**Horgászhelyfoglaló**

**weblap**

Fejlesztők és konzulens személyek:

Fejlesztők

* Vitovszki Tamás:

frontend programozás, tesztelés, web design készítés

* Szekeres Miklós:

backend programozás, tesztelés

Konzulens személyek

* Kaczur Sándor:

Segítség a program logikai felépítésben,   
optimalizációban, hiba mentesitésben  
és a tiszta kód elveinek megfelelő használatában

[Fejlesztők és konzulens személyek: 2](#_Toc163746428)

[Fejlesztők 2](#_Toc163746429)

[Konzulens személyek 2](#_Toc163746430)

[Projekt célja: 5](#_Toc163746431)

[Projekt Célja: 5](#_Toc163746432)

[Miért lett létrehozva, mi a lényege: 5](#_Toc163746433)

[Miért van rá szükség: 5](#_Toc163746434)

[Miért használnák a felhasználók: 6](#_Toc163746435)

[Licence: 6](#_Toc163746436)

[Read-Only License: 6](#_Toc163746437)

[Felhasznált technologiak: 8](#_Toc163746438)

[Frontend (Angular) 8](#_Toc163746439)

[Backend (Laravel) 8](#_Toc163746440)

[Követelmények: 10](#_Toc163746441)

[Funkcionális követelmények: 10](#_Toc163746442)

[Megszorítások: 10](#_Toc163746443)

[Nem funkcionális: 10](#_Toc163746444)

[Tervek: 11](#_Toc163746445)

[Adatbázis terv: 11](#_Toc163746446)

[Látványterv: 12](#_Toc163746447)

[Program leírása: 13](#_Toc163746448)

[Backend: 13](#_Toc163746449)

[AuthController 13](#_Toc163746450)

[emailController 16](#_Toc163746451)

[FishingPlaceController 17](#_Toc163746452)

[ImageController 20](#_Toc163746453)

[ReservationController 22](#_Toc163746454)

[ResponseController 27](#_Toc163746455)

[UserController Leírás 28](#_Toc163746456)

[Modelek 31](#_Toc163746457)

[FishingPlace Model Leírás 31](#_Toc163746458)

[Reservation Model Leírás 32](#_Toc163746459)

[User Model Leírás 34](#_Toc163746460)

[image model Leirás 36](#_Toc163746461)

[Migrations 37](#_Toc163746462)

[Users\_Table: 37](#_Toc163746463)

[Fishing\_places\_table 39](#_Toc163746464)

[add\_additionaldata\_users\_table 41](#_Toc163746465)

[reservations\_table 42](#_Toc163746466)

[images\_table 44](#_Toc163746467)

[Seeders: 45](#_Toc163746468)

[Admin Felhasználó Seeder Leírás 45](#_Toc163746469)

[Horgászhely Táblázat Seeder Leírás 47](#_Toc163746470)

[Foglalás Táblázat Seeder Leírás 49](#_Toc163746471)

[Felhasználó Táblázat Seeder Leírás 50](#_Toc163746472)

[Frontend 52](#_Toc163746473)

[Home page: 52](#_Toc163746474)

[About us: 53](#_Toc163746475)

[Book main: 54](#_Toc163746476)

[Book form: 55](#_Toc163746477)

[Login: 56](#_Toc163746478)

[Sign up: 57](#_Toc163746479)

[Verify email: 58](#_Toc163746480)

[Profile 59](#_Toc163746481)

[Reservation: 60](#_Toc163746482)

[Gallery 61](#_Toc163746483)

[Admin 63](#_Toc163746484)

[1. Felhasználók kártya: 63](#_Toc163746485)

[2. Horgászhelyek kártya: 64](#_Toc163746486)

[3. Foglalások kártya: 65](#_Toc163746487)

[Ábrajegyzék: 68](#_Toc163746488)

[Képek: 68](#_Toc163746489)

[Felhasználói útmutató: 69](#_Toc163746490)

[Ismert problémák 70](#_Toc163746491)

[Jövőbeli tervek 71](#_Toc163746492)

[Felhasznált irodalom: 73](#_Toc163746493)

[Angular 73](#_Toc163746494)

[Laravel 73](#_Toc163746495)

[Köszönet: 74](#_Toc163746496)

Projekt célja:

Projekt Célja:

A weboldal célja a horgászhelyek online foglalásának egyszerűsítése és felhasználóink számára a kényelmes és gyors hozzáférést biztosítani a kiválasztott horgászhelyekhez.

Miért lett létrehozva, mi a lényege:

A horgászhely foglaló weboldalt azért hoztuk létre, hogy lehetőséget teremtsünk a horgászoknak a kívánt területek előzetes foglalására. A felhasználók számára ezáltal lehetőséget nyújtunk arra, hogy elkerüljék a helyszínen történő foglalás nehézségeit, és könnyedén megtervezhessék horgászélményüket.

Miért van rá szükség:

A horgászhely foglaló weboldal megkönnyíti a horgászok életét, lehetővé téve számukra, hogy előre lefoglalják a kívánt horgászhelyeket. Ezáltal időt és energiát takaríthatnak meg, és biztosíthatják, hogy a kiválasztott helyek elérhetők legyenek a kívánt időpontban.

Miért használnák a felhasználók:

* Kényelem és Rugalmasság: A weboldal lehetővé teszi a felhasználók számára, hogy bármikor és bárhol foglalhassanak, anélkül, hogy személyesen meg kellene jelenniük.
* Aktuális Információk: A felhasználók részletes információkat kapnak a horgászhelyekről, beleértve a szabad helyek számát, a horgászati szabályokat és egyebeket.
* Gyors és Egyszerű Foglalás: Az intuitív felület lehetővé teszi a gyors és egyszerű foglalást, minimalizálva a bürokráciát és időveszteséget.
* Értesítések és Ellenőrzések: A felhasználók ellenőrizhetik és szerkeszthetik foglalásaikat a weboldalon.
* Felhasználói Élmény: Az oldal célja, hogy kiváló felhasználói élményt biztosítson, növelve ezzel a weboldal használatának vonzerejét.

Licence:

Read-Only License:

A Szoftver forráskódját csak olvasható, és nem módosítható.

Nem jogosult a forráskódot terjeszteni vagy közzétenni engedély nélkül.

A Szoftvert csak a licensz tulajdonosa használhatja, és másoknak bármilyen formában történő átadása vagy használata tilos.

A Szoftver minden jogát fenntartjuk, és bármikor visszavonhatjuk ezt a licenszet.

Jelen licensz hatálybalépése a Szoftver letöltésével keletkezik és a Szoftverrel való bármilyen használatától számítottan él. A Szoftver használatával beleegyezel ezekhez a feltételekhez.

Felhasznált technologiak:

Frontend (Angular)

Keretrendszer:

* Angular

Programnyelv:

* TypeScript
* HTML
* CSS

Ingyenes / Fizetős:

* Angular ingyenesen használható, az open source licenszének köszönhetően.

Fejlesztési Módszer:

* Az Angular frontend része a komponens-alapú fejlesztési módszert követi, ahol a felhasználói interfész különböző komponensekből épül fel.

Backend (Laravel)

Keretrendszer:

* Laravel

Programnyelv:

* PHP

Ingyenes / Fizetős:

* Laravel ingyenesen használható, az open source licenszének köszönhetően.

Fejlesztési Módszer:

Laravel a PHP alapú MVC (Model-View-Controller) architektúrát követi. A fejlesztés során a Laravel keretrendszer a kódot modell, nézet és vezérlőként szervezi meg.

Követelmények:

Funkcionális követelmények:

* Felhasználó Létrehozása
* Felhasználó Törlése
* Helyek Lefoglalása
* Foglalás Módosítása
* Foglalás Törlése
* Jól Látható Szövegek
* Dinamikus és Gyors Működés:

Megszorítások:

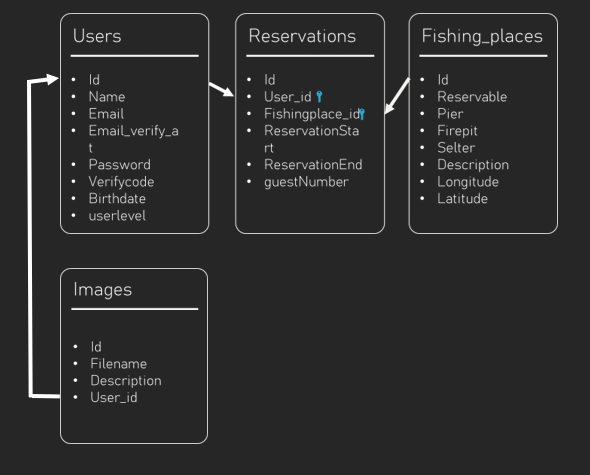
* A regisztrációnál kötelező megadni a felhasználónevet, e-mail címet, jelszót és további releváns adatokat (például teljes név, születési dátum).
* A jelszónak legalább 6 karakter hosszúnak kell lennie, valamint tartalmaznia kell kis- és nagybetűt, valamint speciális karaktert.
* Időpont foglaláshoz bejelentkezés kötelező.
* A galériába való feltöltéshez bejelentkezés kötelező.

Nem funkcionális:

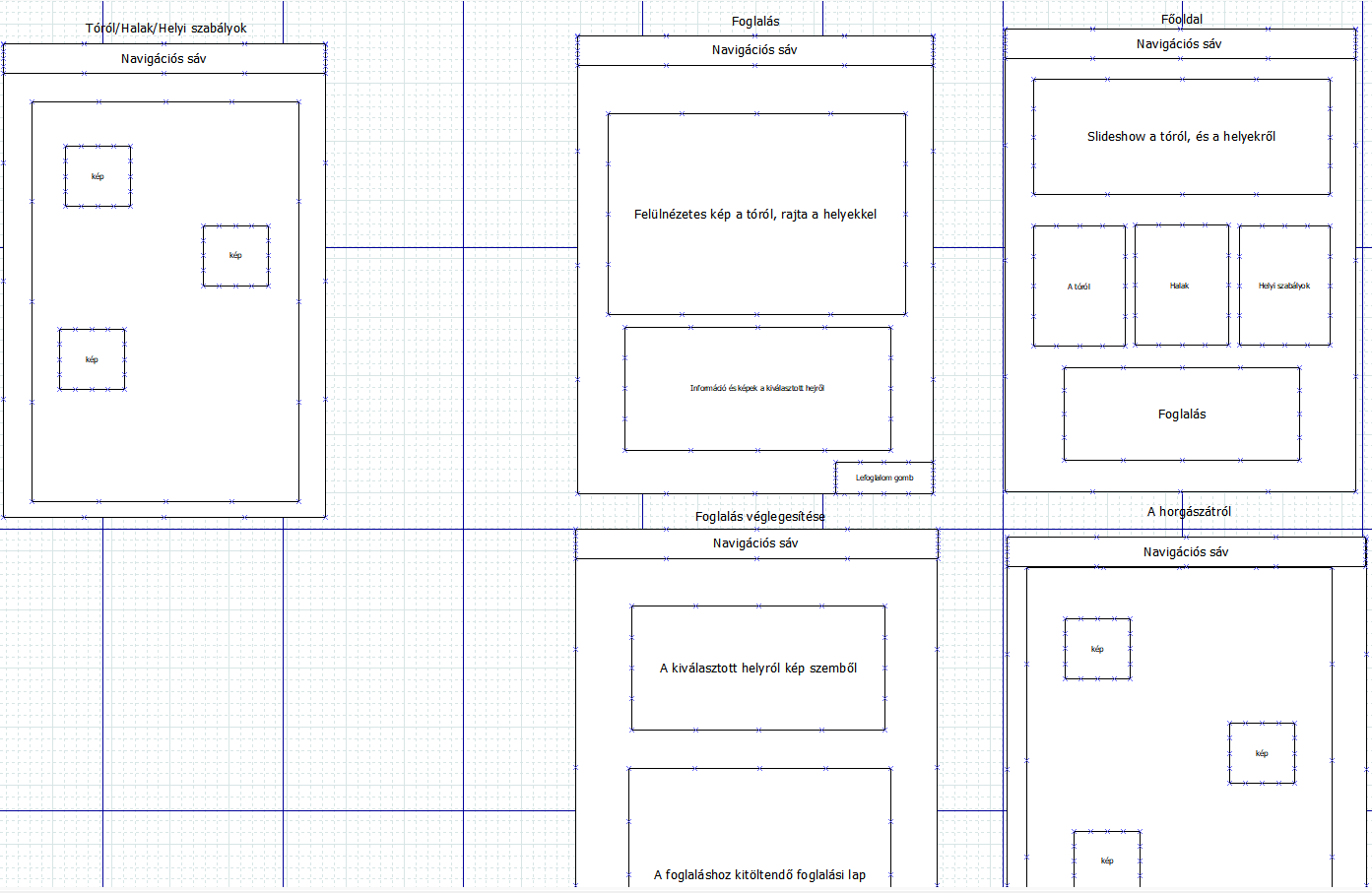
* Angular CLI: 17.2.2
* Node: 18.18.0
* Package Manager: npm 10.4.0
* Bootstrap 5
* Laravel backend
* MySql Adatbázis

Tervek:

Adatbázis terv:



Látványterv:



Program leírása:

Backend:

AuthController

Ez az osztály a Laravel keretrendszerben a felhasználói hitelesítéssel kapcsolatos műveleteket kezeli. A ResponseController osztályt örökli, amely általános válaszformátumokat biztosít az API-hívásokhoz.

**Metódusok:**

* **userRegister(UserRegisterChecker):**
  + Ez a metódus a felhasználó regisztrációját kezeli.
  + A UserRegisterChecker kérésvalidáló osztályt használja annak biztosítására, hogy a kérésben szereplő adatok a várt formátumban legyenek.
  + A validált adatokat lekéri a kérésből ($request->all()).
  + Titkosítja a jelszót a bcrypt függvénnyel (bcrypt($input["password"])).
  + Létrehoz egy véletlen e-mail-ellenőrző kódot az genEmailCode() metódus segítségével.
  + Elküldeni az e-mail-ellenőrző kódot a megadott e-mail címre az emailController osztály sendEmailCode() metódusával.
  + Elmenti a felhasználói adatokat az App\Models\User modell segítségével, a létrehozott ellenőrző kódot is beleértve.
  + Sikeres válaszüzenetet küld vissza a felhasználó nevével és a "sikeres regisztráció" szöveggel a sendResponse() metódus segítségével.
* **emailverify:**
  + Ez a metódus az e-mail-cím megerősítését kezeli.
  + A kérésből lekéri az 'verifycode' paraméter értékét, amely az e-mailben elküldött ellenőrző kód.
  + Lekérdezi a User-t ahol a ’verifycode’ megegyezik a Request-böl kapott koddal.
  + Ha a felhasználó létezik:
    - Frissíti a felhasználó email\_verified\_at tulajdonságát az aktuális dátumra.
    - Elmenti a módosításokat a felhasználón.
    - Sikeres válaszüzenetet küld vissza a dátummal és a "Visszaigazolás elküldve" szöveggel a sendResponse() metódus segítségével.
  + Ha a felhasználó nem létezik:
    - Hiba választ küld vissza a "Hibás visszaigazolokod" szöveggel a sendError() metódus segítségével.
* **userLogin(UserLoginChecker):**
  + Ez a metódus a felhasználó bejelentkezését kezeli.
  + A UserLoginChecker kérésvalidáló osztályt használja annak biztosítására, hogy a kérésben szereplő adatok a várt formátumban legyenek.
  + A Laravel Auth osztály attempt() metódusát használja a bejelentkezési adatok (e-mail cím és jelszó) érvényességének ellenőrzésére.
  + Sikeres bejelentkezés esetén:
    - Ellenőrzi, hogy a felhasználó e-mail címe megerősítve van-e (email\_verified\_at nem üres).
    - Ha az e-mail cím megerősítve van:
      * Létrehoz egy új tokent a felhasználó számára a createToken() metódus segítségével.
      * Sikeres válaszüzenetet küld vissza a tokennel, a felhasználó nevével és azonosítójával a sendResponse() metódus segítségével.
    - Ha az e-mail cím nincs megerősítve:
      * Hibap választ küld vissza a "Bejelentkezési hiba, felhasználói fiók még nincs visszaigazolva" szöveggel a sendError() metódus segítségével.
  + Sikertelen bejelentkezés esetén:
    - Hibap választ küld vissza az "Adatbeviteli hiba, hibás e-mail cím vagy jelszó" szöveggel a sendError() metódus segítségével.
* **userLogout(Request):**
  + Ez a metódus a felhasználó kijelentkezését kezeli.

emailController

Ez az osztály a Laravel keretrendszerben az e-mail küldéssel kapcsolatos műveleteket kezeli. A Controller osztályt örökli.

**Metódus:**

* **sendEmailCode:**
  + Ez a metódus elküld egy e-mailt a megadott e-mail címre a megadott visszaigazolási kóddal.
  + Létrehoz egy $content tömböt, amely a levél tartalmát tartalmazza. A tartalom két elemből áll:
    - title: Az e-mail címe ("visszaigazolási kód").
    - code: A visszaigazolási kód értéke.
  + A Laravel Mail osztály to() metódusát használja a címzett e-mail címének megadására.
  + Létrehoz egy új Email példányt a $content tömbbel, amely meghatározza az e-mail tartalmát.
  + A Mail osztály send() metódusát hívja meg az e-mail elküldésére az Email példány használatával.

FishingPlaceController

Ez az osztály a horgászhelyekkel kapcsolatos műveleteket kezeli a Laravel keretrendszerben. A ResponseController osztályt örökli, amely általános válaszformátumokat biztosít az API-hívásokhoz.

**Metódusok:**

* **getFishingPlaces:**
  + Ez a metódus l-ekéri az összes horgászhelyet az adatbázisból az App\Models\FishingPlace modell segítségével.
  + A lekérdezett horgászhelyeket tartalmazó tömböt adja vissza sikeres válaszüzzenettel ("Horgászhelyek betöltve") a sendResponse() metódus segítségével.
* **getFishingPlace:**
  + Ez a metódus lekér egy adott horgászhelyet az azonosítója alapján.
  + A kérésből ("id" paraméter) kiolvassa a keresett horgászhely azonosítóját.
  + Az App\Models\FishingPlace modell where() metódusát használja az azonosító alapján történő szűréshez.
  + Az first() metódus segítségével az első találatot adja vissza.
  + Ha nem található horgászhely:
    - Hiba választ küld vissza a "A keresett horgászhely nem található" szöveggel a sendError() metódus segítségével.
  + Sikeres esetben:
    - A megtalált horgászhely adatait adja vissza sikeres válaszüzzenettel ("Horgászhely betöltve") a sendResponse() metódus segítségével.
* **addFishingPlace:**

Ez a metódus új horgászhelyet ad hozzá az adatbázishoz.

* + A FishingPlaceChecker kérésvalidáló osztályt használja annak biztosítására, hogy a kérésben szereplő adatok a várt formátumban legyenek.
  + A validált adatokat lekéri a kérésből.
  + Létrehoz egy új FishingPlace példányt.
  + Beállítja a horgászhely tulajdonságait a kérésből származó adatok alapján (móló, tűzrakhely, menedék, leírás, hosszúságifok, szélességifok).
  + Elmenti az új horgászhelyet az adatbázisba a save() metódus segítségével.
  + Sikeres válaszüzenetet küld vissza a hozzáadott horgászhely adataival és a "Horgászhely hozzáadva" szöveggel a sendResponse() metódus segítségével.
* **modifyFishingPlace:**

Ez a metódus módosít egy meglévő horgászhelyet az adatbázisban.

* + A FishingPlaceChecker kérésvalidáló osztályt használja annak biztosítására, hogy a kérésben szereplő adatok a várt formátumban legyenek.
  + A validált adatokat lekéri a kérésből.
  + Lekéri a módosítani kívánt horgászhely azonosítóját a kérésből.
  + Az App\Models\FishingPlace modell where() metódusát használja az azonosító alapján történő szűréshez és az első találat lekéréséhez (first()).
  + Ha nem található horgászhely:
    - Hibap választ küld vissza a "A keresett horgászhely nem található" szöveggel a sendError() metódus segítségével.
  + Sikeres esetben:
    - Frissíti a horgászhely tulajdonságait a kérésből származó adatok alapján (foglalható, móló, tűzrakhely, menedék, leírás, hosszúság, szélesség).
    - Elmenti a módosításokat az adatbázisba a save() metódus segítségével.
    - Sikeres válaszüzenetet küld vissza a módosított horgászhely adataival és a "Horgászhely módosítva" szöveggel

ImageController

Ez az osztály a képekkel kapcsolatos műveleteket kezeli a Laravel keretrendszerben. A Controller osztályt örökli.

**Metódusok:**

* **index:**
  + Ez a metódus lekéri az összes képet az adatbázisból az App\Models\Image modell segítségével.
  + A képeket időrendi sorrendben, a legújabbitól a legrégebbi felé rendezi a latest() metódus használatával.
  + Lekéri a képek adatait a get() metódus segítségével.
  + Meghatározza a képek alap URL-jét a public\_path('images') kifejezéssel, amely a nyilvános "images" mappa elérési útját adja meg.
  + A transform() metódussal módosítja a lekérdezett képeket:
    - A $image változó az aktuális képet tartalmazza a ciklus során.
    - A $baseUrl változót használja az alap URL-cím előtagként.
    - Beállítja a kép URL-jét az alap URL és a kép fájlneve összefűzésével ($baseUrl . '/' . $image->filename).
    - Visszatéríti a módosított képadatokat.
  + A módosított képek tömbjét adja vissza.
* **upload:**
  + Ez a metódus új képet tölt fel a szerverre.
  + Érvényesíti a kérésben szereplő adatokat a validate() metódus segítségével:
    - A 'kép' mező kötelező, kép típusú kell legyen (JPEG, PNG, JPG, GIF), a maximális mérete 10240 KB.
    - A 'leírás' mező kötelező, szöveges típusú, maximum 255 karakter hosszú.
    - A 'user\_id' mező kötelező, a feltöltő felhasználó azonosítóját tartalmazza.
  + Létrehoz egy egyedi fájlnevet az aktuális időpont (másodpercek óta eltelt idő) és a kiterjesztés összefűzésével (time() . '.' . $request->image->extension()).
  + Elmenti a feltöltött képet a megadott fájlnévvel a nyilvános "images" mappába a move() metódus segítségével.
  + Létrehoz egy új kép rekordot az adatbázisban az Image modell create() metódusával:
    - A 'filename' mező a kép fájlnevét tartalmazza.
    - A 'description' mező a kép leírását tartalmazza.
    - A 'user\_id' mező a feltöltő felhasználó azonosítóját tartalmazza.
  + Sikeres válaszüzenetet ad vissza JSON formátumban ("a kép sikeresen feltöltve."), 201-es HTTP státuszkóddal.
* **delete:**
  + Ez a metódus töröl egy képet az adatbázisból és a fájlrendszerből az azonosítója alapján.
  + Az Image modell find($id) metódusával megkeresi a törölni kívánt képet az azonosítója alapján.
  + Ha nem található a kép ("is\_null($image)"):
    - Hibap választ küld vissza JSON formátumban ("Nincs ilyen kép"), 404-es HTTP státuszkóddal.
  + Ha megtalálható a kép:
    - Létrehozza a kép teljes elérési útját a nyilvános "images" mappához és a fájlnévhez fűzve (public\_path('images') . '/' . $image->filename).
    - Ha létezik a kép a fájlrendszerben (file\_exists($imagePath)):
      * Törli a képet a fájlrendszerből a unlink() függvény segítségével.
    - Törli a kép rekordot az adatbázisból a delete() metódus segítségével.
  + Sikeres válaszüzenetet ad vissza JSON formátumban ("Kép törölve"), 200-as HTTP státuszkóddal.

ReservationController

Ez az osztály a horgászhelyek foglalásait kezeli a Laravel keretrendszerben. A ResponseController osztályt örökli, amely általános válaszformátumokat biztosít az API-hívásokhoz.

**Felhasználói jogosultság:**

A metódusok többsége a sanctum jogosultsági megállapítást használja a felhasználó érvényesítéséhez a kérés elején (auth("sanctum")->user()). Ez arra utal, hogy a foglalások kezeléséhez a felhasználónak be kell jelentkeznie.

**Metódusok:**

* **getReservations:**
  + Lekéri az összes foglalást az adatbázisból a Reservation modell segítségével.
  + A felhasználót a metódus elején hitelesíti.
  + A lekérdezett foglalásokat tartalmazó tömböt ad vissza sikeres válaszüzzenettel ("Összes foglalás") a sendResponse() metódus segítségével.
* **getReservation:**
  + Lekér egy adott foglalást az azonosítója alapján.
  + A felhasználót a metódus elején hitelesíti.
  + A kérésből ("id" paraméter) kiolvassa a keresett foglalás azonosítóját.
  + Az App\Models\Reservation modell where() metódusát használja az azonosító alapján történő szűréshez és az első találat lekéréséhez (first()).
  + Ha nem található foglalás ("is\_null($reservation)"):
    - Hiba választ küld vissza a "A keresett foglalás nem található" szöveggel a sendError() metódus segítségével.
  + Sikeres esetben:
    - A megtalált foglalás adatait adja vissza sikeres válaszüzzenettel ("Egy foglalás") a sendResponse() metódus segítségével.
* **getUserReservations:**
  + Lekéri egy felhasználó összes foglalását.
  + A felhasználót a metódus elején hitelesíti.
  + A kérésből ("user\_id" paraméter) kiolvassa a felhasználó azonosítóját.
  + Az App\Models\Reservation modell where() metódusát használja a felhasználó azonosítója alapján történő szűréshez.
  + Ha nem található a felhasználónak foglalása ("is\_null($reservations)"):
    - Hiba választ küld vissza a "A felhasználónak nincs foglalása" szöveggel a sendError() metódus segítségével.
  + Sikeres esetben:
    - A felhasználó foglalásait tartalmazó tömböt adja vissza sikeres válaszüzzenettel ("A felhasználó foglalásai") a sendResponse() metódus segítségével.
* **addReservation(ReservationChecker $request):**
  + Új foglalást hoz létre a rendszerben.
  + A felhasználót a metódus elején hitelesíti.
  + A ReservationChecker kérésvalidáló osztályt használja annak biztosítására, hogy a kérésben szereplő adatok a várt formátumban legyenek.
  + A validált adatokat lekéri a kérésből ($request->all()).
  + A foglalás dátumait ellenőrzi a testDate() metódus segítségével.
  + Ha a dátumok foglalhatók:
    - Létrehoz egy új Reservation példányt.
    - Beállítja a foglalás tulajdonságait a kérésből származó adatok alapján (felhasználó ID, horgászhely ID, foglalás kezdete, foglalás vége, vendégek száma).
    - Elmenti az új foglalást az adatbázisba a save() metódus segítségével.
    - Sikeres válaszüzenetet küld vissza a hozzáadott foglalás adataival és a "Foglalás hozzáadva" szöveggel a sendResponse() metódus segítségével.
* **modifyReservation(ReservationChecker $request):**
  + Módosít egy meglévő foglalást.
  + A felhasználót a metódus elején hitelesíti.
  + A `Reservation

**modifyUserReservation(ReservationChecker $request):**

* **Felhasználói hitelesítés:** Más metódusokhoz hasonlóan a auth("sanctum")->user() hívásával kezdődik a felhasználó hitelesítő adatainak ellenőrzése.
* **Kérés validálása és adatszerzés:**
  + A $request->validated() hívásával ellenőrzi, hogy a kérés adatai megfelelnek-e a ReservationChecker osztályban definiált validációs szabályoknak.
  + A $request->all() használatával lekéri az összes érvényesített adatot a kérésből, és eltárolja a $input változóban.
* **Foglalási azonosító és felhasználói azonosító kinyerése:**
  + A kérés adataiból ($input) kinyeri a foglalási azonosítót ($id) és a felhasználói azonosítót ($user\_id).
* **Foglalás keresése:**
  + Lekéri az összes, a megadott $user\_id-hez rendelt foglalást a Reservation::where("user\_id",$user\_id)->get() használatával. Ez szükségtelen lehet, mivel a foglalást később azonosító ($id) alapján is lekéri.
* **Konkrét foglalás keresése:**
  + Ezután azonosító alapján megkeresi a konkrét foglalást a Reservation::where("id",$id)->first() használatával.
* **Ha a foglalás nem található:**
  + Ha a foglalás nem található (is\_null($reservation)), a sendError() használatával hibaválaszt küld a "A keresett foglalás nem található" üzenettel.
* **Ha a foglalás megtalálható):**
  + Ha a foglalást megtalálja, a $input-ban megadott adatok alapján frissíti a részleteit:
    - user\_id: Frissíti a foglalás felhasználói azonosítóját (bár meg kell egyeznie a megadott $user\_id-vel).
    - fishingplace\_id: Frissíti a foglalás horgászhely azonosítóját.
    - reservationStart: Frissíti a foglalás kezdési dátumát.
    - reservationEnd: Frissíti a foglalás befejezési dátumát.
    - guestNumber: Frissíti a foglalás vendégszámát.
* **Változások mentése és válasz küldése:**
  + A $reservation->save() használatával menti az adatbázisba a frissített foglalási adatokat.
  + A sendResponse() használatával sikeres választ küld a módosított foglalási adatokkal és a "Foglalás módosítva" üzenettel.

**deleteReservation:**

Ez a metódus lehetővé teszi a felhasználó számára, hogy töröljön egy foglalást a foglalási azonosító alapján.

* **Felhasználói hitelesítés:** Más metódusokhoz hasonlóan a auth("sanctum")->user() hívásával kezdődik a felhasználó hitelesítő adatainak ellenőrzése.
* **Foglalási azonosító kinyerése:**
  + A kérés adataiból ($input) kinyeri a foglalási azonosítót ($id) a $request->all() használatával.
* **Foglalás keresése:**
  + Megpróbálja azonosító alapján megtalálni a foglalást a Reservation::find($id) használatával.
* **Ha a foglalás nem található):**
  + Ha a foglalás nem található a sendError() használatával hibaválaszt küld a "A keresett foglalás nem található" üzenettel.
* **ha a foglalás megtalálható):**
  + akor az adott id val rendelkezö foglalás a törlésre kerül.

ResponseController

* **Leírás:** A ResponseController osztály szabványos válaszokat biztosít az API kérésekhez.
* **Funkció:** Ez a kontroller két metódust kínál a válaszok konzisztens formátumban történő visszaküldéséhez: sendResponse és sendError.

**Metódusok:**

* sendResponse($data, $message)
  + **HTTP metódus:** Bármely HTTP metódus (a tényleges metódust az útvonal határozza meg)
  + **Leírás:** Sikeres válasz küldése adatokkal és üzenettel.
  + **Paraméterek:**
    - $data (tömb): A válaszként küldendő adatok.
    - $message (string): A válaszhoz tartozó üzenet.
  + **Visszatérési érték:** JSON válasz 200-as HTTP státuszkóddal. A válasz a következő formátumú tömböt tartalmazza:
    - success (bool): Igaz, ha a kérés sikeres volt.
    - data (tömb): A kért adatok.
    - message (string): A válasz üzenete.
* sendError($error)
  + **HTTP metódus:** Bármely HTTP metódus (a tényleges metódust az útvonal határozza meg)
  + **Leírás:** Hibaüzenet küldése.
  + **Paraméterek:**
    - $error (string): A hibaüzenet.
  + **Visszatérési érték:** JSON válasz 404-es HTTP státuszkóddal. A válasz a következő formátumú tömböt tartalmazza:
    - success (bool): Hamis, ha a kérés sikertelen volt.
    - message (string): A hibaüzenet.

UserController Leírás

**Leírás:**

A UserController osztály a felhasználókkal kapcsolatos API kéréseket kezeli a Laravel projektben. Öröklését a ResponseController osztályból, amely szabványos válaszokat biztosít az API kérésekhez.

**Használt Modell:**

* User: A felhasználói adatokat tároló modell.

**Metódusok:**

* getUsers()
  + **HTTP metódus:** GET
  + **Leírás:** Lekéri az összes felhasználót az adatbázisból.
  + **Paraméterek:** Nincs
  + **Visszatérési érték:** JSON válasz az összes felhasználó adataival a sendResponse metódus használatával. A válasz sikeres (success: true) és az üzenet "Összes felhasználó betöltve".
* getUser($request)
  + **HTTP metódus:** GET
  + **Leírás:** Lekéri a megadott azonosítójú felhasználót az adatbázisból.
  + **Paraméterek:**
    - $request (Request): A kérelem objektum.
    - id (integer a request body-ban): A felhasználó azonosítója.
  + **Visszatérési érték:**
    - Ha a felhasználó létezik: JSON válasz a felhasználó adataival a sendResponse metódus használatával. A válasz sikeres (success: true) és az üzenet "Egy felhasználó betöltve".
    - Ha a felhasználó nem létezik: JSON válasz hibaüzenettel a sendError metódus használatával. A válasz sikertelen (success: false) és az üzenet "A keresett felhasználó nem található".
* modifyUser($request)
  + **HTTP metódus:** PUT
  + **Leírás:** Módosítja a megadott azonosítójú felhasználó adatait.
  + **Paraméterek:**
    - $request (UserChecker Request): A kérelem objektum. A kérelemnek meg kell felelnie a UserChecker validációs szabályoknak.
  + **Visszatérési érték:**
    - Sikeres módosítás esetén: JSON válasz a módosított felhasználó adataival a sendResponse metódus használatával. A válasz sikeres (success: true) és az üzenet "Felhasználó adatai módosítva".
    - Hiba esetén (pl. felhasználó nem létezik): JSON válasz hibaüzenettel a sendError metódus használatával. A válasz sikertelen (success: false) és a megfelelő hibaüzenet.
* deleteUser($request)
  + **HTTP metódus:** DELETE (feltételezve)
  + **Útvonal:** /users/{id} (feltételezve)
  + **Leírás:** Törli a megadott azonosítójú felhasználót az adatbázisból.
  + **Paraméterek:**
    - $request (Request): A kérelem objektum.
    - id (integer a request body-ban): A felhasználó azonosítója.
  + **Visszatérési érték:**
    - Sikeres törlés esetén: JSON válasz a törölt felhasználó adataival (üres tömb) a sendResponse metódus használatával. A válasz sikeres (success: true) és az üzenet "Felhasználó törölve".
    - Hiba esetén (pl. felhasználó nem létezik): JSON válasz hibaüzenettel a sendError metódus használatával. A válasz sikertelen (success: false) és a megfelelő hibaüzenet.

**Megjegyzések:**

* A kód a Sanctum hitelesítést használja a felhasználók jogosultságainak ellenőrzésére.
* A UserChecker request osztályt a bemeneti adatok validálására használják a modifyUser metódusban.
* A DB osztály használata a példában elavult. A Laravel Eloquent ORM ajánlott az adatbázis műveletekhez.

Modelek

FishingPlace Model Leírás

**Leírás:**

A FishingPlace modell a horgászhelyeket reprezentálja a Laravel projektben. Ez az Eloquent ORM modellt használja az adatbázis műveletekhez.

**Adattábla:**

A FishingPlace modell feltételezhetően egy fishing\_places nevű adatbázis táblához kapcsolódik.

**Mezők:**

A modell a következő mezőket definiálja az adatbázis táblában:

* id (integer, primary key): A horgászhely azonosítója (automatikusan növekvő).
* pier (boolean): Igaz, ha van móló a horgászhelyen.
* firepit (boolean): Igaz, ha van tűzrakóhely a horgászhelyen.
* shelter (boolean): Igaz, ha van menedék a horgászhelyen.
* description (text): A horgászhely leírása.
* longitude (float): A horgászhely földrajzi hosszúsága.
* latitude (float): A horgászhely földrajzi szélessége.

**Property-k:**

* $timestamps: A modell false értékre van állítva, ami azt jelenti, hogy az Eloquent nem fog automatikusan létrehozni created\_at és updated\_at mezőket a táblában.
* $fillable: Ez a property meghatározza, hogy mely mezők tömeges feltöltése engedélyezett a modell példányosításakor (create vagy update metódusok).

**Kapcsolatok:**

* Reservation(): A modell egy hasMany kapcsolatot definiál a Reservation modellel. Ez azt jelenti, hogy egy horgászhelyhez több foglalás is tartozhat.

Reservation Model Leírás

**Leírás:**

A Reservation modell a horgászhely foglalásokat reprezentálja a Laravel projektben. Ez az Eloquent ORM modellt használja az adatbázis műveletekhez.

**Adattábla:**

A Reservation modell feltételezhetően egy reservations nevű adatbázis táblához kapcsolódik.

**Mezők:**

A modell a következő mezőket definiálja az adatbázis táblában:

* id (integer, primary key): A foglalás azonosítója (automatikusan növekvő).
* user\_id (integer): A foglalást végző felhasználó azonosítója.
* fishingplace\_id (integer): A foglaláshoz kapcsolódó horgászhely azonosítója.
* reservationStart (date): A foglalás kezdete.
* reservationEnd (date): A foglalás vége.
* guestNumber (integer): A foglaláshoz tartozó személyek száma.
* actualRate (integer): Ez a mező nincs megjegyzéssel ellátva a kódban, így a funkciója nem teljesen világos. Lehet, hogy a tényleges fizetendő díjat tárolja, de további információ szükséges a mező pontos meghatározásához.

**Property-k:**

* $fillable: Ez a property meghatározza, hogy mely mezők tömeges feltöltése engedélyezett a modell példányosításakor (create vagy update metódusok).

**Kapcsolatok:**

* fishingplace(): A modell egy belongsTo kapcsolatot definiál a FishingPlace modellel. Ez azt jelenti, hogy egy foglaláshoz egy horgászhely tartozik.
* user(): A modell egy belongsTo kapcsolatot definiál a User modellel. Ez azt jelenti, hogy egy foglaláshoz egy felhasználó tartozik.

User Model Leírás

**Leírás:**

A User modell a felhasználókat reprezentálja a Laravel projektben. Ez az Eloquent ORM modellt és a Laravel Auth csomagot használja a felhasználókezeléshez, beleértve a regisztrációt, bejelentkezést és egyéb jogosultságkezelési funkciókat.

**Adattábla:**

A User modell feltételezhetően egy users nevű adatbázis táblához kapcsolódik.

**Mezők:**

A modell a következő mezőket definiálja az adatbázis táblában:

* id (integer, primary key): A felhasználó azonosítója (automatikusan növekvő).
* name (string): A felhasználó neve.
* email (string): A felhasználó e-mail címe (egyedi).
* password (string): A felhasználó jelszava (titkosítva).
* birthdate (date): A felhasználó születési dátuma.
* verifycode (string): Ez a mező nincs megjegyzéssel ellátva a kódban, így a funkciója nem teljesen világos. Lehet, hogy a felhasználó e-mail címének megerősítéséhez használt kód, de további információ szükséges a mező pontos meghatározásához.
* email\_verified\_at (datetime): Az a dátum és időpont, amikor a felhasználó e-mail címe megerősítésre került.
* remember\_token (string): A felhasználó bejelentkezési tokenje (opcionális).

**Property-k:**

* $fillable: Ez a property meghatározza, hogy mely mezők tömeges feltöltése engedélyezett a modell példányosításakor (create vagy update metódusok).
* $hidden: Ez a property meghatározza, hogy mely mezőket kell kizárni a modell JSON reprezentációjából biztonsági okokból (pl. jelszó, token).
* $casts: Ez a property meghatározza, hogy bizonyos mezőket hogyan kell átalakítani a adatbázis tárolás és lekérdezés során (pl. dátum, titkosítás).

**Kapcsolatok:**

A User modell önmagában nem definiál kapcsolatokat más modellekkel. Azonban más modellek, például a Reservation modell, belongsTo kapcsolatot definiálhat a User modellel, hogy meghatározza a foglaláshoz tartozó felhasználót.

**Laravel Auth Integráció:**

A User osztály az Authenticatable osztályt örökli, amely a Laravel Auth csomag alapvető felhasználókezelési funkcióit biztosítja. Ez lehetővé teszi a felhasználók regisztrációját, bejelentkezését, kijelentkezését és egyéb jogosultságkezelési műveleteket.

image model Leirás

**Funkció:**

Ez az osztály a images adatbázis táblához kapcsolódó Eloquent modellt definiálja a Laravel alkalmazásban. Az Eloquent modellek segítenek a

* rekordokkal való interakcióban (lekérdezés, létrehozás, frissítés, törlés)
* az adatbázis-táblák mezőinek leképezésében

**Mezők:**

* id (automatikusan növekvő egész szám, primary key) - **Az Eloquent modelleknél ezt a mezőt általában nem kell definiálni, automatikusan hozzáadódik.**
* filename (string) - A kép fájlneve
* description (string) - A kép leírása
* user\_id (integer) - Idegen kulcs, amely a képet feltöltő felhasználóra mutat (feltételezhetően hivatkozik a users tábla id mezőjére)

**Metódusok:**

* fillable (tömb): Ez a tulajdonság meghatározza, hogy mely tömegelemek tömegsorokkal való tömeges feltöltéséhez vagy létrehozásához használhatóak. A példában a filename, description, és user\_id mezők engedélyezettek a tömeges műveletekhez.

Migrations

Users\_Table:

**Leírás:**

Ez a Laravel migráció a users nevű adatbázis táblát hozza létre. A migrációk olyan fájlok, amelyek a projekt adatbázis struktúrájának módosításait kezelik. A Laravel Schema Builder osztályt használja a tábla létrehozásához és a szükséges mezők definiálásához.

**Tábla:** users

**Mezők:**

* id (integer, primary key): Automatikusan növekvő egész szám, amely az adatbázis sorok egyedi azonosítója.
* name (string): A felhasználó neve.
* email (string, unique): A felhasználó e-mail címe. Ez a mező egyedi értéket kell tartalmazzon, azaz nem lehet két felhasználónak azonos e-mail címe.
* email\_verified\_at (date, nullable): A felhasználó e-mail címének megerősítési dátuma és ideje. Ez a mező lehet null értékű, ha a felhasználó még nem erősítette meg az e-mail címét.
* password (string): A felhasználó jelszava (titkosítva).
* remember\_token (string, nullable): A felhasználó bejelentkezési tokenje. Ez a mező lehet null értékű, ha a felhasználó nem választotta a "Bejelentkezve tartás" lehetőséget bejelentkezéskor.
* timestamps (datetime): A Laravel által automatikusan karbantartott mezők, amelyek a sor létrehozásának és frissítésének dátumát és időpontját tárolják.

**Metódusok:**

* up(): Ez a metódus fut le a migráció végrehajtásakor. A Schema::create metódust használja a users tábla létrehozásához, és a Blueprint osztályt a mezők definiálásához.
* down(): Ez a metódus a migráció visszaállításakor fut le. A Schema::dropIfExists metódust használja a users tábla törléséhez.

**Megjegyzés:**

* A Laravel biztonsági okokból titkosítja a felhasználói jelszavakat a tárolás során.

Ilyen módon lehet leírni a további migrációkat is. A leírásnak tartalmaznia kell a migráció által létrehozott vagy módosított táblák nevét, a táblák mezőit, valamint az up() és down() metódusok funkcióit.

Fishing\_places\_table

**Leírás:**

Ez a Laravel migráció a fishing\_places nevű adatbázis táblát hozza létre. Ahogy az első migráció, ez is a Laravel Schema Builder osztályt használja a tábla létrehozásához és a szükséges mezők definiálásához.

**Tábla:** fishing\_places

**Mezők:**

* id (integer, primary key): Automatikusan növekvő egész szám, amely az adatbázis sorok egyedi azonosítója.
* pier (boolean): Logikai érték, amely jelzi, hogy van-e móló a horgászhelyen (alapértelmezett érték: 0 - nincs).
* firepit (boolean): Logikai érték, amely jelzi, hogy van-e tűzrakóhely a horgászhelyen (alapértelmezett érték: 0 - nincs).
* shelter (boolean): Logikai érték, amely jelzi, hogy van-e menedék a horgászhelyen (alapértelmezett érték: 0 - nincs).
* description (string): A horgászhely leírása.
* longitude (string): A horgászhely földrajzi hosszúsága.
* latitude (string): A horgászhely földrajzi szélessége.
* timestamps (datetime): Ez a megjegyzés a kód **nincs** benne a migrációban, így valószínűleg nincsenek automatikusan karbantartott időbélyegző mezők a táblában.

**Metódusok:**

* up(): Ez a metódus fut le a migráció végrehajtásakor. A Schema::create metódust használja a fishing\_places tábla létrehozásához, és a Blueprint osztályt a mezők definiálásához.
* down(): Ez a metódus a migráció visszaállításakor fut le. A Schema::dropIfExists metódust használja a fishing\_places tábla törléséhez.

**Megjegyzés:**

A logikai mezők alapértelmezett értékeként 0 (hamis) van megadva. Ez azt jelenti, hogy ha a horgászhely nem rendelkezik mólóval, tűzrakóhellyel vagy menedékkel, akkor ezek a mezők automatikusan 0 értéket kapnak.

add\_additionaldata\_users\_table

**Leírás:**

Ez a Laravel migráció módosítja a meglévő users adatbázis táblát. A Schema::table metódust használja a táblához új mezők hozzáadására.

**Módosított Tábla:** users

**Hozzáadott Mezők:**

* verifycode (string, default 0): Ez a mező a kód alapján valószínűleg a felhasználó e-mail címének megerősítéséhez használt kód tárolására szolgál. Az alapértelmezett érték 0.
* birthdate (date): A felhasználó születési dátuma.
* userlevel (boolean, default 0): Ez a mező a felhasználói jogosultsági szintet jelzi (alapértelmezett érték: 0).

**Metódusok:**

* up(): Ez a metódus fut le a migráció végrehajtásakor. A Schema::table metódust használja a users táblához való hozzáféréshez, és a Blueprint osztályt az új mezők definiálásához.
* down(): A kód **nem** tartalmazza ennek a metódusnak a megvalósítását. Ez azt jelenti, hogy a migráció visszaállításának (a módosítások visszavonásának) jelenleg nincs implementálva.

reservations\_table

**Leírás:**

Ez a Laravel migráció a reservations nevű adatbázis táblát hozza létre. Ahogy az előző migrációk, ez is a Laravel Schema Builder osztályt használja a tábla létrehozásához és a szükséges mezők definiálásához.

**Tábla:** reservations

**Mezők:**

* id (integer, primary key): Automatikusan növekvő egész szám, amely az adatbázis sorok egyedi azonosítója.
* user\_id (foreign key): A foglalást rögzítő felhasználóra mutató idegen kulcs. Feltételezhetően hivatkozik a users tábla id mezőjére.
* fishingplace\_id (foreign key): A foglaláshoz tartozó horgászhelyre mutató idegen kulcs. Feltételezhetően hivatkozik a fishing\_places tábla id mezőjére.
* reservationStart (date): A foglalás kezdete.
* reservationEnd (date): A foglalás vége.
* guestNumber (integer, default 1): A foglalkozáshoz tartozó személyek száma. Az alapértelmezett érték 1.
* timestamps (datetime): A Laravel által automatikusan karbantartott mezők, amelyek a sor létrehozásának és frissítésének dátumát és időpontját tárolják.

**Metódusok:**

* up(): Ez a metódus fut le a migráció végrehajtásakor. A Schema::create metódust használja a reservations tábla létrehozásához, és a Blueprint osztályt a mezők definiálásához.
* down(): Ez a metódus a migráció visszaállításakor fut le. A Schema::dropIfExists metódust használja a reservations tábla törléséhez.

**Megjegyzés:**

* A migráció idegen kulcsokat használ a users és fishing\_places táblákhoz való kapcsolódáshoz. Ez biztosítja az adatintegritást és megakadályozza az érvénytelen adatok bevitelét.

images\_table

**Leírás:**

Ez a Laravel migráció az images nevű adatbázis táblát hozza létre. Ahogy az előző migrációk, ez is a Laravel Schema Builder osztályt használja a tábla létrehozásához és a szükséges mezők definiálásához.

**Tábla:** images

**Mezők:**

* id (integer, primary key): Automatikusan növekvő egész szám, amely az adatbázis sorok egyedi azonosítója.
* filename (string): A kép fájlneve.
* description (string): A kép leírása.
* user\_id (integer): A képet feltöltő felhasználóra mutató idegen kulcs. Feltételezhetően hivatkozik a users tábla id mezőjére.
* timestamps (datetime): A Laravel által automatikusan karbantartott mezők, amelyek a sor létrehozásának és frissítésének dátumát és időpontját tárolják.

**Metódusok:**

* up(): Ez a metódus fut le a migráció végrehajtásakor. A Schema::create metódust használja az images tábla létrehozásához, és a Blueprint osztályt a mezők definiálásához.
* down(): Ez a metódus a migráció visszaállításakor fut le. A Schema::dropIfExists metódust használja az images tábla törléséhez.

**Megjegyzés:**

* A migráció idegen kulcsot használ a users táblához való kapcsolódáshoz. Ez lehetővé teszi annak nyomon követését, hogy melyik felhasználó töltött fel egy adott képet.

Seeders:

Admin Felhasználó Seeder Leírás

**Leírás:**

Ez a Laravel seeder két rekordot illeszt be a users adatbázis táblába. A rekordok adminisztrátori felhasználókat hoznak létre az alkalmazáshoz.

**Metódusok:**

* run(): Ez a metódus a DB::table metódust használja a users táblába való adatbeillesztéshez. Két INSERT műveletet hajt végre, mindegyiket a szükséges mezőkkel és értékekkel.

**Beillesztett Adatok:**

* **Első Sor:**
  + name: Vitovszki Tomi
  + email: tomin@gmail.com
  + email\_verified\_at: 2001-09-23 (e-mail cím megerősítésének dátuma)
  + password: Aa123. (titkosított jelszó)
  + verifycode: 67891 (valószínűleg e-mail megerősítéshez használt kód)
  + birthdate: 2001.07.23 (születési dátum)
  + userlevel: 1 (adminisztrátori jogosultsági szint)
* **Második Sor:**
  + name: Plesovszki István
  + email: istvan@gmail.com
  + email\_verified\_at: 2001-09-23 (e-mail cím megerősítésének dátuma)
  + password: Aa123. (titkosított jelszó)
  + verifycode: 12345 (valószínűleg e-mail megerősítéshez használt kód)
  + birthdate: 2001.07.23 (születési dátum)
  + userlevel: 0 (alapértelmezett felhasználói jogosultsági szint)

**Megjegyzés:**

* A jelszavakat a Laravel bcrypt függvényével titkosítják a biztonság érdekében.
* A verifycode mező pontos funkciója további vizsgálatot igényelhet.

Horgászhely Táblázat Seeder Leírás

**Leírás:**

Ez a Laravel seeder hat rekordot illeszt be a fishing\_places adatbázis táblába. A rekordok horgászhelyeket hoznak létre az alkalmazáshoz, mindegyik a következő jellemzőkkel:

* **móló:** van-e móló a horgászhelyen (boolean)
* **tűzrakóhely:** van-e tűzrakóhely a horgászhelyen (boolean)
* **menedék:** van-e menedék a horgászhelyen (pl. esővédő) (boolean)
* **leírás:** a horgászhely rövid leírása
* **hosszúság:** a horgászhely GPS koordinátái (hosszúság)
* **szélesség:** a horgászhely GPS koordinátái (szélesség)

**Metódusok:**

* run(): Ez a metódus az DB::select metódust használja a fishing\_places táblába való adatbeillesztéshez. Hat INSERT műveletet hajt végre, mindegyiket a szükséges mezőkkel és értékekkel.

**Beillesztett Adatok:**

A seeder hat horgászhelyet hoz létre az alábbi specifikációkkal:

1. Móló, tűzrakóhely, menedék, "Gyönyörű tó tiszta vízzel és bőséges halállománnyal.", 19.572352, 47.536311
2. Nincs móló, tűzrakóhely, menedék, "Festői folyópart nyugodt hangulattal.", 19.572169, 47.534790
3. Móló, nincs tűzrakóhely, nincs menedék, "Bársonyos növényzettel körülvett hangulatos tavacska.", 19.570300, 47.535814
4. Móló, tűzrakóhely, menedék, "Csendes tó, karbantartott horgászlétesítményekkel.", 19.567223, 47.537463
5. Nincs móló, tűzrakóhely, nincs menedék, "Nyugodt folyóparti hely rusztikus tűzhellyel.", 19.566816, 47.538346
6. Nincs móló, tűzrakóhely, menedék, "Nyugalmas tavacska fedett menedékkel a pihenéshez.", 19.574025, 47.535666

**Megjegyzés:**

Ez a seeder nem használ Eloquent modelleket. A DB::select metódust használja az SQL INSERT utasítások közvetlen végrehajtására.

Foglalás Táblázat Seeder Leírás

**Leírás:**

Ez a Laravel seeder két rekordot illeszt be a reservations adatbázis táblába. A rekordok foglalásokat hoznak létre a rendszerben.

**Metódusok:**

* run(): Ez a metódus az DB::table metódust használja a reservations táblába való adatbeillesztéshez. Két INSERT műveletet hajt végre, mindegyiket a szükséges mezőkkel és értékekkel.

**Beillesztett Adatok:**

A seeder két foglalást hoz létre a következő adatokkal:

1. Foglalás:
   * user\_id: 2 (feltételezhetően a felhasználói azonosító)
   * fishingplace\_id: 3 (feltételezhetően a horgászhely azonosítója)
   * reservationStart: 2024-05-28 (foglalás kezdete)
   * reservationEnd: 2024-05-30 (foglalás vége)
   * guestNumber: 3 (vendégek száma)
2. Foglalás:
   * user\_id: 2 (feltételezhetően a felhasználói azonosító)
   * fishingplace\_id: 4 (feltételezhetően a horgászhely azonosítója)
   * reservationStart: 2024-05-28 (foglalás kezdete)
   * reservationEnd: 2024-05-30 (foglalás vége)
   * guestNumber: 4 (vendégek száma)

**Megjegyzés:**

* A seeder nem használ Eloquent modelleket. Az DB::table metódust használja az SQL INSERT utasítások közvetlen végrehajtására.
* A user\_id és fishingplace\_id mezők értékei feltételezhetően a kapcsolódó táblákban (felhasználók és horgászhelyek) tárolt azonosítókra hivatkoznak.-

Felhasználó Táblázat Seeder Leírás

**Leírás:**

Ez a Laravel seeder tizenöt rekordot illeszt be a users adatbázis táblába. A rekordok tesztadatokat hoznak létre a fejlesztés számára.

**Metódusok:**

* run(): Ez a metódus a Faker könyvtárat használ a véletlenszerű adatok generálásához, és a DB::table metódust az adatbeillesztéshez. Egy ciklus segítségével tizenöt alkalommal végrehajtja az adatbeillesztést.

**Használt Könyvtárak:**

* Faker: Ez a könyvtár véletlenszerű adatokat generál, például neveket, e-maileket és dátumokat.

**Beillesztett Adatok:**

A seeder tizenöt felhasználói rekordot hoz létre az alábbi mezőkkel:

* name: Teljes név (véletlenszerűen generált a Faker könyvtárral)
* email: E-mail cím (véletlenszerűen generált a Faker könyvtárral)
* password: Jelszó (véletlenszerűen generált a Faker könyvtárral, titkosítva a Hash::make függvénnyel)
* birthdate: Születési dátum (véletlenszerűen generált a Faker könyvtárral)
* userlevel: Felhasználói jogosultsági szint (0 - alapértelmezett felhasználó)

**Megjegyzés:**

* A jelszavakat a Laravel Hash::make függvénnyel titkosítják a biztonság érdekében.
* A tesztadatok generálásához a Faker könyvtárat használjuk. Így valósághűbb adatokat kapunk, mint a manuálisan megadott értékek.
* A userlevel mező értéke minden rekordnál 0, ami az alapértelmezett felhasználói jogosultsági szintet jelenti.

Frontend

Home page:

* Képkarusszel: Képek jelennek meg a horgászatot bemutató diavetítés formájában. A felhasználók manuálisan vagy automatikusan lapozhatnak a képek között.
* Szöveges szekciók: Üdvözlő üzenetek és leírások olvashatók a horgásztóhoz kapcsolódó kényelmi szolgáltatásokról és lehetőségekről.
* Foglalás kártya: Rövid leírás látható a foglalási funkcióról, és egy gomb segítségével tovább lehet lépni a foglalási oldalra.
* Galéria kártya: Az oldalon szerepel egy rövid leírás a galériafunkcióról, és egy gomb segítségével lehetőség van a galériát megtekinteni.
* További szöveges szekció: Egy további szöveges részlet bátorítja a felhasználókat, hogy fedezzék fel a horgászati élményt, amit a horgásztelep kínál.
* Metódusok itt nincsenek.

About us:

Ez az oldal struktúrája és tartalma részletesen bemutatja a Dányi Tó és a hozzá kapcsolódó horgásztelep jellemzőit és lehetőségeit.

* Cím és bevezető szöveg: Az oldal felső részén egy cím és egy rövid bevezető szöveg található, amely a Dányi Tó természeti jellemzőit és vonzerejét emeli ki.
* Képek: Két kép látható a Dányi Tóról, amelyek megmutatják a tó környezetét és különlegességeit.
* Felkapott halak: Egy rövid leírás olvasható a tóban található különböző halakról, kiegészítve képekkel.
* Horgászidények: A szöveges rész elmagyarázza, hogy az év különböző időszakaiban mire lehet számítani a horgászat során a Dányi Tónál.
* Dányi Horgásztelep: Egy további szöveges rész tartalmazza a horgásztelep bemutatását, beleértve a környezetet, az infrastruktúrát és az eseményeket.
* Helyszín információk: Az oldal végén olyan információk találhatók, mint a település neve, ahol a tó és a horgásztelep található, valamint egy link a Google Térképhez.
* Metódusok itt nincsenek

Book main:

Térkép oldal (HTML):

* A <h1> elem a térkép oldal címét tartalmazza.
* Az <app-map> komponens beágyazása a térkép megjelenítéséhez.
* Egy link található a Google Térképhez, amely megmutatja, hol található a tó.

Térkép komponens (TypeScript):

* A CustomMarker osztály egyedi Leaflet jelölőket definiál a térképhez.
* Az ngOnInit() függvény inicializálja a térképet és betölti a horgászhelyeket.
* A closeModal() függvény bezárja a hely részleteit megjelenítő modális ablakot.
* Az initializeMap() függvény létrehozza és inicializálja a térképet.
* A addMarkers() függvény hozzáadja a jelölőket a térképhez és beállítja a kattintás eseménykezelőt.
* A centerMap() függvény középre helyezi a térképet a horgászhelyek körül.
* A trueOrFalse() függvény megállapítja, hogy egy adott tulajdonság igaz-e vagy hamis.
* A navigate() függvény a felhasználót átirányítja a foglalási űrlapra, ha be van jelentkezve.
* A loggedIn() függvény ellenőrzi, hogy a felhasználó be van-e jelentkezve.
* Az openSnackBar() függvény egy snackbar értesítést jelenít meg a felhasználónak.

Book form:

Ezek a komponensek lehetővé teszik a felhasználók számára, hogy foglalást hozzanak létre a kiválasztott horgászhelyen, és megjelenítsék annak képeit a karusszel segítségével. A TypeScript kód felelős az adatok feldolgozásáért és az interakciók kezeléséért.

Foglalási űrlap (HTML):

* Az <h2> címsor után a kiválasztott hely képeinek megjelenítése.
* Egy karusszel található, amely a kiválasztott hely képeit jeleníti meg.
* Egy űrlap, amelyben a felhasználó megadhatja a foglalási információkat, mint a kezdő és végső dátum, valamint az érkező vendégek száma.
* Az űrlap validálása után, valamint elküldése után, a gomb visszairányítja a felhasználót az előző oldalra.

Foglalási űrlap komponens (TypeScript):

* Az osztály felelős a foglalási adatok kezeléséért, a felhasználói interakciók figyeléséért és a kiválasztott hely képeinek betöltéséért.
* Az ngOnInit() függvény inicializálja az alapvető változókat és beállítja az alapértelmezett foglalási dátumokat.
* A getImageUrl() függvény a kiválasztott hely képeinek URL-jét állítja elő a megfelelő helyszám és fájlnév alapján.
* Az ngAfterViewInit() függvény indítja el a karusszel automatikus lejátszását.
* A prevSlide() és nextSlide() függvények a karusszel előző és következő diáját állítják be.
* A sendReservation() függvény elküldi a foglalási adatokat a szervernek, és megjeleníti az üzeneteket a sikeres vagy sikertelen foglalás esetén.
* Az showCustomAlert() és hideCustomAlert() függvények egyedi figyelmeztetéseket jelenítenek meg vagy rejtenek el.

Login:

Ezek a komponensek lehetővé teszik a felhasználók számára, hogy bejelentkezzenek az alkalmazásba, ellenőrizzék az e-mail címet, jelszót, és megfelelő üzeneteket kapjanak a bejelentkezés eredményéről.

Bejelentkezési űrlap (HTML):

* Az <h1> címsor a bejelentkezés oldalának címét jeleníti meg.
* Egy űrlap található, amelyben a felhasználó megadhatja az e-mail címét és a jelszavát.
* Az e-mail cím és jelszó mezők kötelezőek, és megjelenik egy hibaüzenet, ha nem megfelelő formátumban vannak.
* Egy gomb, amely lehetővé teszi az e-mail cím megerősítését.
* Gombok a bejelentkezés, regisztráció és visszalépés funkcióhoz.

Bejelentkezési komponens (TypeScript):

* Az osztály felelős a bejelentkezési adatok kezeléséért, a felhasználói interakciók figyeléséért és az authentikációs szolgáltatások hívásáért.
* Az ngOnInit() függvény inicializálja az űrlapot és annak validátorait.
* Az onSubmit() függvény kezeli az űrlap beküldését, ellenőrzi az adatokat, majd elküldi azokat a szervernek.
* Az openSnackBar() függvény megjeleníti a felugró üzeneteket sikeres vagy sikertelen bejelentkezés esetén.

Sign up:

Ezek a komponensek lehetővé teszik az új felhasználók regisztrációját az alkalmazásba, ellenőrzik az űrlap kitöltését, és megjelenítik a megfelelő üzeneteket a regisztráció sikerességéről vagy hibáiról.

Regisztrációs űrlap (HTML):

* Egy <h1> címsor jelzi a regisztrációs oldal címét.
* Négy mezőt tartalmaz: teljes név, e-mail cím, jelszó, jelszó megerősítése és születési dátum.
* Az e-mail cím, jelszó és jelszó megerősítése mezők kötelezőek, és megjelenik egy hibaüzenet, ha a felhasználó nem megfelelően tölti ki ezeket a mezőket.
* Az e-mail cím mező mellett egy kis szöveges üzenet jelenik meg, ha az e-mail cím formátuma nem megfelelő.
* A regisztráció gomb aktív, ha az összes kötelező mező megfelelően van kitöltve.

Regisztrációs komponens (TypeScript):

* Az osztály felelős az új felhasználók regisztrációjáért, az űrlap validálásáért és a szükséges információk kezeléséért.
* Az isValidEmail() függvény ellenőrzi az e-mail cím formátumát.
* Az isFormValid() függvény ellenőrzi, hogy az összes kötelező mező megfelelően van-e kitöltve.
* A signUp() függvény kezeli az űrlap beküldését, elküldi az adatokat a szervernek, és navigál a megfelelő oldalra a regisztráció sikeressége esetén.
* Az openSnackBar() függvény megjeleníti a felugró üzeneteket sikeres vagy sikertelen regisztráció esetén.

Verify email:

Ez a komponens lehetővé teszi a felhasználók számára, hogy megerősítsék az e-mail címüket azáltal, hogy megadják a kapott megerősítő kódot, és ezáltal bejelentkezzenek a rendszerbe. A megfelelő visszajelzések megjelenítése segíti a felhasználókat a folyamat során.

HTML kód:

* Egy űrlapot jelenít meg, amely egy megerősítő kód bevitelére szolgál.
* Az input mezőt megfelelő címkével látjuk el, amely leírja, mit kell megadni az input mezőbe.
* A "Küldés" gombra kattintva az űrlapot elküldi, ami a verifyEmail() függvényt hívja meg.

TypeScript kód:

* Az osztály egy code változót tartalmaz, amelybe a felhasználó által megadott megerősítő kódot tároljuk.
* Az email és password változók a bejelentkezéshez használt adatokat tárolják, amelyeket a komponens az URL paramétereiből kap meg.
* Az ngOnInit() metódus lekéri az URL paramétereit (email és jelszó), és beállítja az email és password változókat.
* A verifyEmail() metódus meghívja az auth.verifyEmail() függvényt, hogy ellenőrizze a megerősítő kódot. Ha az ellenőrzés sikeres, a felhasználót bejelentkezteti a login() függvény segítségével.
* A login() metódus bejelentkezteti a felhasználót az auth.login() függvény segítségével. Ha a bejelentkezés sikeres, akkor megjeleníti a sikeres üzenetet és átirányítja a felhasználót a főoldalra (/home), ellenkező esetben hibaüzenetet jelenít meg.

Profile

Ez a komponens lehetővé teszi a felhasználók számára, hogy frissítsék a profiljukat, valamint töröljék a fiókjukat, ha szükséges. A felhasználók adatainak kezelése, valamint a megfelelő üzenetek megjelenítése segít a felhasználók interakcióinak hatékony kezelésében és az alkalmazás használatának kényelmességében.

HTML kód:

* Egy űrlapot tartalmaz, amely lehetővé teszi a felhasználói profil frissítését.
* A felhasználó nevének, e-mail címének és születési dátumának frissítésére szolgáló mezőket tartalmazza.
* Egy "Fiók törlése" gombot is tartalmaz, ami egy modal ablakot nyit meg a felhasználói fiók törléséhez.
* A modal ablakban a felhasználónak lehetősége van megerősíteni a fiók törlését.

TypeScript kód:

* Az osztály felelős a felhasználói adatok kezeléséért, a profil frissítéséért és a fiók törléséért.
* A getData() metódus lekéri a felhasználói adatokat a szerverről, és beállítja azokat a megfelelő mezőkben.
* Az updateProfile() metódus frissíti a felhasználó profilját a megadott adatok alapján, és megjeleníti az üzenetet a frissítés sikerességéről.
* A deleteUser() metódus törli a felhasználó fiókját, ha a felhasználó megerősíti a törlést a modal ablakban.
* Az openSnackBar() metódus megjeleníti a felugró üzeneteket a felhasználói műveletek sikerességéről vagy sikertelenségéről.

Reservation:

Ez a komponens lehetővé teszi a felhasználók számára, hogy kezeljék foglalásaikat, beleértve a szerkesztést és a törlést is. A felhasználói interakciók kezelése és a megfelelő visszajelzések megjelenítése segít a felhasználók számára, hogy hatékonyan használják az alkalmazást.

HTML kód:

* A foglalások listáját jeleníti meg egy ngFor direktívával, minden foglalás egy kártyában.
* Minden kártyán megjelenik a helyszám, valamint szerkesztés és törlés gomb.
* Alul egy visszalépés gomb található, amely visszairányít a fő foglalási oldalra.
* A HTML kód tartalmaz egy modális ablakot is, amelyben szerkeszteni lehet a foglalást.

TypeScript kód:

* A komponens felelős a felhasználó foglalásainak lekéréséért, frissítéséért és törléséért.
* A getReservations() metódus lekéri a felhasználó foglalásait, majd a getUser() metódust használja, hogy lekérje a felhasználó nevét minden foglaláshoz.
* A updateReservationData() metódus előkészíti a foglalás szerkesztéséhez szükséges adatokat, majd megnyitja a szerkesztéshez szükséges modális ablakot.
* A deleteReservation() metódus törli a megadott foglalást, majd frissíti a foglalások listáját.
* Az updateReservation() metódus frissíti a foglalás adatait a megadott adatok alapján.
* A getUser() metódus lekéri a felhasználó nevét az azonosító alapján.

Gallery

Ez a kód lehetővé teszi a felhasználók számára, hogy feltöltsék, szerkesszék és töröljék a képeiket a galériában, valamint megváltoztathassák a képek leírását. A megfelelő visszajelzések segítenek a felhasználóknak abban, hogy megfelelően navigáljanak a galériában.

HTML kód:

* A Kép feltöltés gombra kattintva megjelenik egy modális ablak, ahol a felhasználók feltölthetik a képeket.
* A galéria részben a feltöltött képek jelennek meg kártyák formájában. Minden kártyán megjelenik a kép, a felhasználó neve, és a kép leírása.
* Ha a felhasználó a kép tulajdonosa vagy rendszergazda, akkor lehetősége van a kép törlésére vagy szerkesztésére.

TypeScript kód:

* A GalleryComponent osztály kezeli a galéria működését.
* Az ngOnInit() metódus meghívja a fetchImages() függvényt, hogy betöltse a képeket a galériába.
* A fetchImages() függvény meghívja az imageService.getImages() függvényt, hogy lekérje a képeket a szerverről, majd betölti ezeket a images tömbbe.
* A canModifyOrDelete() függvény ellenőrzi, hogy a felhasználó jogosult-e a kép módosítására vagy törlésére.
* Az onFileSelected() függvény tárolja a felhasználó által kiválasztott fájlt.
* Az onSubmit() függvény felelős a kép feltöltéséért a szerverre. A feltöltés után frissíti a galériát.
* A deleteImage() függvény törli a kiválasztott képet a szerverről, majd frissíti a galériát.
* A modifyImageDescription() függvény lehetővé teszi a felhasználó számára, hogy módosítsa a kép leírását. Ezután frissíti a galériát.
* Az openEditModal() függvény megnyitja a szerkesztési modált, és betölti az aktuális kép adatait a szerkesztéshez.
* Az onEditSubmit() függvény elindítja a kép leírásának módosítását.

Admin

1. Felhasználók kártya:

Feladata:

Ez a kód lehetővé teszi az adminisztrátorok számára, hogy megtekintsék, szerkesszék és töröljék a felhasználói adatokat. Az adminisztrátori felület kártyákon jeleníti meg a felhasználó nevét, e-mail címét és születési dátumát. Az adminisztrátoroknak lehetőségük van a felhasználói adatok szerkesztésére és törlésére. Emellett egy modalitás űrlap is rendelkezésre áll a felhasználói információk frissítésére.

HTML kód:

* A "Felhasználók" cím és egy vissza gomb jelenik meg a tetején.
* A felhasználói adatok egy cikluson belül kártyákon jelennek meg, megjelenítve a felhasználó nevét, e-mail címét és születési dátumát. Minden kártya tartalmaz gombokat a felhasználói adatok szerkesztéséhez és törléséhez.
* Az alján még egy vissza gomb található navigációhoz.
* Egy modalitás űrlap is rendelkezésre áll a felhasználói információk frissítéséhez, beleértve a név, e-mail és születési dátum mezőket.

TypeScript kód:

* A UsersComponent osztály kezeli az adminisztrátori felület működését.
* Az ngOnInit() metódus lekéri a felhasználói adatokat, amikor a komponens inicializálódik.
* A getUsers() metódus lekéri a felhasználói adatokat a szerverről, és frissíti a helyi felhasználók tömbjét.
* A deleteUser() metódus elküldi a felhasználó törlését kérő kérést és frissíti a felhasználói listát ennek megfelelően.
* Az updateUser() metódus elküldi a felhasználó adatainak frissítését kérő kérést és sikeres frissítés esetén egy üzenetet jelenít meg.
* A showUpdateMessage() metódus az Angular Material Snackbar segítségével megjeleníti az üzenetet a sikeres profilfrissítésről.
* Az updateUserData() metódus előtölti a modalitás űrlapot a kiválasztott felhasználó adataival.
* Az openModal() metódus programatikusan megnyitja a modalitás űrlapot a felhasználói információk frissítéséhez.

1. Horgászhelyek kártya:

Feladata:

Ez a kód lehetővé teszi az adminisztrátorok számára, hogy megtekintsék, szerkesszék és töröljék a horgászhelyek adatait. Az adminisztrátori felület kártyákon jeleníti meg a horgászhelyek azonosítóját és a rendelkezésre álló szolgáltatásokat. Az adminisztrátoroknak lehetőségük van a horgászhelyek adatainak szerkesztésére és törlésére. Emellett modalitás űrlapok is rendelkezésre állnak a horgászhelyek adatainak frissítésére és új horgászhelyek hozzáadására.

HTML kód:

* A "Horgászhelyek" cím és két gomb jelenik meg a tetején. Az egyik a visszalépéshez, a másik pedig a horgászhely hozzáadásához.
* A horgászhelyek adatai egy cikluson belül kártyákon jelennek meg, megjelenítve a horgászhely azonosítóját és a rendelkezésre álló szolgáltatásokat. Minden kártya tartalmaz gombokat a horgászhely adatainak szerkesztéséhez és törléséhez.
* Az alján még egy vissza gomb található navigációhoz.

TypeScript kód:

* A SpotsComponent osztály kezeli az adminisztrátori felület működését.
* Az ngOnInit() metódus lekéri a horgászhelyek adatait, amikor a komponens inicializálódik.
* A getFishingplaces() metódus lekéri a horgászhelyek adatait a szerverről, és frissíti a helyi tömböt.
* A deleteFishingplace() metódus elküldi a horgászhely törlését kérő kérést és frissíti a horgászhelyek listáját ennek megfelelően.
* Az updateFishingplaceData() metódus előtölti a szerkesztési modalitás űrlapot a kiválasztott horgászhely adataival.
* Az addFishingplaceData() metódus megnyitja az új horgászhely hozzáadását segítő modalitás űrlapot.
* Az openUpdateModal() és az openAddModal() metódusok programatikusan megnyitják a megfelelő modalitás űrlapokat.
* Az updateFishingplace() és az addFishingplace() metódusok elküldik a frissítési és hozzáadási kéréseket, majd megjelenítik a sikeres üzeneteket.
* A showUpdateMessage() és showAddMessage() metódusok az Angular Material Snackbar segítségével jelenítik meg az üzeneteket a sikeres frissítések és hozzáadásokról.

1. Foglalások kártya:

Feladata:

Ez a kód lehetővé teszi az adminisztrátorok számára, hogy megtekintsék, szerkesszék és töröljék a foglalások adatait. Az adminisztrátori felület kártyákon jeleníti meg a foglalások adatait, beleértve a horgászhely azonosítóját, a foglaló nevét és a foglalás kezdetét és végét. Az adminisztrátoroknak lehetőségük van a foglalások adatainak szerkesztésére és törlésére.

HTML kód:

* A "Foglalások" cím és egy vissza gomb jelenik meg a tetején.
* A foglalások adatai egy cikluson belül kártyákon jelennek meg, megjelenítve a foglaláshoz kapcsolódó információkat. Minden kártya tartalmaz gombokat a foglalás adatainak szerkesztéséhez és törléséhez.
* Az alján még egy vissza gomb található navigációhoz.

TypeScript kód:

* A ReservationsComponent osztály kezeli az adminisztrátori felület működését.
* Az ngOnInit() metódus lekéri a foglalások adatait, amikor a komponens inicializálódik.
* A getReservations() metódus lekéri a foglalások adatait a szerverről, majd frissíti a helyi tömböt.
* A updateReservationData() metódus előtölti a szerkesztési modalitás űrlapot a kiválasztott foglalás adataival.
* Az openModal() metódus programatikusan megnyitja a szerkesztési modalitás űrlapot.
* A deleteReservation() metódus elküldi a foglalás törlését kérő kérést és frissíti a foglalások listáját ennek megfelelően.
* Az updateReservation() metódus elküldi a frissítési kérést, majd megjelenít egy sikeres üzenetet.
* A showUpdateMessage() metódus az Angular Material Snackbar segítségével jeleníti meg az üzenetet a sikeres frissítésről.
* A getUser() és getUserByIndex() metódusok lekérik a felhasználó nevét az azonosító alapján és frissítik a megfelelő változókat.

Ábrajegyzék:

Képek:

* Háttér: <https://www.pexels.com/photo/green-trees-and-lake-photo-227416/>
* Ponty: <https://www.pngwing.com/en/free-png-hhidq>
* Harcsa: <https://pnghq.com/catfish-png-252-download-80870/>
* Galéria: <https://www.pngall.com/gallery-png/download/61203>
* Foglalás: <https://www.freepik.com/icon/online-booking_751663>
* Marker: <https://www.clipartbest.com/clipart-dc85yybgi>

Ezen kívül mindent saját fotósunkkal készítettünk, aki beleegyezett a képek használatába.

Felhasználói útmutató:

1. Látogasd meg a Kezdőlapot: Az új felhasználók a kezdőlapon kezdhetik a böngészést. Ide kattints az alapvető információk, képek és szolgáltatások megtekintéséhez.
2. További információk Rólunk oldalon: Ha érdekel, hogy többet megtudj a Dányi Tóról és a hozzá kapcsolódó horgásztelepről, kattints a "Rólunk" menüpontra. Itt részletes leírásokat és képeket találsz a helyről, a halakról és az idényekről.
3. Foglalás: Ha úgy döntesz, hogy szeretnél foglalni a horgászaton, kattints a "Foglalás" menüpontra. Itt találsz egy űrlapot, ahol megadhatod a foglalási információkat, és elküldheted a foglalást.
4. Bejelentkezés/Regisztráció: Ha már van fiókod, jelentkezz be a "Bejelentkezés" menüpontra kattintva. Ha még nem regisztráltál, kattints a "Regisztráció" menüpontra, és kövesd az ott található utasításokat a fiók létrehozásához.
5. Profilkezelés: Miután bejelentkeztél, a "Profil" menüpont alatt tudod frissíteni a profilodat, és kezelni a foglalásaidat.
6. Galéria: Ha érdekelnek mások képei és élményei a horgászaton, nézd meg a "Galéria" menüpontot. Itt feltölthetsz, szerkeszthetsz és törölhetsz képeket, és megoszthatod saját élményeidet másokkal.

Ismert problémák

1. Dátumellenőrzés hiánya foglalások módosításakor: Jelenleg nem ellenőrizzük a felhasználó által módosított foglalások dátumait. Ez azt jelenti, hogy a felhasználók szabadon változtathatják a foglalások időpontjait, még akkor is, ha azok ütköznek más foglalásokkal. Jövőbeli fejlesztéseink során ezt a hiányosságot javítani fogjuk, hogy megakadályozzuk a dátumütközéseket és a kényelmetlenségeket.
2. Képek késleltetett betöltése a foglalási űrlapon: Tapasztaltuk, hogy a képek betöltése némi késlekedéssel jár az első betöltés során a foglalási űrlapon. Bár jelenleg nincs ismert megoldás erre a problémára, a fejlesztőcsapatunk aktívan dolgozik azon, hogy megtaláljuk és kijavítsuk ezt a hibát a lehető leghamarabb, és biztosítsuk a zökkenőmentes felhasználói élményt.
3. Nincs lehetőség jelszóváltoztatásra: Jelenleg nincs lehetőség a felhasználóknak a jelszavuk megváltoztatására az alkalmazásban. Bár ez a funkció jelenleg nem elérhető, terveink között szerepel ennek bevezetése a közeljövőben. A felhasználók ezt a funkciót fogják tudni használni a jelszavuk biztonságosan frissítésére és kezelésére.

Jövőbeli tervek

1. Többnyelvűség támogatása: A jövőben tervezzük az oldal többnyelvűségének bevezetését, hogy a felhasználók angol és francia nyelven is hozzáférhessenek a tartalomhoz. Ez lehetővé teszi számukra, hogy kényelmesen böngésszenek és használják az oldalt az anyanyelvükön.
2. Sötét mód bevezetése: A sötét mód lehetőségének hozzáadása az oldalhoz egy olyan funkció, amely lehetővé teszi a felhasználók számára, hogy válasszanak a világos és a sötét témák között. Ez nemcsak a felhasználói élményt javítja, hanem segít csökkenteni a szemfáradtságot is.
3. Helyek értékelésének lehetősége: Terveink között szerepel egy értékelési rendszer bevezetése a felhasználók által foglalt helyek számára. Ez lehetővé teszi a felhasználók számára, hogy véleményt alkossanak és értékeljék a foglalt helyeket, amelyeket mások is láthatnak az oldalon.
4. Kommentek hozzáadása a galériához: A galériához kommentelési lehetőség bevezetése egy olyan funkció, amely lehetővé teszi a felhasználók számára, hogy megosszák gondolataikat és véleményüket a galériában található képekről. Ez növeli az interakciót és a közösségi aktivitást az oldalon.
5. Értesítési rendszer bevezetése: Terveink között szerepel egy értesítési rendszer bevezetése, amely értesíti a felhasználókat a foglalásaikról, a galériában történő tevékenységeikről és a halászengedélyük lejáratának közeledtéről. Ez segít a felhasználóknak naprakészen maradni és hatékonyabban kezelni az oldalon történő eseményeket.

Felhasznált irodalom:

Angular

* "Angular fejlesztés TypeScripttel" - Yakov Fain és Anton Moiseev
* "ng-book: A teljes könyv az Angular-ről" - Nate Murray, Ari Lerner, Felipe Coury és Carlos Taborda
* "Angular működés közben" - Jeremy Wilken
* "Pro Angular" - Adam Freeman
* "Angular alapok" - Mathieu Nayrolles

Laravel

* "Laravel: Induljunk neki!" - Matt Stauffer
* "Laravel: Code Bright" - Dayle Rees
* "Laravel tervezési minták és legjobb gyakorlatok" - Arda Kılıçdağı
* "Laravel 5 tanulása" - Antonio Leiva
* "Laravel alkalmazásfejlesztési receptkönyv" - Terry Matula

Köszönet:

Szeretnénk ezúton köszönetet mondani minden tanárunknak és mindenki másnak, aki segített ebben a projektben. Az önök támogatása, segítsége és tanácsai nélkül nem értük volna el ezt az eredményt. Köszönjük, hogy mindig mellettünk álltak, és segítették a fejlődésünket. Nagyon hálásak vagyunk a lelkesítő szavakért és a támogató légkörért, amelyben dolgozhattunk. Reméljük, hogy a munkánk eredményei elismerik az önök hozzájárulását és segítségét.