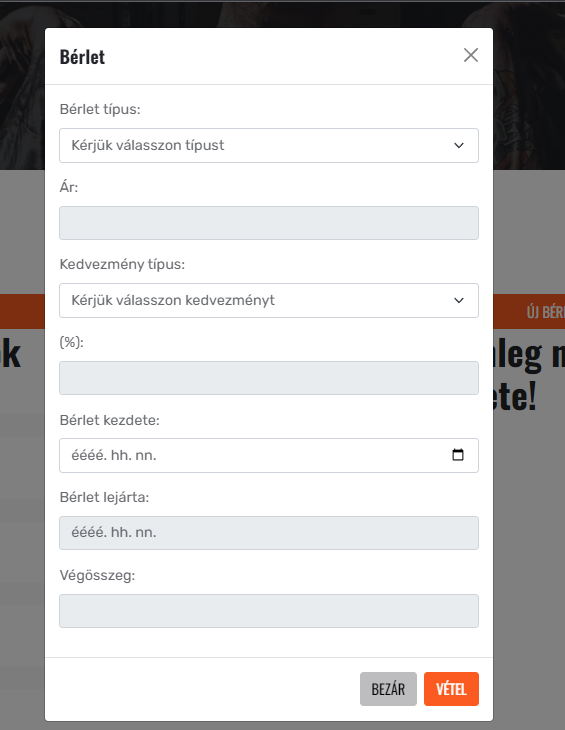
#### Profil szerkesztése



Itt személyre szabhatjuk személyes adatainkat, kivéve az Email címet, majd a ’Ment’ gombra el is menti az adatbázisba az adatainkat.

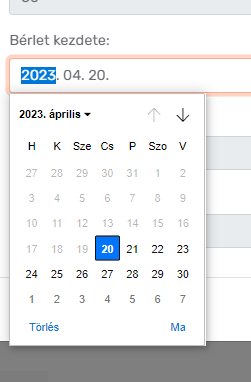
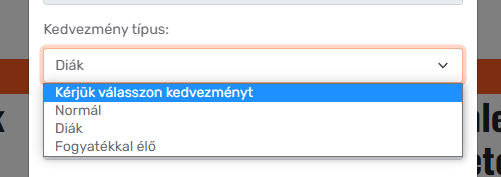
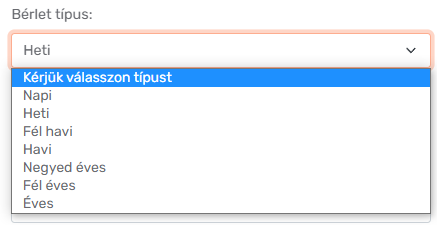
#### Új bérlet

’Új bérlet?’ gombra kattintva (ha még nincs bérletünk) felugrik egy bekérő ablak, ahol igényünk szerint alakíthatjuk a bérletet.

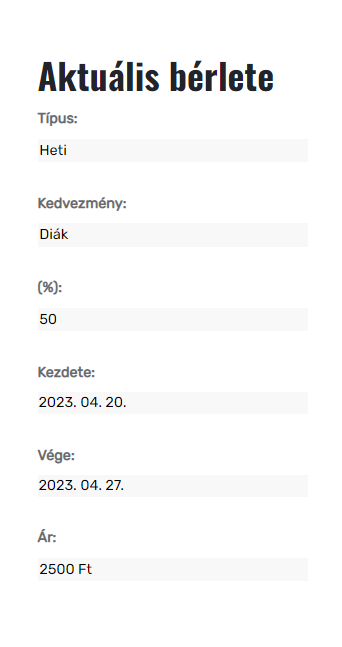


Itt kiválaszthatjuk a kívánt bérlettípust a lehetőségek közül, amit korábban az ’Áraink’ menüpont alatt is megnézhettünk.

Kiválaszthatjuk a bérlet és a kedvezmény típusát, melyből a rendszer automatikusan kiszámolja az árat. Bérlet kezdetét is megadhatjuk, ami nem kezdődhet korábban a jelenlegi napnál.

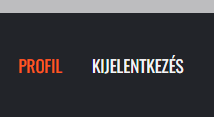


Majd a ’Vétel’ megnyomásával beíródik a bérletünk, amit utólagosan már nem lehet szerkeszteni, csak ha lejárt az adott bérletünk dátuma.



### Kijelentkezés menüpont

Kijelentkezés gomb megnyomásakor a felhasználót kijelentkezteti, és a bejelentkezési oldalra dobja.

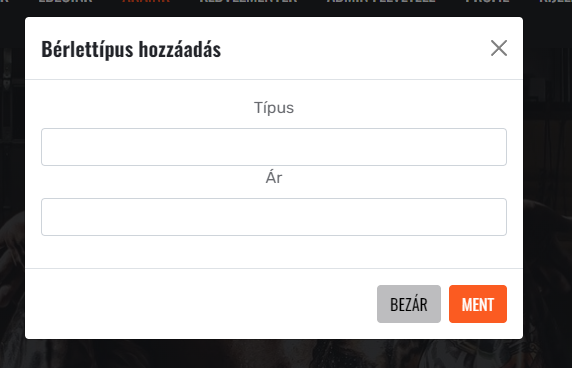


### Ami az adminisztrátornál más:

#### Áraink menüpont

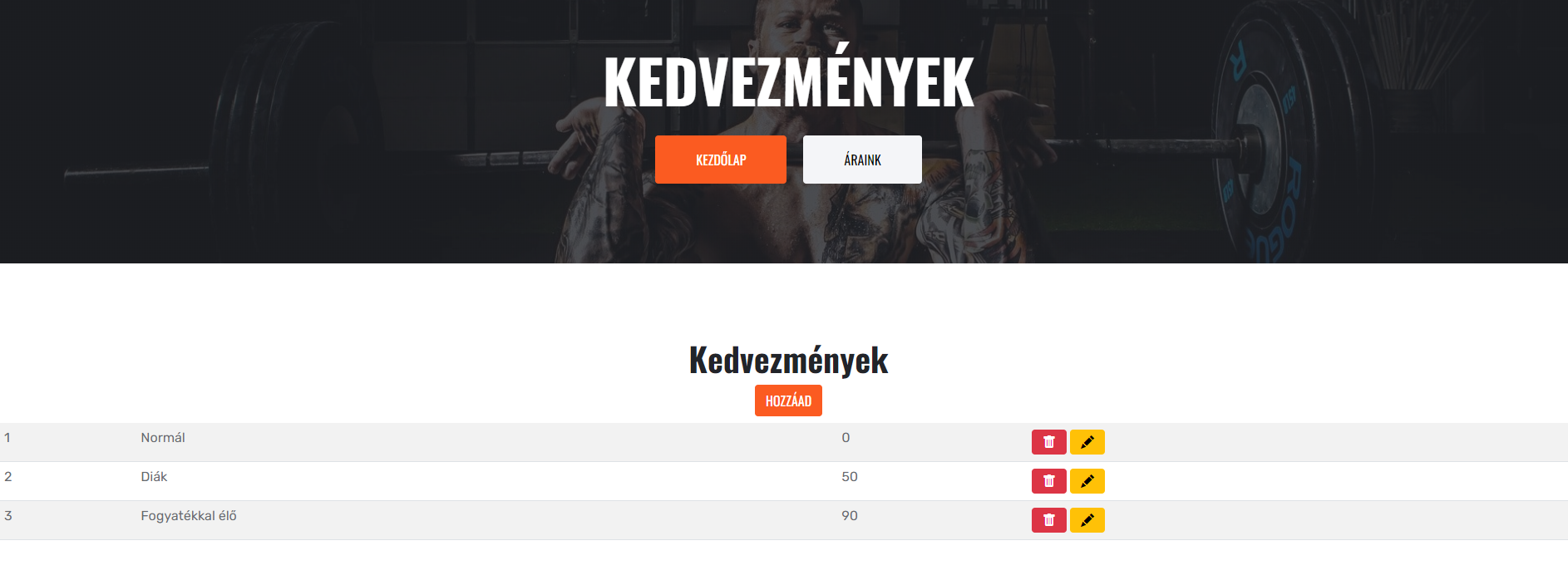
Itt az admin, illetve a későbbiekben létrehozott ’moderátorok’ tudnak bérlettípust törölni (a szemetes ikonnal), szerkeszteni (ceruza ikonnal), és létrehozni is a „Hozzáad” gombbal.

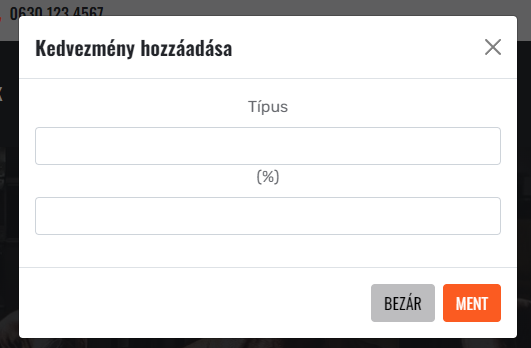




#### Kedvezmények

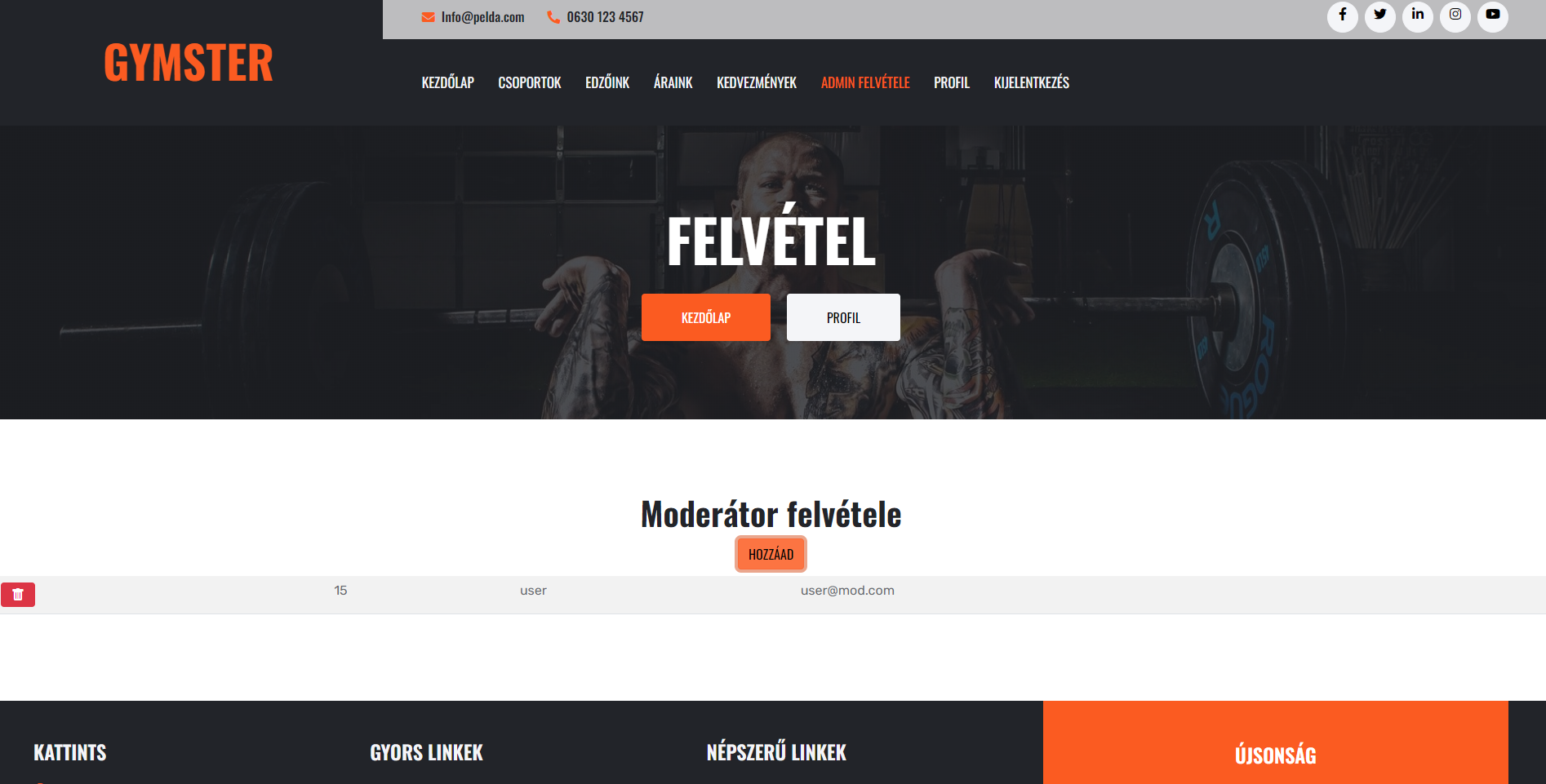
Itt az adminok szerkeszteni tudják az adott kedvezményeket. Tudnak törölni (a szemetes ikonnal), szerkeszteni (ceruza ikonnal), és létrehozni is a „Hozzáad” gombbal.



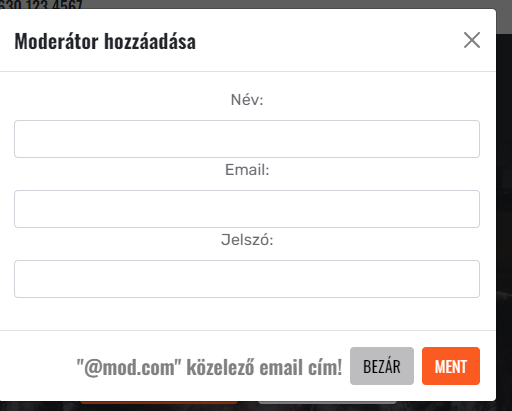


#### Admin felvétele

Itt az admin fel tud venni plusz admint, (aki moderátor szerepet kap, hogy ne tudja törölni a fő admint a rendszerből.) és későbbiekben törölni is tudja, ha úgy adódik.



Új moderátort csakis a fő admin tud hozzáadni, hozzáadott felhasználóknak ez a funkció nem elérhető.



# Összefoglalás

Ez a projekt egyfajta prototípus, amely már használható, de még korántsem kész. A projekt eredeti tervében szerepelt, hogy a felhasználók profiljához városokat és megyéket lehessen hozzárendelni, (ahogy a bérletekhez is lehet a lenyíló választási lehetőségeknél), de sajnos ez a tervezett funkció nem valósult meg. Azonban, úgy döntöttünk, hogy nem szeretnénk eltávolítani a projektből, és reméljük, hogy később sikerül bevezetni ezt a funkciót.

A bérletek okozták a legnagyobb kihívást a projekt során, különösen az aktuális dátum kiszámolása és kiíratása, emellett a bérlettípushoz való (bérlet vége) számolása volt nagyon nehéz feladat.

Emellett a kedvezményekkel való számítás és kiíratás is időigényes volt a bérlettől függően. Azonban sikerült megoldást találni ezekre a problémákra, és a projekt továbbfejlesztésekor ezekre is odafigyelünk majd.

A jövőbeli terveink között szerepel a projekt további fejlesztése és a weboldalnak az arculat bővítése. További funkciók között szerepel a létszám számláló, amely lehetővé teszi, hogy a felhasználók nyomon követhessék, hogy hányan vannak az edzőteremben egy adott időpontban, valamint a heti átlagos látogatottságot is láthassák.

Az egész projekt során rengeteget tanultunk, különösen a lekérdezések felépítése és logikája terén. A projekt során szerzett tapasztalatok segítenek majd a jövőbeli projektekben is, és reméljük, hogy a fejlesztéseknek köszönhetően a projekt még hasznosabbá válik a felhasználók számára.