BikeApp

Fejlesztői dokumentáció

Készítette:

Klemán Dávid

Pesztránszki Dániel

Bevezetés

Miért ezt a témát választottuk?

A BikeApp program elkészítésével a célunk egy olyan mobilalkalmazás volt, amely egyszerre kínál széles körű túraválasztékot és lehetőséget új, felhasználói javaslatok beküldésére. A program kifejlesztése során arra törekedtünk, hogy:

- Testre szabható túrák: A felhasználók ne csak előre meghatározott útvonalak közül választhassanak, hanem saját igényeik szerint, megye, időtartam, nehézség és kerékpártípus alapján szűrhessenek.
- Közösségi részvétel: Lehetővé tettük, hogy minden regisztrált felhasználó új túrajavaslatokat küldhessen be, ezzel együtt formálva a közös túratérképet.
- Egyszerű és átlátható felület: Fontos volt számunkra a könnyű használhatóság, ezért a backend egyszerű, REST-szerű végpontokkal, világos dokumentációval támogatja a frontend fejlesztést és a tesztelést.

Használt technológiák és fejlesztői környezet

A BikeApp projektünk megvalósításához a következő, széles körben elterjedt keretrendszereket és eszközöket választottuk, hogy a fejlesztés gyors, áttekinthető és jól karbantartható legyen.

Frontend – Angular

- A felhasználói felületet az Angular (Google által karbantartott, komponens-alapú, TypeScript-re épülő keretrendszer) segítségével építettük fel.
- Előnyei:
    - Beépített modul- és routing-rendszer
    - Erős típusosság a TypeScript révén
    - Könnyen újrahasznosítható komponensek és szolgáltatások
- A kliensoldali kódot az Angular CLI-vel generáltuk és építettük, így a projekt felépítése követi az Angular stílusirányelveit.

Backend – Laravel

- A szerveroldali logikát a Laravel keretrendszer biztosítja, ami egy PHP-alapú, MVC architektúrára épülő, Eloquent ORM-et, beépített hitelesítést és migrációkezelést kínáló eszköz.
- Előnyei:
    - Gyors útvonal- és kontroller-definíció, REST API támogatás
    - Eloquent modellek és migrations a könnyű adatbázis-kezeléshez
    - Middleware és service-provider réteg a rugalmas bővíthetőségért

Adatbázis – MySQL (XAMPP fejlesztői környezetben)

- A projekt relációs adatbázis-kezelőjeként MySQL-t használtunk, amelyet helyileg a XAMPP csomag részeként futtattunk:
    - Apache webszerver
    - MySQL/MariaDB adatbázis-motor
    - PHP és phpMyAdmin a kényelmes táblakezeléshez
- A XAMPP-ot választottuk, mert otthoni fejlesztéshez könnyen telepíthető, minden szükséges komponenssel együtt érkezik, és jól izolált környezetet biztosít az adatbázis-migrációk és a futtatás teszteléséhez.

Frontend dokumentáció

Készítette: Pesztránszki Dániel

Komponensek

Admin component

HTML felépítés

A komponens egyetlen nagy .admin-container div-be ágyaz mindent:

1. **Fejléc**
   A tetején egy <header class="admin-header"> elem mutatja az alkalmazás nevét („BikeApp”), ami vizuálisan elkülönül a tartalomtól.
2. **Túra hozzáadása űrlap**
   A .admin-form szekcióban találjuk a túra felviteléhez szükséges mezőket. Egy <h2> cím vezeti be, majd az alábbi mezőkből áll:
   - **Megye**: sima szöveges input, kétirányú kötés newTour.tour_country-re.
   - **Túra ideje**: szöveges input időadat beviteléhez, kötött a newTour.tour_time változóhoz.
   - **Nehézség**: legördülő lista a három fokozattal („Könnyű”, „Közepes”, „Nehéz”), kötés newTour.difficulty-re.
   - **Bicikli típusa**: hasonló select, ahol a Gravel, Országúti és Mountainbike közül választhatunk.
   - **Útvonal neve**, **Túra leírása**, **Bicikli kép URL**, **Háttérkép URL** – ezek mindegyike [(ngModel)]-lel kötött input vagy textarea.
   - A szekció végén egy kétirányú kötéses gomb ((click)="addTour()"), ami csak akkor aktiválódik, ha minden mező érvényes és ki van töltve.
3. **Túrák listája**
   A .admin-list blokkon belül egy táblázat (<table>) mutatja az összes mentett túrát. A <thead> definiálja az oszlopcímeket (túra neve, ideje, nehézség stb.), a <tbody> pedig egy *ngFor="let tour of tours" direktívával iterál a tours tömbön. Minden sorban két műveletgomb található:
   - **Törlés**: (click)="deleteTour(tour)"
   - **Szerkesztés**: (click)="editTour(tour)"
4. **Regisztrált felhasználók**
   A .admin-users szekció első felében egy újabb táblázat sorolja fel az összes

felhasználót. A mezők: ID, név, felhasználónév, email és születési dátum. Minden sor végén ott a Törlés gomb, amelyről a deleteUser(user) hívást indítjuk.

5. **Beküldött túrajavaslatok**
   Ugyanebben a .admin-users kontextusban, a felhasználók alatt egy harmadik táblázat következik a javaslatokkal. Itt már *ngFor="let s of suggestions; let i = index" fut, és minden sor első cellája a sorszámot (i+1), majd felhasználó, túra adatai és a beküldés dátuma szerepel. A sor végén a deleteSuggestion(s) metódus hívódik meg.

6. **Navigáció**
   A komponens alján egy egyszerű <button (click)="goToLogin()">Vissza a bejelentkező oldalra</button> navigál vissza a belépő felületre.

Metódusok és működésük

A komponens az OnInit interfészt implementálja, így az ngOnInit() hívódik meg automatikusan:

- **ngOnInit()** Ebből indul minden adatbetöltés

Túrák kezelése

- **addTour()**
  Ellenőrzi, hogy a newTour objektum minden mezője nem üres-e. Ha igen, BaseService.post(this.newTour) hívással hozza létre az új túrát a szerveren, sikernél frissíti a listát és törli a formot, hibánál pedig alertet dob.

- **listTours()**
  A BaseService.get() segítségével tölti be a tours tömböt, így frissül a táblázat.

- **deleteTour(tour: any)**
  Meghívja BaseService.delete(tour)-t, majd visszahívja listTours()-t, hogy a táblázatban azonnal eltűnjön a törölt elem.

- **editTour(tour: any)**
  Ellenőrzi, hogy a túrának van-e id mezője, majd a Router.navigate(['/tour-edit'], { state: { tour } })-ral átirányítja a szerkesztő oldalra.

- **updateTour() és cancelEdit()**
  Ezek az inline szerkesztést támogatnák: az előbbi BaseService.put(this.editingTour)-t hív, a siker után frissít, majd editingTour = null-ra állít; utóbbi csak visszavonja az épp szerkesztett állapotot.

- **resetForm()**
  Újra inicializálja a newTour objektumot, hogy az űrlap üres legyen.

Felhasználók kezelése

- **loadUsers()**
  A UserService.getUsers()-tel tölti be a users listát.

- **deleteUser(user: any)**
  Megerősítést kér a confirm() segítségével, majd ha igent kap, UserService.deleteUser(user.id)-et hívva törli és újratölti a táblázatot.

Túrajavaslatok kezelése

- **loadSuggestions()**
  A TourSuggestionService.getSuggestions()-ből gyűjti be a suggestions tömböt.
- **deleteSuggestion(s: TourSuggestion)**
  Szintén megerősítést kér, ellenőrzi, hogy van-e id, majd a TourSuggestionService.deleteSuggestion(id)-nel eltávolítja a javaslatot és frissíti a listát.

Navigáció

- **goToLogin()**
  Egyszerű router.navigate(['/login']) hívás a belépő oldalra való visszatéréshez.

Login.component

HTML felépítése:
A teljes tartalom egy login-background osztályú div-ben helyezkedik el, amely mögött általában egy dekoratív háttérkép vagy szín található. Ennek tetején a login-header tartalmazza a „BikeApp” címet, ami statikusan jelenik meg.

A középső login-container blokkban egy <form> kezeli a bejelentkezést. A formnak van egy helyi referenciája (#loginForm="ngForm"), és az elküldéskor (ngSubmit) a komponens onSubmit() metódusa fut.

- Az első mező a felhasználónév vagy email, amely [(ngModel)]="credentials.login"-nel kötődik a komponens credentials objektumához, és required attribútummal biztosítjuk az érvényességet.
- A második mező a jelszó, amit egy „mutasd/elrejt” gomb is kiegészít: a gombbal a showPassword boolean értéket kapcsolgatjuk, így váltogatva a <input> mező típusát password és text között.
- A form alján a „Bejelentkezés” gomb (type="submit") csak akkor aktiválódik, ha minden mező ki van töltve.

A form alatt egy „Regisztráció” gomb viszi át a felhasználót a regisztrációs oldalra (goToRegister() metódus), ez szintén egy egyszerű Angular navigációs hívás.

Működés, metódusok

- **togglePasswordVisibility()**
  Egy egyszerű állapotváltó: ha a felhasználó a „Mutasd/Elrejt” gombra kattint, a showPassword változó értéke megfordul, és ennek hatására a jelszóbeviteli mező type attribútuma váltakozik password és text között. Ez a felhasználói élményt javítja anélkül, hogy újabb komponenseket kellene beemelni.
- **onSubmit(form: NgForm)**
  A bejelentkező űrlap elküldésekor először ellenőrizzük, hogy nincs-e „invalid” mező. Ha valamelyik mező üres vagy nem felel meg az required szabálynak, egy alert figyelmezteti a használót. Ha minden rendben, meghívjuk az AuthService.login(credentials)-et, ami egy Observable-t ad vissza.
  - Sikeres válasz esetén kiírunk egy „Sikeres bejelentkezés!” üzenetet, elmentjük a kapott api_token-t, a username-et és az is_admin flaget a localStorage-ba, majd a felhasználó jogosultságától függően átirányítjuk /admin vagy /search útvonalra.
  - Hiba esetén konzolra logoljuk a hiba részleteit, és egy alert-ben megjelenítjük az error response-ból kinyert hibaüzenetet vagy egy általános „Ismeretlen hiba” szöveget.
- **goToRegister()**
  Ha a felhasználó a regisztrációs gombra kattint, egyszerűen a Router.navigate(['/register']) hívással átirányítjuk őt a regisztrációs oldalra. Ez lehetővé teszi, hogy a belépési oldalról közvetlenül érje el a regisztrációs felületet.

Tour-detail.component

A TourDetailComponent a kiválasztott túra részletes adatlapját jeleníti meg: ha van átadott tour objektum, a háttérképet beállítja, kiírja a túra nevét, leírását és az ajánlott bicikli képét, valamint mutatja a bejelentkezett felhasználó nevét. Ha viszont nincs túra (például közvetlen URL-betöltéskor), egy „Nincs kiválasztott túra!” üzenet és egy vissza-gomb jelenik meg.

HTML felépítése:

- **Feltételes wrapper**
  A teljes tartalom egy <div class="tour-detail-wrapper" *ngIf="tour; else noTour"> elemen belül van. Ha tour nem definiált, az ng-template #noTour ugrik be.
- **Háttérkép**
  A wrapper ngStyle segítségével állítja be inline a background-image CSS-t a tour.backgr_pic URL-jével, így minden túrához saját háttér tartozik.
- **Fejléc**
  Egy egyszerű <header class="tour-detail-header"> tartalmaz egy „BikeApp” logót, ami visszahozza a főoldali márkaarculatot.

- **Bejelentkezett felhasználó**
  A div.user-info sorban jelenik meg a currentUsername értéke, így látható, ki van épp bejelentkezve.
- **Túra részletek**
    - Két oszlopba rendezve (.tour-content):
        1. **Túra leírás:** egy section-title címmel, alatta a tour.tour_description szövegével.
        2. **Ajánlott bicikli**: szintén section-title alatt egy <img> a tour.bike_pic URL-ből.
- **Navigációs gomb**
  A túra alján egy „Vissza a túraválasztás oldalra” gomb ((click)="goBack()"), amely visszavisz a /search oldalra.
- **Nincs túra template**
  Az ng-template #noTour csak akkor látszik, ha tour nincs: egy rövid üzenet és ugyanolyan vissza-gomb.

Főbb metódusok és logika

A Router segítségével az előző navigáció során átadott tour objektumot olvassa ki a state-ből. Ha valaki nem ezen az úton érkezik, tour undefined marad.

- **ngOnInit()** Betölti a korábban belépéskor eltárolt felhasználónevet a localStorage-ból, hogy a felületen megjeleníthesse a „Bejelentkezve” sort.
- **goBack()** Egyszerűen visszairányít a túraválasztó oldalra, ahol a felhasználó új túrát választhat ki.

Tour-edit.component

A TourEditComponent lehetővé teszi a kiválasztott túra adatainak szerkesztését: a háttérkép URL-je, a túra neve, ideje, nehézsége, biciklitípusa, leírása és kép URL-je mind módosítható, majd a „Mentés” gombra a változások visszaíródnak a szerverre. Ha nincs érvényes túra átadva, visszairányít az admin felületre.

HTML felépítése:

Külső wrapper
A teljes oldal egy <div class="tour-edit-page" *ngIf="tour"> elemen belül jelenik meg. Ha tour nincs definiálva (például közvetlen URL-hívásnál), nem renderelődik, és az ngOnInit-ben rögtön visszanavigálunk.

- **Dinamikus háttérkép**
  Az inline ngStyle-szel a tour.backgr_pic mező alapján állítjuk be a háttérképet, így azonnal látható a módosítás is.
- **Fix header**
  A lap tetején egy egyszerű BikeApp tartja a márkanevet.
- **Háttérkép URL szerkesztő**
  Egy önálló panelben (.background-url) lehet közvetlenül begépelni vagy módosítani a háttérkép URL-jét, majd azonnal vissza is tükröződik a háttérben.
- **Szerkesztő űrlap**
  A fehér .tour-edit-container belső részében:
    - **Túra neve**: a <h1>-en belüli input mező ([(ngModel)]="tour.route_name"), így a címet is azonnal szerkeszthetjük.
    - **Alapadatok**: egy rugalmas .edit-info blokkban három sorban:
        - Túra ideje (tour.tour_time) számként.
        - Nehézség (tour.difficulty) szövegként.
        - Bicikli típusa (tour.bike_type) szövegként.
    - **Leírás és bicikli kép**:
        - A leírás textarea-ban (tour.tour_description), ahol több sorban írható.
        - A bicikli kép URL-je szöveges inputként, és ha érvényes URL kerül a mezőbe, élőben jelenik meg az <img> is (*ngIf="tour.bike_pic").
- **Akciógombok**
  A form alján két gomb:
    - **Mentés** (type="submit") – elküldi a változásokat.
    - **Mégsem** (type="button") – visszalép az admin felületre (cancel()).

Főbb metódusok és logika

- **ngOnInit()**
  A komponens inicializáláskor a history.state.tour-ból veszi át a szerkesztendő túra objektumot. Ha nincs tour vagy hiányzik az id, alertet dob és visszanavigál az /admin oldalra:
- **updateTour()**
  A „Mentés” gombra fut le:
    1. Ellenőrzi, hogy a tour.id megvan-e, különben megszakítja.
    2. A tour_time mezőt stringgé konvertálja, hogy a backend is fogadja.

3. BaseService.put(this.tour) hívással elküldi a módosított objektumot.
    - Siker esetén konzolra logol, alertet dob és visszanavigál az admin felületre.
    - Hiba esetén kiírja a hibaüzenetet és alertben megjeleníti.

- **cancel()**
  Ha a felhasználó mégsem akar módosítani, egyszerű router.navigate(['/admin']) hívással visszaviszi az admin felületre, az aktuális változásokat elvetve.

Tour-search.component

A TourSearchComponent a BikeApp-ben a túrák szűrését és kiválasztását teszi lehetővé: a felhasználó sorrendben megadja a megyét, nehézséget, biciklitípust és túraidőt, a komponens pedig ezek alapján felépíti a megfelelő legördülő menüket, végül pedig a „Túra kiválasztása” gombbal a részletes oldalra navigál.

HTML felépítése:

A template egy tour-search-container div-be ágyazva jeleníti meg a felületet:

1. **Fejléc**
   Egy egyszerű <header class="tour-search-header">BikeApp</header> hívja elő az alkalmazás nevét.
2. **Bejelentkezett felhasználó**
   A div.user-info sorban mutatja a currentUsername értékét, így mindig látszik, ki van belépve.
3. **Szűrőablak** (.search-window)
    - **Megye**:

```
<select [(ngModel)]="selectedLocation" (ngModelChange)="onCountyChange()">
  <option value="">Válassz...</option>
  <option *ngFor="let loc of tourLocations" [value]="loc">{{ loc }}</option>
</select>
```

    - **Nehézség**: csak az előzőleg kiválasztott megye túráihoz tartozó szintekkel töltődik. Hasonló felépítésű select, onDifficultyChange() metódussal.
    - **Bicikli típusa**: az eddigi szűrők alapján generált bikeTypes tömbből épül.
    - **Túra ideje**: az összesített tourTimes listából választhat a felhasználó.

Minden dropdown kétirányúan kötött ngModel-lel, és a változáskor meghív egy-egy metódust, ami újrarendezi a következő lista elemeit.

4. **Akciógombok**
    - **Túra kiválasztása**: indítja a selectTour()-t, ami ellenőrzi a feltételeket, kiszűri a megfelelő túrát és a /tour-detail oldalra navigál.
    - **Vissza a bejelentkező oldalra**: a goBack() metódus hajtja végre.
    - **Új túra javaslat**: a goToSuggestion() viszi át a javaslat oldara.

Működés és főbb metódusok

1. **ngOnInit()**
    - Betölti a currentUsername-t a localStorage-ból.
    - BaseService.get() hívással lekéri a teljes tours tömböt a szerverről.
    - Ebből kigyűjti az egyedi megyéket:

this.tourLocations = Array.from(new Set(this.tours.map(t => t.tour_country)));

    - A többi szűrő-listát üresre állítja, amíg a felhasználó nem választ megyét.
2. **onCountyChange()**
   Ha megyeválasztás történik, a tours tömböt leszűri az adott tour_country alapján, és kiszámolja az ebből a körből jövő nehézségi szinteket. Emellett törli a későbbi dropdown-ok jelenlegi értékeit és listáit, hogy tiszta állapotból induljon a következő szintű szűrés.
3. **onDifficultyChange()**
   A megye és nehézség együttes szűrésével állítja elő a lehetséges bikeTypes értékeket, majd reseteli az alatta lévő mezőket.
4. **onBikeTypeChange()**
   A három kiválasztott szűrő alapján kiszámolja a rendelkezésre álló tourTimes listát, amelyből az óraszámot választhatja a felhasználó.
5. **selectTour()**
    - Ellenőrzi, hogy mind a négy szűrőből lett-e választás; különben egy alert figyelmeztet.
    - A tours tömböt normalizálva (trim().toLowerCase()) újraszűri a selected* értékekre, és ha talál egyezést, az elsőt kiválasztja.
    - Sikeres találatkor navigál a /tour-detail oldalra, átadva a kiválasztott túra objektumát a router state-jében.
    - Ha nincs találat, értesítést (alert) dob a felhasználónak.
6. **Navigációs metódusok**
    - **goBack()**: visszavisz ['/login']-re.
    - **goToSuggestion()**: átirányít ['/tour-suggestion']-re.

Register.component

The RegisterComponent egy önálló (standalone) Angular-komponens, amely a BikeApp felhasználóinak regisztrációs felületét valósítja meg. Importálja a CommonModule-t és a FormsModule-t, így az űrlap kezelését és a kétirányú adat-kötést (ngModel) is támogatja.

HTML felépítés:

1. **Háttér és fejléc**
   A teljes tartalom egy register-background osztályú div-ben kap helyet, benne a register-header blokkal, amelyben statikusan megjelenik a „BikeApp” márkanév.
2. **Regisztrációs űrlap** (register-container)
   - Egy <h1> vezeti be a „Regisztráció” címet.
   - A <form>-nak van egy helyi referenciája (#registerForm="ngForm"), és az elküldéskor (ngSubmit) a komponens onSubmit() metódusa fut.
   - **Mezők**:
     - Teljes név (name)
     - Felhasználónév (username)
     - Email cím (email, e-mail validációval)
     - Születési dátum (birthdate, date-pickerrrel)
     - Jelszó és jelszó megerősítése: mindkettőhöz van „Mutasd/Elrejt” gomb, ami a bemeneti mező típusát váltogatja password és text között.
   - Minden mező required attribútummal rendelkezik, hogy a form ne legyen elküldhető hiányos adat esetén.
3. **Akciógombok**
   - **Regisztráció** gomb (type="submit"), ami a validált adatokat elküldi.
   - **Vissza a kezdőoldalra** gomb, ami a goBack() metódussal visszavisz a bejelentkezéshez.

Működés, metódusok

- **togglePasswordVisibility() / toggleConfirmVisibility()**
  Ezek a metódusok felváltva kapcsolgatják a showPassword és showConfirmPassword boolean értékeket, amivel a jelszó és a megerősítő jelszó mezők típusát (password vs. text) váltogatják. Így a felhasználó bármikor láthatja vagy elrejtheti a beírt karaktereket.
- **onSubmit(form: NgForm)**

1. **Űrlap validálása**: ha bármelyik mező üres, figyelmeztető alert jelenik meg.
2. **Jelszó erősség ellenőrzése**: legalább 8 karakter és minimum 2 számjegy szükséges; ha nem teljesül, külön alert jelzi.
3. **Jelszavak egyezősége**: ha a két mező nem egyezik, alert, és nem történik továbbküldés.
4. **Regisztrációs objektum elkészítése**: a user modell mezőit egy registrationData objektumba csomagolja, ahol a backend által elvárt password_confirmation kulcsot is beállítja.
5. **API hívás**: az AuthService.register(registrationData) Observable-ét feliratkoztatva:
    - **Siker** esetén „Sikeres regisztráció!” alert, majd navigáció a /login oldalra.
    - **Hiba** esetén konzolra log és egy alert a válaszból kinyert hibaüzenettel vagy egy általános „Ismeretlen hiba” szöveggel.

- **goBack()**
  Egyszerű Router.navigate(['/login']) hívás, amivel a felhasználót visszairányítja a belépési oldalra.

Tour-suggestion.component

A TourSuggestionComponent lehetővé teszi, hogy a felhasználók új túrajavaslatokat küldjenek be a BikeApp-ba. Egy reaktív űrlapon adhatják meg a túra országát, idejét, nehézségét, biciklitípusát, útvonal nevét és leírását, majd a „Javaslat elküldése” gombra a rendszer továbbítja az adatokat a backendnek.

HTML felépítés:

- Az egész egy register-background konténerbe ágyazódik, amelyen belül a register-header mutatja a „BikeApp” címet.
- A div.user-info sorban jelenik meg az aktuális felhasználónév (currentUsername), hogy látható legyen, ki van bejelentkezve.
- A register-container alatt egy <form [formGroup]="form"> kezeli a reaktív űrlapot:
    - **Megye**, **Túra ideje**, **Nehézség**, **Bicikli fajtája**, **Útvonal neve** és **Túra leírása** mezők sorolódnak fel label-lel és a megfelelő input vagy select és textarea elemekkel, mindegyik formControlName-nel kötve a FormGroup-hoz.
    - Két gomb van alul: a „Javaslat elküldése” (type="submit") és a „Vissza a túraválasztáshoz” (type="button" (click)="goBack()").

Főbb metódusok és logika

- **Konstruktor**
  A FormBuilder segítségével hozza létre a form objektumot, mind a hat mezőre kötelező (Validators.required) szabályt beállítva. Importálja a TourSuggestionService-t és a Router-t.
- **ngOnInit()**
  A localStorage-ból olvassa ki a username-t, hogy a felhasználó adatai megjelenjenek a fejlécben.
- **onSubmit()**
  1. Ellenőrzi, hogy a reaktív űrlap valid-e; ha nem, figyelmeztető alert.
  2. submitting flag-et true-ra állít, hogy a felhasználó lássa, folyamatban van a beküldés.
  3. A form.value mezőit átnevezi a backend által elvárt kulcsokra (tour_country, tour_time, difficulty, bike_type, route_name, tour_description).
  4. Meghívja a sendSuggestion(payload) metódust:
     - **Siker** esetén „Köszönjük a javaslatot!” alert, a form reset(), submitting = false.
     - **Hiba** esetén konzolra logol, majd alertben megjeleníti az error response üzenetét és a státuszkódot, végül submitting = false.
- **goBack()**
  Egyszerű router.navigate(['/search']) hívás, visszaviszi a felhasználót a túraválasztó oldalra.

Szervizek

AuthService

Az AuthService felel a felhasználók regisztrációjáért és bejelentkezéséért: a HttpClient segítségével a háttér-API /register és /login végpontjait hívja. Mivel a szolgáltatás providedIn: 'root'-ként van beállítva, az alkalmazásban bárhonnan ugyanaz a példány használódik.

- **register(user: any): Observable<any>**
  A kapott user objektum — amely a név, felhasználónév, email, jelszó stb. mezőket tartalmazza — POST kéréssel elküldi a http://localhost:8000/api/register URL-re. A visszatérő Observable-re feliratkozva kezelhetjük a sikeres regisztrációt vagy az esetleges hibákat.
- **login(credentials: any): Observable<any>**
  A credentials (email/username és jelszó) POST kéréssel kerül a http://localhost:8000/api/login végpontra. A visszaérkező Observable-ből például a

kapott API-tokent és a felhasználói információkat olvashatjuk ki, majd tárolhatjuk az alkalmazásban.

BaseService

A BaseService egyszerű CRUD-műveleteket kínál a túrák kezeléséhez: lekérés, létrehozás, frissítés és törlés. A providedIn: 'root' miatt az egész alkalmazásban ugyanaz az egyetlen példány fut, így könnyen újra felhasználható bárhol.

- **get(): Observable<any[]>**
  GET kérést küld a http://localhost:8000/api/tours végpontra, és az összes elérhető túra adatát adja vissza egy Observable tömb formájában.
- **post(body: any): Observable<any>**
  A megadott body objektumot (új túra adatai) POST kéréssel elküldi a …/api/tours URL-re. A válaszban általában a létrehozott erőforrás visszakapott adatai szerepelnek.
- **put(body: any): Observable<any>**
  A body.id mező alapján összeállítja a …/api/tours/{id} URL-t, és PUT kéréssel elküldi a frissített túrainformációkat. Ez a metódus akkor hasznos, ha egy meglévő túra adatait akarjuk módosítani.
- **delete(body: any): Observable<any>**
  A body.id-t használva DELETE kérést küld a …/api/tours/{id} végpontra, ami a szerveren eltávolítja a megadott azonosítójú túrát.

TourSuggestionService

A TourSuggestionService arra szolgál, hogy a bejelentkezett felhasználók könnyen be tudjanak küldeni új túrajavaslatokat, le tudják kérni a korábban beküldött javaslataikat, és szükség esetén törölhessenek is javaslatokat. Mivel ezek a műveletek védett végpontokat érintenek, a szolgáltatás automatikusan hozzáfűzi a felhasználó Bearer tokenjét minden HTTP-kéréshez.

- **Jogosultság-kezelés**
  A getAuthHeaders() privát metódus kiolvassa a localStorage-ból az api_token értékét, majd létrehoz egy fejlécet, amely tartalmazza a Content-Type: application/json és az Authorization: Bearer <token> beállításokat. Így nem kell minden komponensben ismételni a token hozzáadását.
- **Új javaslat beküldése**
  A sendSuggestion(data) metódus POST kérést indít a …/api/tour-suggestions végpontra a felhasználó által kitöltött javaslat-adatokkal. Siker esetén visszakapjuk a létrehozott javaslat teljes objektumát, hogy azonnal frissíteni tudjuk a felületen (például kiüríthetjük az űrlapot).
- **Beküldött javaslatok lekérdezése**
  A getSuggestions() GET kérése lekéri az összes, a felhasználó által korábban elküldött túrajavaslatot. Ezt használjuk a listázó oldalnál, hogy a felhasználó áttekintse, melyeket küldte be.

- **Javaslat törlése**
  A deleteSuggestion(id) metódus a megadott azonosítójú javaslatot törli a szerverről egy DELETE kéréssel. Ezután a komponensben eltávolíthatjuk a listából, így a felhasználó azonnal látja a változást.

UserService

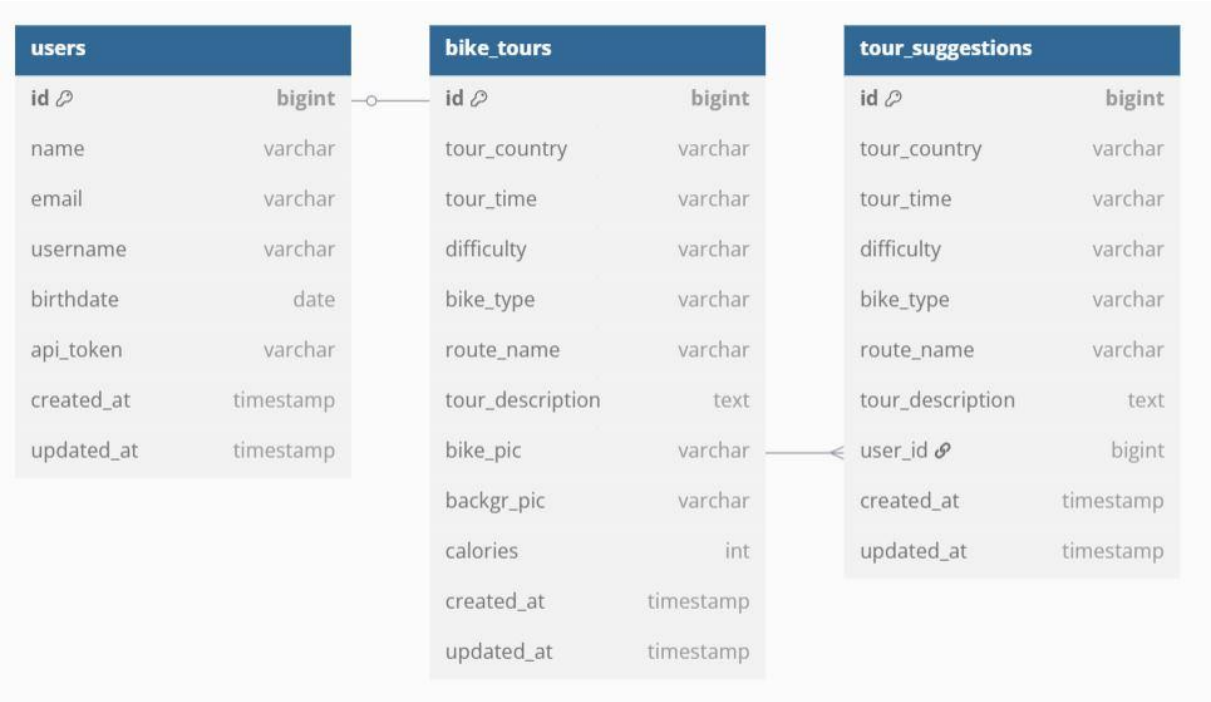
A UserService az alkalmazás felhasználóinak lekérdezéséért és törléséért felelős. A providedIn: 'root' beállítás miatt singletonként fut az egész Angular-alkalmazásban.

- **baseUrl**
  A szolgáltatás a http://localhost:8000/api/users végponton kommunikál a backenddel.
- **Felhasználók listázása**
  - **Metódus:** getUsers(): Observable<any>
  - **Működés:** GET kérést indít a …/api/users URL-re, és visszaadja az összes regisztrált felhasználó adatait tartalmazó Observable-t. Ezt használja például az AdminComponent loadUsers() metódusa az adatbetöltéshez.
- **Felhasználó törlése**
  - **Metódus:** deleteUser(id: number): Observable<any>
  - **Működés:** DELETE kérést küld a …/api/users/{id} végpontra, ahol az {id} a törölni kívánt felhasználó azonosítója. A komponensben a válaszra (subscribe) feliratkozva lehet frissíteni a felhasználói listát vagy megjeleníteni egy megerősítő üzenetet.

Backend dokumentáció

Készítette: Klemán Dávid

Adatbázis szerkezete



API végpontok

Összefoglaló táblázat

HTTP-módszer	Végpont	Kontroller@metódus	Leírás	Auth szükséges
GET	/api/countries	TourController@getUniqueCountries	Elérhető megyék listázása	Nem
POST	/api/register	AuthController@register	Új felhasználó regisztrációja	Nem
POST	/api/login	AuthController@login	Bejelentkezés	Nem
GET	/api/tours	TourController@index	Túrák lekérése (opcionális szűrőkkel)	Nem

GET	/api/tours/filters	TourController@getTourFilters	Szűrőfeltételek (megye, nehézség, stb.)	Nem
POST	/api/tours	TourController@store	Új túra létrehozása	Nem
PUT	/api/tours/{id}	TourController@update	Létező túra frissítése	Nem
DELETE	/api/tours/{id}	TourController@destroy	Túra törlése	Nem
GET	/api/users	UserController@index	Felhasználók listázása	Nem
DELETE	/api/users/{id}	UserController@destroy	Felhasználó törlése	Nem
POST	/api/tour-suggestions	TourSuggestionController@store	Túraötlet beküldése	Igen
GET	/api/tour-suggestions	TourSuggestionController@index	Küldött túraötletek listázása	Igen
DELETE	/api/tour-suggestions/{id}	TourSuggestionController@destroy	Túraötlet törlése	Igen

Migration file-ok:

Fájl név:	**Leírás:**
0001_01_01_000000_create_users_table.php	users tábla létrehozása (id, name, email, password, remember_token, timestamps stb.)
0001_01_01_000001_create_cache_table.php	Laravel cache táblázat (cache driver adatainak tárolása)
0001_01_01_000002_create_jobs_table.php	Laravel queue jobs tábla (queued jobok sorban tartása)
2025_03_28_182055_create_bike_tours_table.php	bike_tours tábla létrehozása – itt tároljuk a túrák alapadatait
2025_04_02_172656_add_description_and_pics_to_bike_tours_table.php	Leírás- (description) és képek- (pics) mezők hozzáadása a bike_tours táblához
2025_04_10_082456_add_api_token_and_birthdate_to_users_table.php	users tábla bővítése: api_token és birthdate mezők hozzáadása
2025_04_10_091409_add_username_to_users_table.php	users tábla bővítése: username mező hozzáadása
2025_04_21_190457_create_tour_suggestions_table.php	tour_suggestions tábla létrehozása (cím, leírás, megye, user_id, timestamps)

Táblák

Users table

A 0001_01_01_000000_create_users_table.php migráció egyszerre három táblát hoz létre: a users-t, a password_reset_tokens-t és a sessions-t.

Oszlop neve	Adattípus	Alapértelmezett	Leírás
id	unsigned bigInteger (PK)	not null	Elsődleges kulcs, auto-increment
name	varchar(255)	not null	Felhasználó teljes neve
email	varchar(255)	not null	Felhasználó e-mail címe, egyedi
email_verified_at	timestamp	null	E-mail ellenőrzés időpontja
password	varchar(255)	not null	Hash-elt jelszó
remember_token	varchar(100)	null	remember me token
created_at	timestamp	null	Rekord létrehozásának időpontja
updated_at	timestamp	null	Rekord utolsó frissítésének időpontja

password_reset_tokens tábla

Oszlop neve	Adattípus	Alapértelmezett	Leírás
email	varchar(255) (PK)	not null	A reset kérelemhez tartozó e-mail cím
token	varchar(255)	not null	Reset token
created_at	timestamp	null	Token létrehozásának időpontja

sessions table

Oszlop neve	Adattípus	Alapértelmezett	Leírás
id	varchar(255) (PK)	not null	Szekció azonosító
user_id	unsigned bigInteger	null	Hivatkozás a users.id-re
ip_address	varchar(45)	null	Felhasználó IP címe
user_agent	text	null	Böngésző/User agent információ
payload	longText	not null	A session tartalma binárisan serializálva
last_activity	integer	not null	Utolsó aktivitás időpontja (UNIX timestamp)

bike_tours table

Oszlop neve	Adattípus	Alapértelmezett	Leírás
id	unsigned bigInteger (PK)	auto-increment	Elsődleges kulcs
tour_country	varchar(255)	not null	Túra helyszíne (megye, ország, város stb.)
tour_time	integer	not null	Túra becsült időtartama órában
difficulty	varchar(255)	not null	Nehézség (pl. „könnyű”, „közepes”, „nehéz”)
bike_type	varchar(255)	not null	Ajánlott kerékpártípus
route_name	varchar(255)	not null	Az útvonal elnevezése
calories	integer	not null	Becsült energiafelhasználás kalóriában
created_at	timestamp (nullable)	null	Rekord létrehozásának időpontja
updated_at	timestamp (nullable)	null	Rekord utolsó frissítésének időpontja

add_description_and_pics_to_bike_tours_table

Ez a migráció három új mezőt ad a már meglévő bike_tours táblához:

Oszlop neve	Adattípus	Alapértelmezett	Leírás
tour_description	text	Null	A túra részletes leírása
bike_pic	string(255)	Null	A túra fő képének fájlneve vagy URL
backgr_pic	string(255)	Null	Háttérkép fájlneve vagy URL

add_api_token_and_birthdate_to _users_table

Ez a migráció két új mezőt ad a users táblához:

Oszlop neve	Adattípus	Alapértelmezett	Leírás
birthdate	date	not null	Felhasználó születési dátuma
api_token	string(80)	null	API-kliens hitelesítéséhez használt token

A birthdate mező nem enged NULL értéket, és date formátumú adatot tárol.

Az api_token mező VARCHAR(80), alapértelmezett NULL, egyedi index-szel (UNIQUE), hogy egyszerre csak egy token legyen aktív felhasználónként.

add_username_to_users_table

Ez a migráció egyetlen új mezőt ad hozzá a users táblához:

Oszlop neve	Adattípus	Alapértelmezett	Leírás
username	string(255)	not null	Felhasználó egyedi belépési név

A username mező VARCHAR(255) típusú, nem enged NULL-t, és UNIQUE index garantálja, hogy minden felhasználónév csak egyszer szerepeljen.

A mező elhelyezése az email mező után történik.

create_tour_suggestions_table

Ez a migráció létrehozza a tour_suggestions táblát a felhasználók által beküldött túraötletek tárolására.

Oszlop neve	Adattípus	Alapértelmezett	Leírás
id	unsigned bigInteger (PK)	auto-increment	Elsődleges kulcs
tour_country	varchar(255)	not null	Javasolt túra helyszíne (megye vagy város)
tour_time	integer	not null	A javasolt túra becsült időtartama órában
difficulty	varchar(255)	not null	Nehézség (pl. „könnyű”, „közepes”, „nehéz”)
bike_type	varchar(255)	not null	Javasolt kerékpártípus
route_name	varchar(255)	not null	Az útvonal elnevezése
calories	integer	not null	Becsült energiafelhasználás kalóriában
tour_description	text	not null	Részletes leírás a javasolt túráról
created_at	timestamp (nullable)	null	Rekord létrehozásának időpontja
updated_at	timestamp (nullable)	null	Rekord utolsó frissítésének időpontja

Modellek

BikeTour.php

A BikeTour modell a túra-adatok olvasásáért és írásáért felel a rendszerben:

- **Tábla leképezése**
  A modell kifejezetten a bike_tours táblát használja (protected $table = 'bike_tours'). Így ha a kódban például BikeTour::all()-t hívsz, a bike_tours tábla összes sorát kapod vissza.
- **Időbélyegek (timestamps)**
  Mivel a migrációban a timestamps() oszlopokat hoztuk létre, de a modellben public $timestamps = false van beállítva, a frissítések és beszúrások során nem kezeli automatikusan a created_at és updated_at mezőket. Ha szeretnéd, hogy ezek is automatikusan frissüljenek, érdemes public $timestamps = true-re állítani.
- **Kitölthető (fillable) mezők**
  A $fillable tömbben soroltad fel a tábla összes olyan mezőjét (név, időtartam, nehézség, képek, kalória stb.), amelyeket tömegével (mass assignment) szeretnél létrehozni vagy frissíteni. Például:
- **Gyári adatok és HasFactory**
  A use HasFactory; lehetővé teszi, hogy a modelhez tartozó factory-k segítségével gyorsan generálj teszt- vagy demóadatokat.
- **Nincsenek relációk**
  A BikeTour modell nem hivatkozik más entitásokra (pl. nincs belongsTo vagy hasMany), mert a túrainformációk statikus listaként szolgálnak a felhasználók számára.

TourSuggestion.php

A TourSuggestion modell a felhasználók által beküldött túraötletek kezeléséért felel:

- **Tábla leképezése**
  Mivel nem adtál meg protected $table tulajdonságot, alapból a tour_suggestions tábla lesz hozzárendelve a modellhez.
- **Időbélyegek (timestamps)**
  A modell nem állítja át a public $timestamps értékét, így a Laravel automatikusan kezeli a created_at és updated_at mezőket.
- **Kitölthető mezők ($fillable):**

'tour_country',

'tour_time',

'difficulty',

'bike_type',

'route_name',

'tour_description',

'user_id',

Ezek azok az oszlopok, amelyeket tömeges rendelés (mass assignment) során egyszerűen át lehet adni a create() vagy update() hívásoknál.

- **Relációk:**

return $this->belongsTo(User::class);

Ez biztosítja, hogy egy TourSuggestion példány ->user kifejezéssel visszaadja azt a User objektumot, aki az ötletet beküldte (a user_id mező alapján).

- **Gyári adatok**
  A use HasFactory; sor lehetővé teszi, hogy a hozzá tartozó factory-val gyorsan generálj teszt- vagy demóadatokat (például TourSuggestion::factory()->count(5)->create()).

User.php

A User modell felelős az autentikációért, a jogosultságok kezeléséért és a felhasználói profiladatok tárolásáért.

A User modell az alkalmazás hitelesítési és jogosultságkezelési magját képezi: az Illuminate\Foundation\Auth\User osztályból öröklődik, és a HasFactory mellett a Notifiable trait-tel támogatja az értesítések (pl. jelszó-visszaállítási e-mail) küldését. A $fillable tömbben szerepelnek azok a mezők – mint a name, email, username, password, birthdate és api_token – , amelyeket tömeges hozzárendeléssel is biztonságosan be tudunk állítani, míg a $hidden segítségével a jelszó és a remember_token nem kerül ki a JSON-válaszokba. A $casts beállítások automatikusan Carbon példánnyá alakítják az email_verified_at mezőt, illetve gondoskodnak arról, hogy a password mező beállításakor a Laravel beépített hash-elési mechanizmusa fusson le. Javasolt kiegészítésként érdemes lehet a suggestions() relációt is definiálni, hogy egy felhasználó könnyedén elérhesse a neki tulajdonított TourSuggestion rekordokat.

Kontrollerek

AuthController.php

Az AuthController felelős a felhasználók regisztrációjáért és bejelentkezéséért. A register metódus először Validator segítségével ellenőrzi a beérkező mezőket (név, felhasználónév, e-mail, születési dátum és jelszó), majd manuálisan is biztosítja a felhasználónév és e-mail egyediségét, továbbá Carbon segítségével kiszámolja a korhatárt (min. 18 év). Sikertelen validáció esetén 422-es hibakódot ad vissza, egyébként generál egy 80 karakteres API-tokent (Str::random), létrehozza a User rekordot (jelszó hashelve), majd két értesítő e-mailt küld – az új felhasználónak és az adminnak. Végül egy 201 Created választ küld vissza a felhasználó adataival és az API-tokennel.

A login metódus login mezőként elfogadja az e-mailt vagy felhasználónevet, majd a jelszó ellenőrzése után (Hash::check) szintén API-token-t generál, ha még nincs, és beállít egy is_admin jelzőt attól függően, hogy az e-mail megegyezik-e a mail.admin_address értékével. Hibás hitelesítésnél 401-es választ küld. Mindkét metódus middleware nélkül, publikus végpontként érhető el.

TourController.php

A **TourController** kezeli a statikus túraadatokat (BikeTour), azaz a túrák listázását, létrehozását, frissítését és törlését, valamint a frontend számára szükséges szűrőértékek szolgáltatását.

- A getTourFilters metódus adatbázis-lekérdezésekkel gyűjti a legördülő menükbe a distinct megye-, időtartam-, nehézség- és kerékpártípus-értékeket, és JSON-ben adja vissza.
- Az index metódus dinamikusan építi fel az Eloquent query-t a kérés paraméterei alapján (tour_country, tour_time, difficulty, bike_type), majd visszaküldi a szűrt túralistát.
- A store és update metódusok $request->validate() segítségével ellenőrzik a kötelező mezőket (tour_country, tour_time, difficulty, bike_type, route_name), opcionálisan a leírást és képeket, és alapértelmezés szerint calories = 0 értékkel hoznak létre/frissítenek rekordot.
- A destroy metódus a megadott azonosító alapján törli a túrát, és üzenettel visszajelzi a sikeres törlést.

Minden metódus nyílt, auth middleware-t nem használ, és response()->json()-t alkalmaz a válaszokhoz.

TourSuggestionController.php

A TourSuggestionController az autentikált felhasználók által beküldött túraötletek kezelésére szolgál, auth:api middleware-rel védve.

- A store metódus egy TourSuggestionRequest-tel biztosítja a bemenet érvényességét (kötelező mezők: tour_country, tour_time, difficulty, bike_type, route_name, tour_description), majd hozzáadja a bejelentkezett felhasználó user_id-jét és elmenti az ötletet, végül 201-es státusszal visszaküldi az új TourSuggestion objektumot.
- Az index metódus adminisztratív céllal with('user')-rel betölti a javaslatokat felhasználói adatokkal, időrendi sorrendben, és JSON tömbben adja vissza őket.
- A destroy metódus ellenőrzi, hogy létezik-e a javaslat; ha nem, 404-es választ küld, különben törli a rekordot és 204 No Content státuszt ad vissza.

UserController.php

A UserController egyszerű felhasználókezelő:

- Az index metódus az összes regisztrált User rekordot visszaadja JSON tömbként (akár paginációval is bővíthető a nagy adatmennyiség kezeléséhez).
- A destroy metódus a findOrFail segítségével lekéri a megadott azonosítójú felhasználót, törli az adatbázisból, majd visszaküld egy megerősítő üzenetet.

Jelenleg nem használ auth middleware-t, így érdemes lehet később admin jogosultságokhoz kötni ezeket a műveleteket**.**

Mailek

AdminUserRegistered.php

Az osztály a regisztráció után automatikusan e-mailt küld az adminisztrátori címre, hogy értesítse az új felhasználóról. A levél tárgya “Admin User Registered”, a tartalmát pedig egy Markdown-alapú sablon (emails.admin_user_registered) adja – így könnyen karbantartható és átlátható formában jelennek meg benne a user adatai. Jelenleg nincs benne konstruktorparaméter, de érdemes lehet átadni neki a regisztrált User objektumot, hogy a sablonban konkrét névvel, e-maillel és dátummal személyre szabhassuk az üzenetet. Ha nagy forgalmú a rendszered, a ShouldQueue interfész implementálásával pedig aszinkron módon, háttérfolyamatban is kiküldheted az értesítéseket.

NewUserRegisteredMail.php

A NewUserRegisteredMail felel azért, hogy új felhasználó regisztrációja után a rendszer azonnal e-mail értesítést küldjön az adminnak: a konstruktorban átveszi a User objektumot, a build() metódus pedig dinamikusan felépíti a tárgysort („Új felhasználó regisztrált: {username}”), majd egy Markdown-sablont (emails.registration.admin_notification) használ a levél törzséhez, melybe a regisztrált felhasználó nevét, felhasználónevét és e-mail címét továbbítja. Ha később aszinkron küldésre vagy további adatok átadására van szükség, egyszerűen kiterjeszthető a ShouldQueue implementálásával vagy további változókkal a with() tömbben.

RegistrationConfirmationMail.php

A RegistrationConfirmationMail osztály feladata, hogy a sikeres regisztrációt követően automatikusan üdvözlő e-mailt küldjön a frissen regisztrált felhasználónak. A konstruktorban átveszi a User objektumot, így a felhasználó neve, felhasználóneve és e-mail címe könnyen elérhető a sablonban. A build() metódus dinamikus tárgysort állít össze – például „Üdvözlünk a BikeApp-ban, Dávid!” –, majd a emails.registration.confirmation Markdown nézetet tölti fel a with() tömbben átadott változókkal (name, username, email). Így a levél tartalma személyre szabott, és a Markdown-sablon alkalmazása biztosítja a konszistent formázást mind HTML, mind sima szöveges verzióban.

Providerek

RouteServiceProvider

- **Szerep:** Az alkalmazás útvonalainak (routes) csoportosítása és betöltése a megfelelő middleware-ek és prefixek alkalmazásával.

1. **Regisztráció(register()):**
   A Laravel alapértelmezett register() metódusa itt öröklődik a ServiceProvider osztályból, és nem tartalmaz egyedi logikát, de ha később például custom route bindingeket vagy feltételekhez kötött route-regisztrációt szeretnél, ezt a metódust is módosíthatod.
2. **Bootstrap(boot()):**
   A boot() metódus felelős azért, hogy az alkalmazás indításakor betöltse az útvonalfájlokat:
   - **API-útvonalak**
     Az /api előtag és az api middleware-csoport (token-hitelesítés, throttle, JSON-formátum) automatikusan minden routes/api.php-ben definiált végponthoz hozzáadódik.
   - **Web-útvonalak**
     A routes/web.php fájlban található útvonalakat a web middleware-csoport (session, cookie, CSRF-védelem) kezeli, előtag nélkül.

- **Automatikus betöltés:**
  A config/app.php providers tömbjében szereplő App\Providers\RouteServiceProvider::class biztosítja, hogy a register() és boot() metódusok minden alkalmazásindításkor futnak.

Ez a Service Provider ad keretet az API- és webes útvonalak kezelésének, és teszi lehetővé, hogy minden route egységes beállításokkal, könnyen karbantartható módon kerüljön betöltésre.

AppServiceProvider

- **Szerep:** Ez az egyik alapértelmezett Laravel service provider, amely az alkalmazás életciklusának korai fázisában fut le.

1. **Regisztráció (register()):**
   Itt lehet regisztrálni a szolgáltatásokat a Laravel konténerébe. Például interfész–implementáció párokat, egyedi gyári függvényeket vagy külső csomag konfigurációt kötünk ide. Mivel a metódus jelenleg üres, egyelőre nem történik semmilyen szolgáltatásregisztráció.

2. **Bootstrap (boot()):**
   Ebben a metódusban történik az alkalmazásszintű inicializáció: esemény-hallgatók (event listener), model-observer-ek, egyedi Blade-direktívák vagy lokális konfigurációk indíthatók el itt. Jelenleg szintén üres, de bármikor bővíthető, ha a projekt igényel valamilyen globális beállítást.

- **Automatikus betöltés:**
  A config/app.php providers tömbjében szerepel, így a Laravel automatikusan meghívja a register() és boot() metódusokat minden alkalmazásindításkor.

Statikus tesztelés

Regisztráció

Elvárt funkció: a Login oldalon a Regisztráció gombra kattintva az oldal a regisztrációs felületre navigál.

Működés: OK

Elvárt funkció: Ha a regisztrációs oldalon nincs kitöltve minden mező a program felugró ablakban közli a felhasználóval, hogy tölts ki minden mezőt.

Működés: OK

Elvárt funkció: Ha a jelszóra beállított kritériumok nem teljesülnek a program figyelmezteti a felhasználót.

Működés: OK

Elvárt funkció: A jelszó és a jelszó megerősítése mezőben ugyan azoknak az értékeknek kell lennie, ha ez nem teljesül a program felugró ablakban figyelmeztet, hogy a jelszavak nem egyeznek.

Működés: OK

Elvárt funkció: A jelszó és a megerősített jelszó pontként mutatása átváltható betükre, számokra, és fordítva

Működés: OK

Elvárt funkció: Ha minden mezőt kitöltök és rákattintok a regisztráció gombra, a program kiírja hogy sikeres regisztráció.

Működés: OK

Elvárt funkció: A sikeres regisztráció után a program automatikusan a Login oldalra navigál vissza.

Működés: OK

Elvárt funkció: Ha nincs kitöltve minden mező, és nem szeretnénk regisztrálni akkor a „Vissaz a kezdőoldalra” gombbal a Login oldalra navigál vissza.

Működés: OK

Elvárt funkció: Ha a regisztráció sikeres, a felhasználó emailt a beregisztrált adatokkal.

Működés: OK

Elvárt funkció: Ha a regisztráció sikeres, az admin emailt kap az új beregisztrált falhasználó adatairól.

Működés: OK

Elvárt funkció: Ha már korábban regisztrált felhasználónévvel vagy email címmel próbálunk regisztrálni a program kiírja hogy ezek valamelyike foglalt.

Működés: OK

Bejelentkezés

Elvárt funkció: A beregisztrált felhasználónévvel és a hozzá tartozó jelszóval be tudunk jelentkezni, a program kiírja, hogy sikeres bejelentkezés.

Működés: OK

Elvárt funkció: A beregisztrált email címmel és a hozzá tartozó jelszóval be tudunk jelentkezni, a program kiírja, hogy sikeres bejelentkezés.

Működés: OK

Elvárt funkció: Ha a felhasználónév vagy a hozzá tartozó jelszó eltérő a program erre figyelmeztet, és nem enged belépni.

Működés: OK

Elvárt funkció: Ha az email cím vagy a hozzá tartozó jelszó eltérő a program erre figyelmeztet, és nem enged belépni.

Működés: OK

Elvárt funkció: Sikeres bejelentkezés után a program a Túraválasztó oldalra navigál

Működés OK

Túra választó oldal

Elvárt funkció: Az oldal jobb felső sarkában a bejelentkezett felhasználó user neve látható.

Működés: OK

Elvárt funkció: Ha nem választunk mind a 4 opcióból és úgy próbálunk túrát választani akkor erre a program figyelmeztet.

Működés: OK

Elvárt funkció: A 4 opció közül először a megye aktív, aztán ezt kiválasztva, nehézség, bicikli típusa és az idő.

Működés: OK

Elvárt funkció: A „Túra kiválasztása” gombra kattintva a 4 opciónak megfelelő túra oldalára navigál az oldal.

Működés: OK

Elvárt funkció: A „Vissza a bejelentkező oldalra” gombra kattintva a program a Login oldalra navigál.

Működés: OK

Elvárt funkció: Az „Új túra javaslat” gombra kattintva a program a túra javaslat készítő oldalra navigál.

Működés: OK

Kiválasztott túra oldal

Elvárt funkció: Az oldal jobb felső sarkában a bejelentkezett felhasználó user neve látható

Működés: OK

Elvárt funkció: Az oldal a kiválasztott túra adatait tölti be.

Működés: OK

Elvárt funkció: A „Vissza a túraválasztás oldalra” gombra kattintva a túra választó oldalra navigál a program.

Túra javaslat készítő oldal

Elvárt funkció: A javaslat elküldéséhez az összes mezőt ki kell küldeni, erre a program figyelmeztet is.

Működés: OK

Elvárt funkció: A nehézség és a bicikli fajtája opcióknál a lenyíló menük működnek

Működés: OK

Elvárt funkció: A „Javaslat elküldése” gombra kattintva a program tudatja a felhasználóval hogy a javaslatküldés sikeres volt.

Működés: OK

Elvárt funkció: Az admin oldalon a túra javaslatok között az elküldött javaslat megjelenik

Működés: OK

Elvárt funkció: A „Vissza a túra választó oldalra” gombra kattintva a program a túra választó oldalra navigál

Működés: OK

Admin oldal

Elvárt funkció: A login oldalon az előzetesen beállított admin email címmel és jelszóval lehet beregisztrálni, ha a felhasználónév és a jelszó nem egyezik, a program küzli hogy a felhasználónév vagy a jelszó érvénytelen.

Működés: OK

Elvárt funkció: A megfelelő admin felhasználónévvel és jelszóval a „Bejelentkezés” gombra kattintva a program kiírja, hogy sikeres bejelentkezés. Az OK gombra kattintva a program az admin oldalra navigál.

Működés: OK

Elvárt funkció: A megfelelő admin email címmel és jelszóval a „Bejelentkezés” gombra kattintva a program kiírja, hogy sikeres bejelentkezés. Az OK gombra kattintva a program az admin oldalra navigál.

Működés: OK

Elvárt funkció: Az oldalon látható egy Túra hozzáadása ablak, amivel új túrát tudunk hozzáadni az túra adatbázishoz.

Működés: OK

Elvárt funkció: Az új túra hozzáadásához minden mezőt ki kell tölteni, erre a program figyelmeztet is.

Működés: OK

Elvárt funkció: A nehézség és a bicikli típusa kiválasztásánál a legördülő menük vannak.

Működés: OK

Elvárt funkció: Ha minden mező ki van töltve az új tóra hozzáadásánál, akkor a „Hozzáadás” gombra kattintva a program kiírja, hogy „Sikeresen hozzáadtad a túrát” Az OK gomb lenyomása utána túra bekerül a túrák listájába.

Működés: OK

Elvárt funkció: Az admin oldalon ki van listázva az összes olyan túra ami szerepel az adatbázisban.

Működés: OK

Elvárt funkció: Minden kilistázott túrát lehet szerkeszteni a műveletek között található „Szerkesztés” gombbal. A gombra kattintva egy Túra szerkesztése oldalra jutunk ahol tetszőlegesen lehet változtatni a túra adatait.

Működés: OK

Elvárt funkció: A túra learás kivételével egyik mező sem maradhat üresen, különben nem tudjuk elmenteni a túra módosításokat. A program hibát írni ki és kéri, hogy a mezők legyenek kitöltve

Működés: OK

Elvárt funkció: Ha elvégeztük a szükséges módosításokat és minden mező ki van töltve akkor a „Mentés” gombra kattintva a mentés megtörténik, a program pedig kiírja, hogy „Sikeres frissítés”. Az OK gombra kattintva visszatérünk az admin oldalra.

Működés: OK

Elvárt funkció: Az admin oldalra visszatérve, már a módosított túra adatai látszódnak a listában.

Működés: OK

Elvárt funkció: A listában található túrák mellett található egy „Törlés” gomb, amellyel listából törölhetők a túrák. A gombra rákattintva a program megkérdezi hogy biztos törölni szeretnéd-e a túrát? Az OK gombra kattintva a túra törlődik és eltűnik a listából.

Működés: OK

Elvárt funkció: Az admin oldalon ki van listázva az aktuális felhasználók listája, regisztrációs adataikkal együtt.

Működés: OK

Elvárt funkció: A regisztrált felhasználók listájából törölni lehet a felhasználókat a sor végén található törlés gombbal. A „Törlés” gombra kattintva a program megkérdezi, hogy „Biztosan törlöd a kiválasztott felhasználót?” Az OK gombra kattintva a program kiírja, hogy a „A felhasználó sikeresen törölve” A kitörölt felhasználó eltűnik a listából.

Működés: OK

Elvárt funkció: Az admin oldalon ki vannak listázva beküldött túrajavaslatok, látható, hogy melyik felhasználó küldte be a javaslatot, és a túra adatai is.

Működés: OK

Elvárt funkció: A túra javaslatokat a sorok végén található „Törlés” gombbal ki lehet törölni. A gombra kattintva a program megkérdezi, hogy „Biztosan törlöd a kijelölt túra javaslatot?” Az OK gombra kattintva a program kiírja, hogy a „Javaslat sikeresen törölve.” Az OK gombra kattintva a javaslat eltűnik a listából.

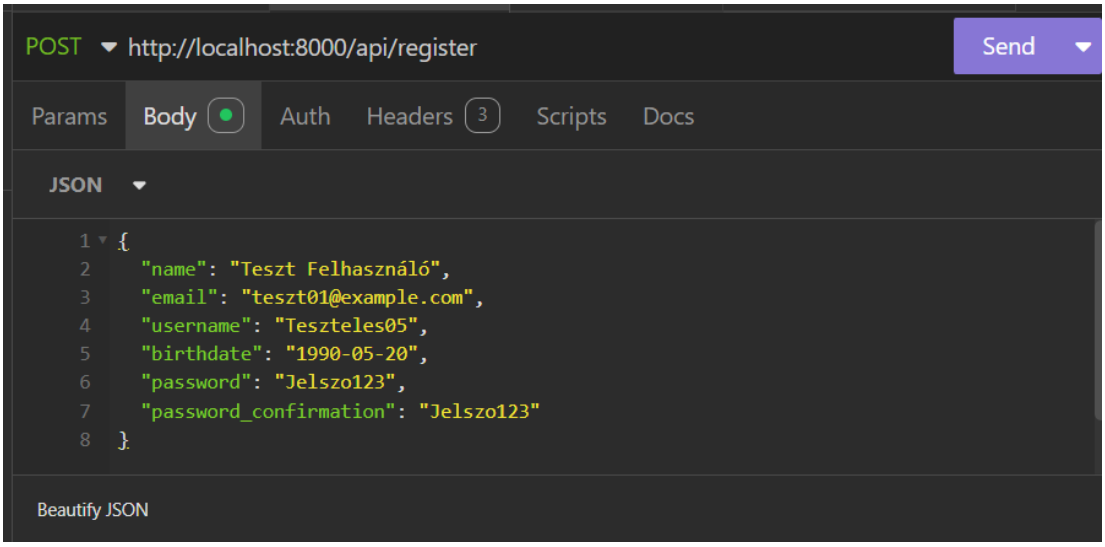
Működés: OK

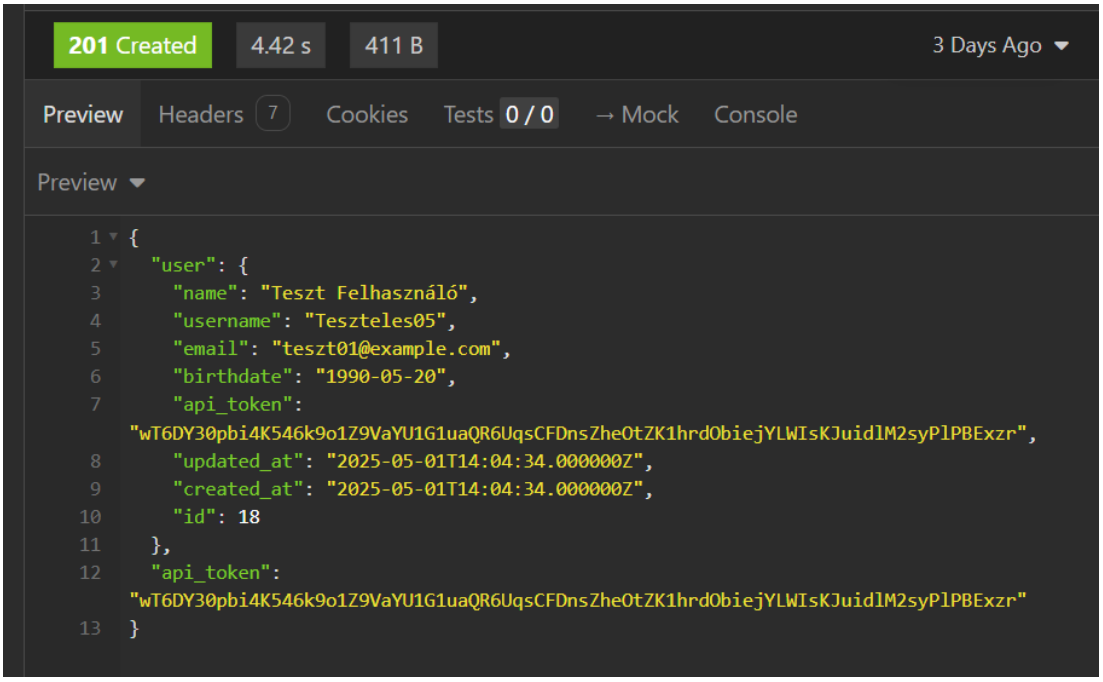
Elvárt funkció: Az oldal alján található „Vissza a bejelentkező oldalra” kattintva a program a Login oldalra navigál, automatikusan kijelentkezve az admin oldalról.

Működés: OK

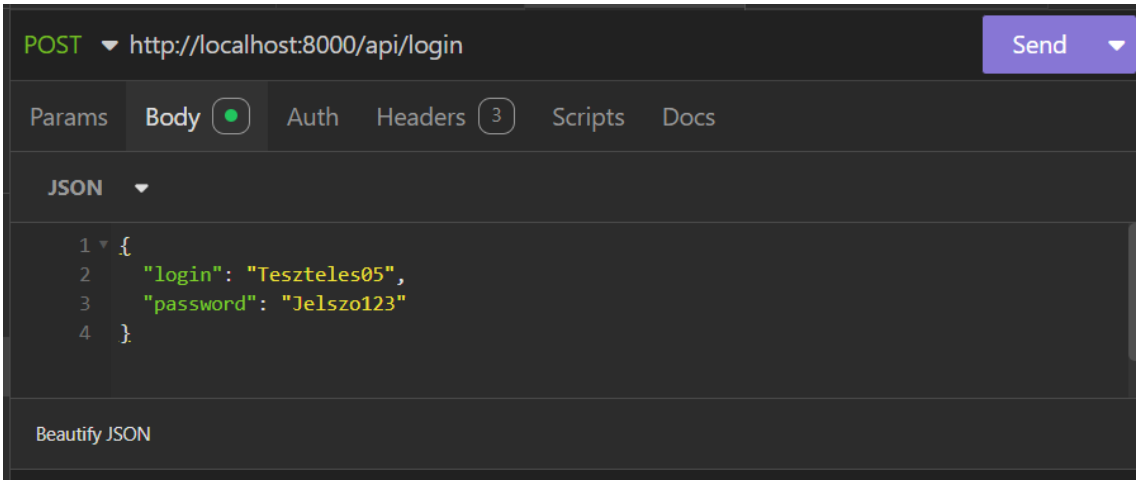
Insomnia tesztek

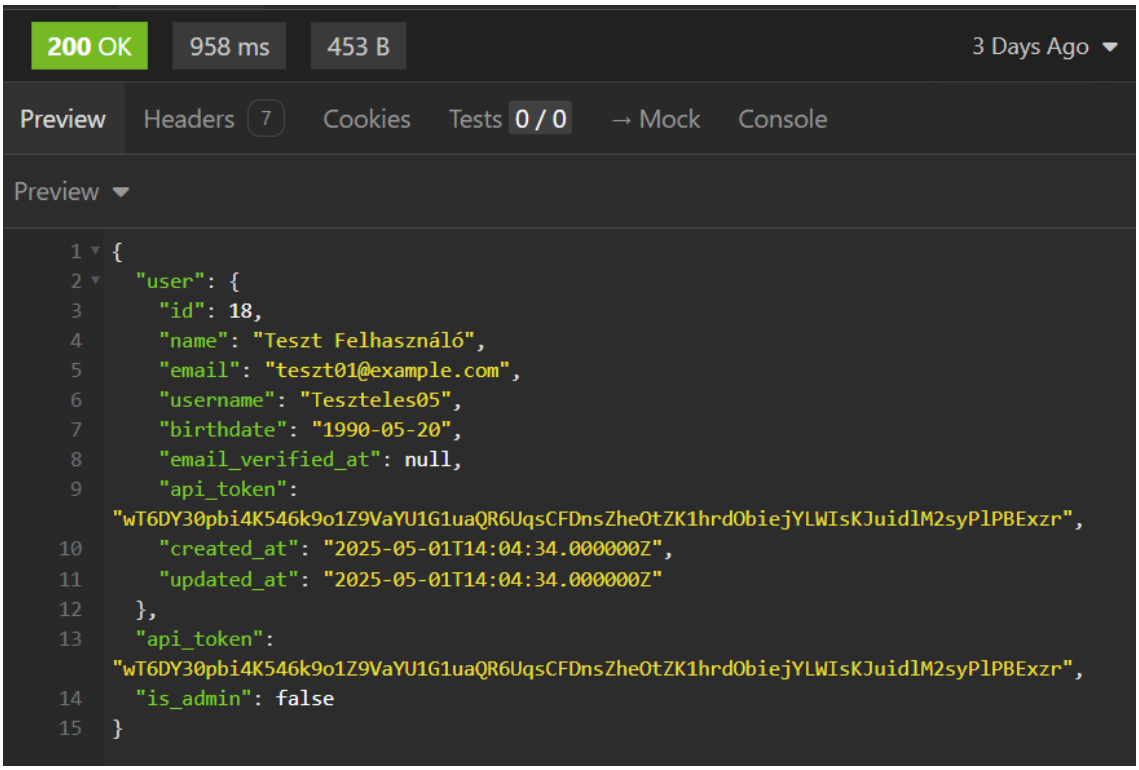
Regisztráció



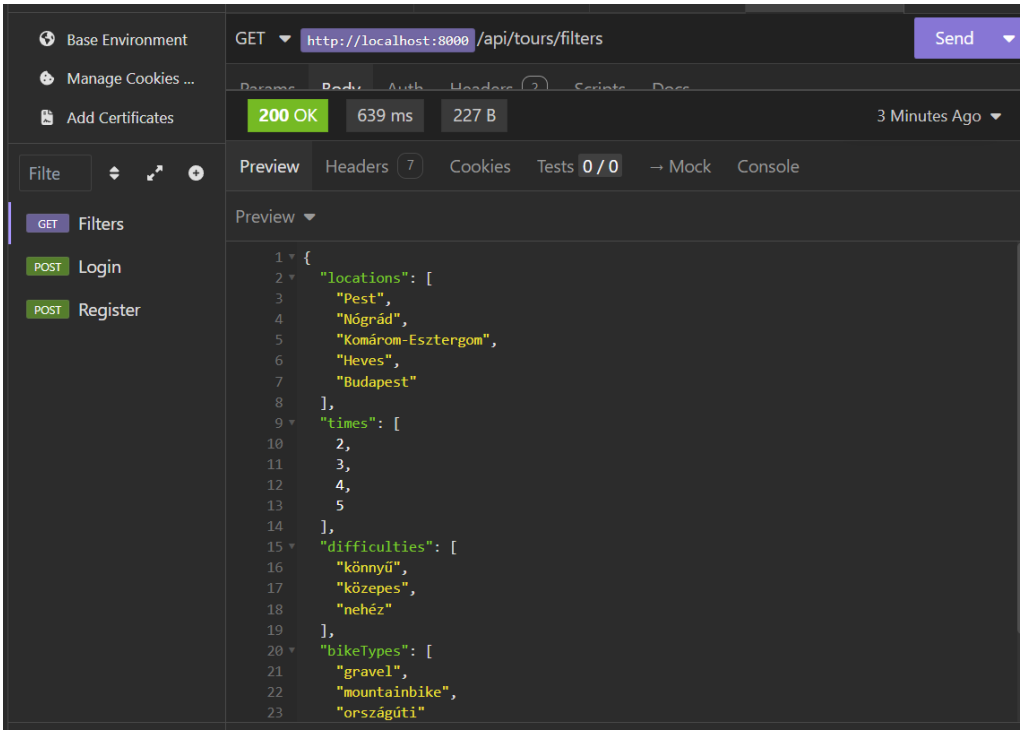


Bejelentkezés

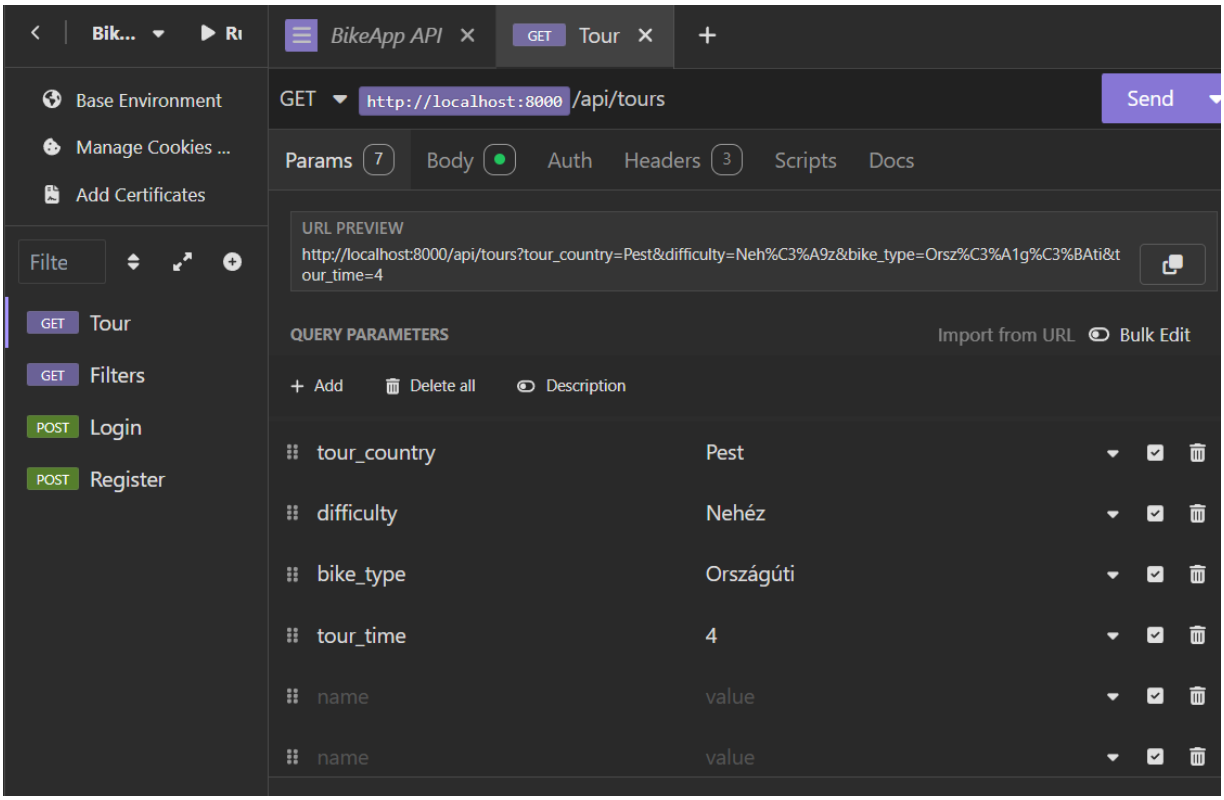


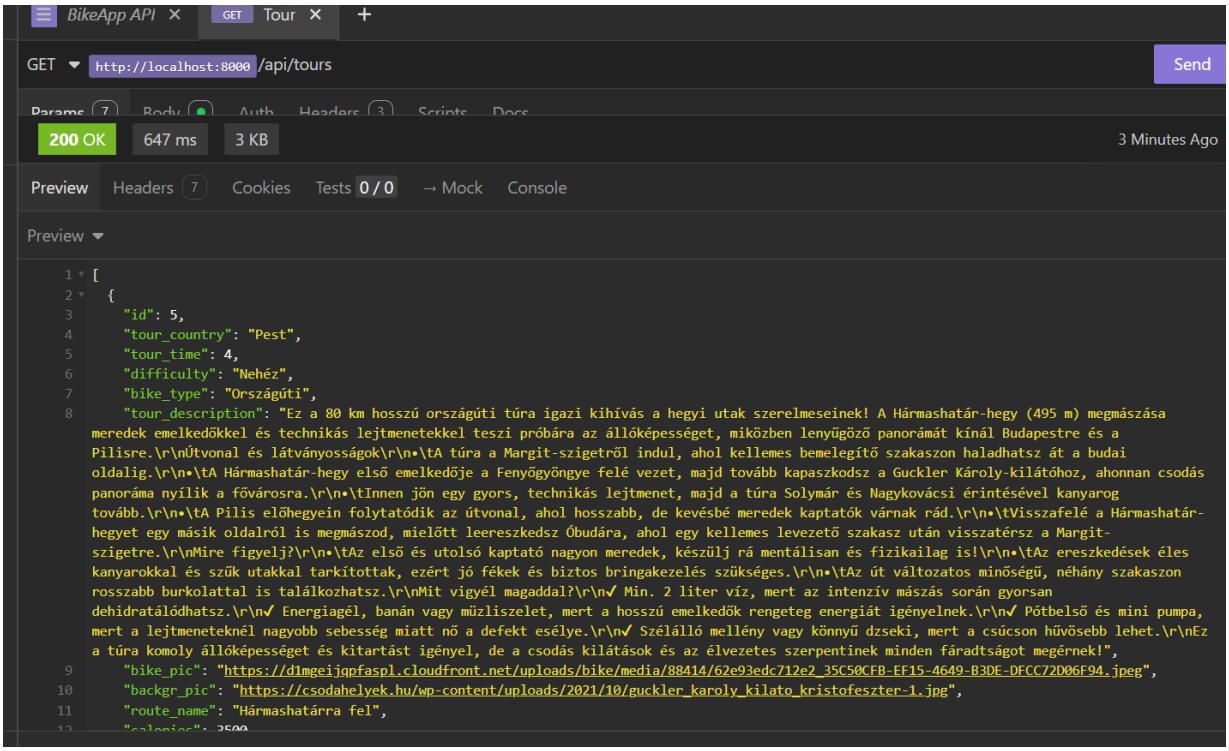


Szűrési paraméterek a túrák kiválasztásához

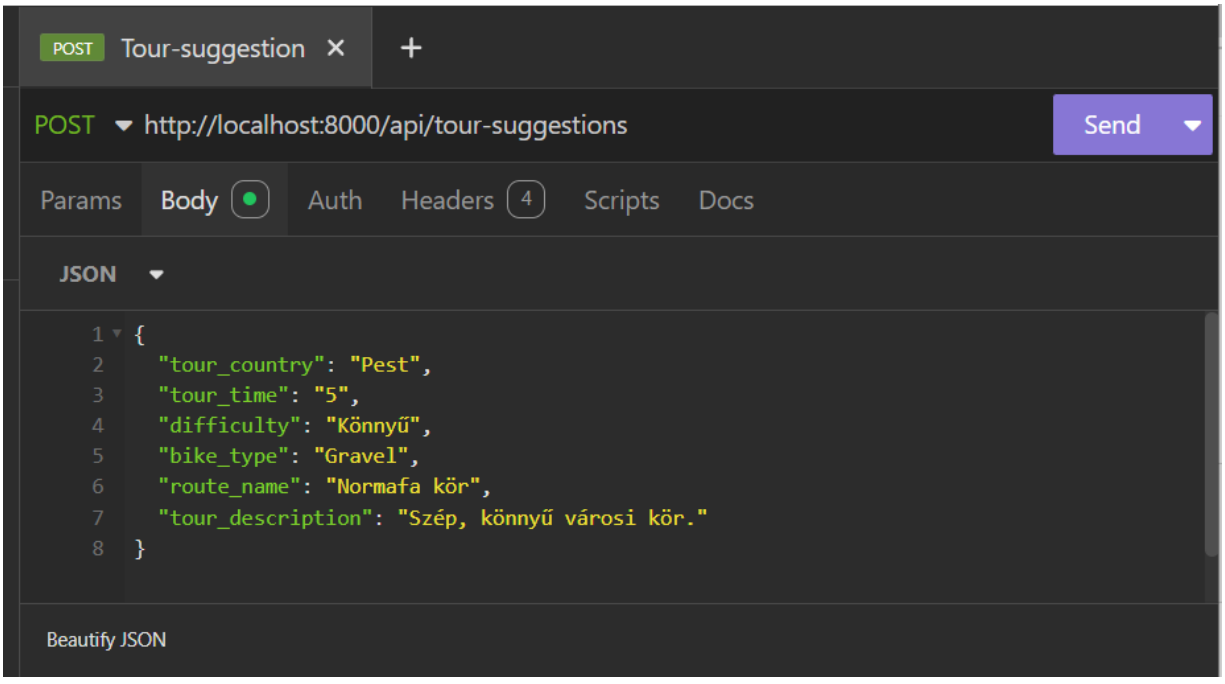


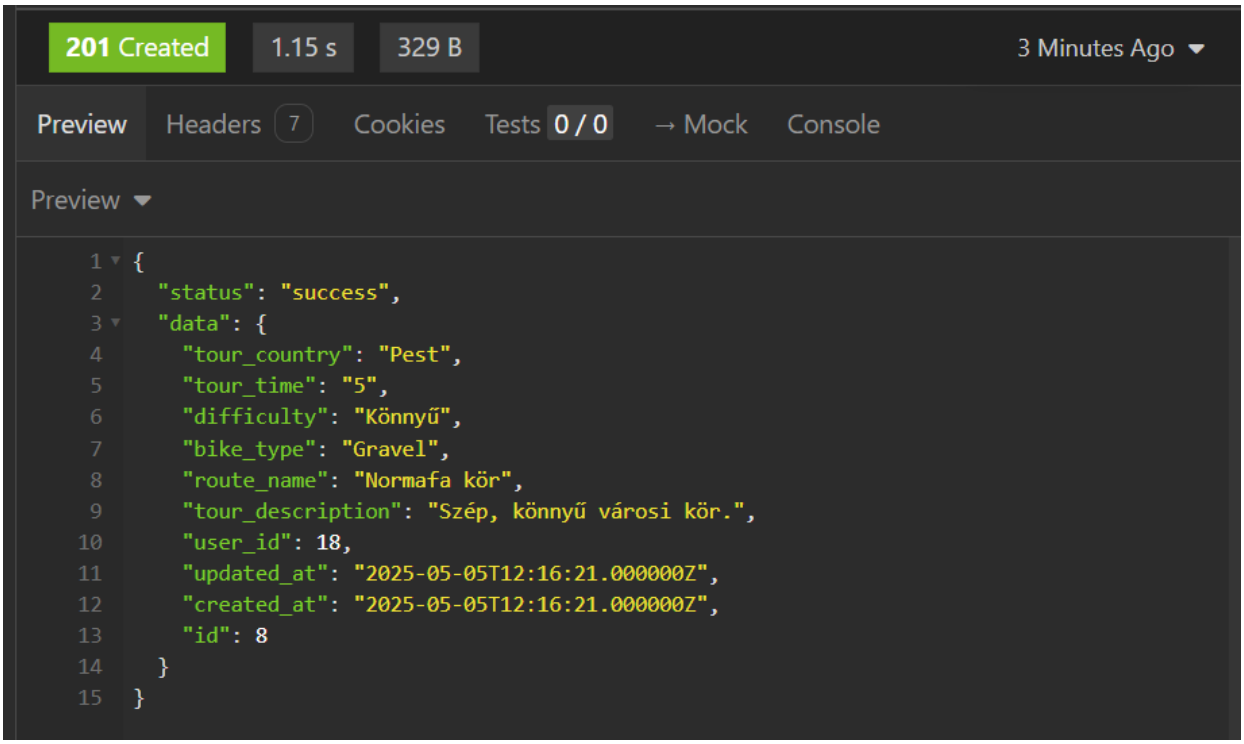
Példa egy túra kiválasztására



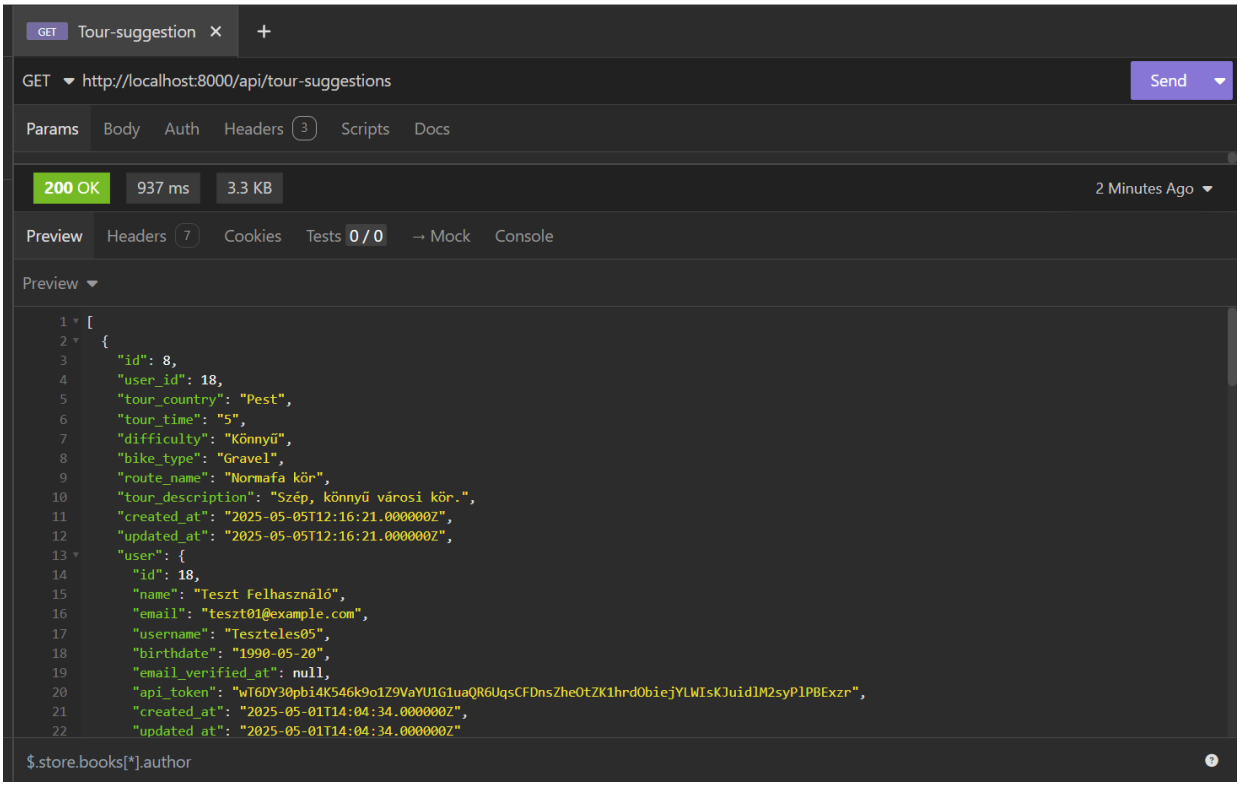


Túrajavaslat elküldése

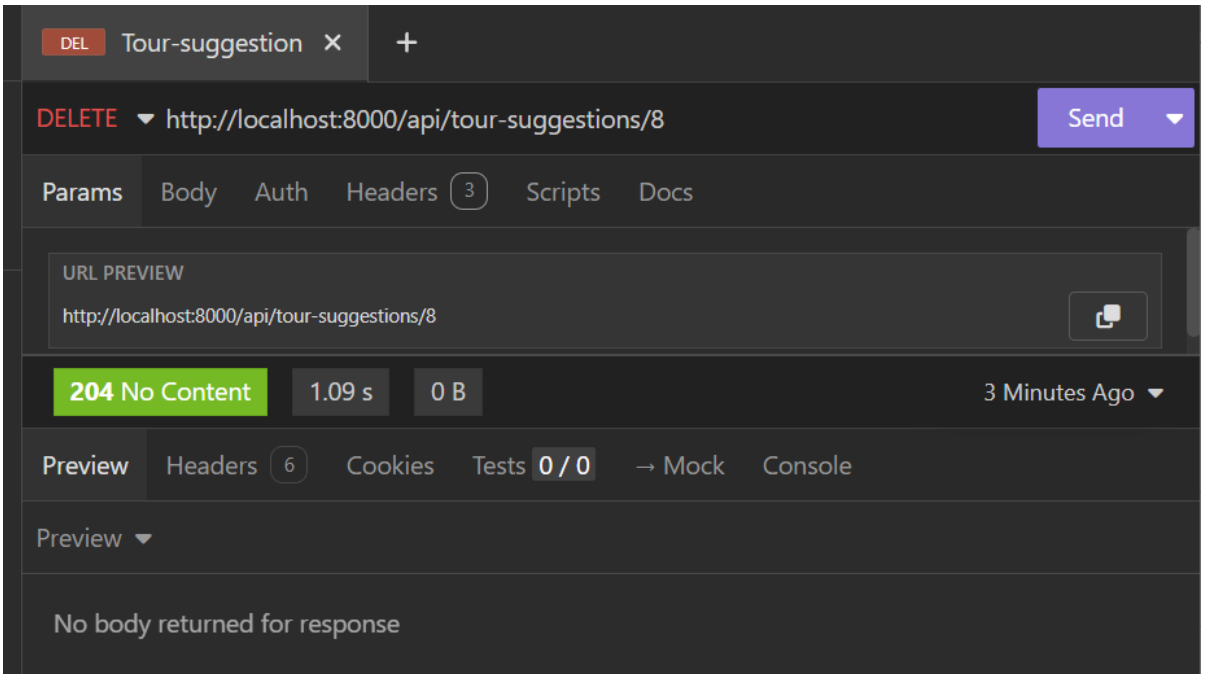




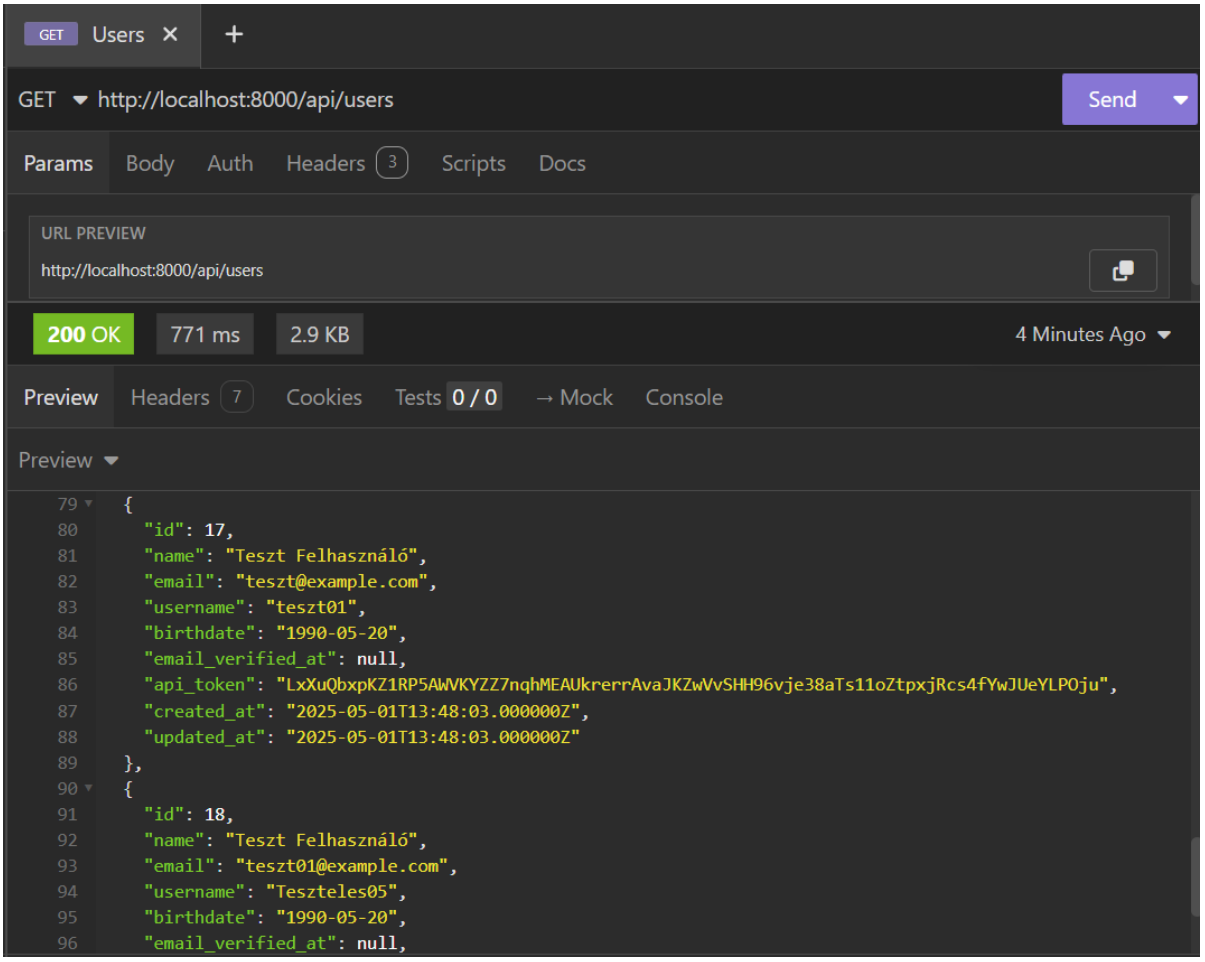
Túrajavaslatok listázása



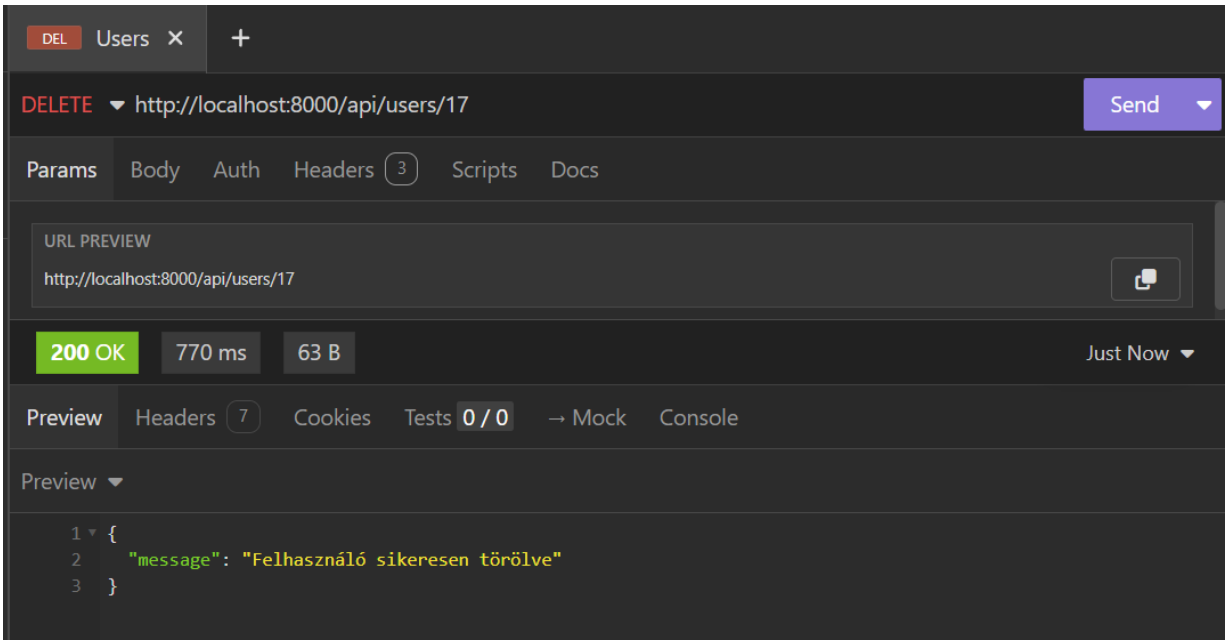
Túrajavaslat törlése



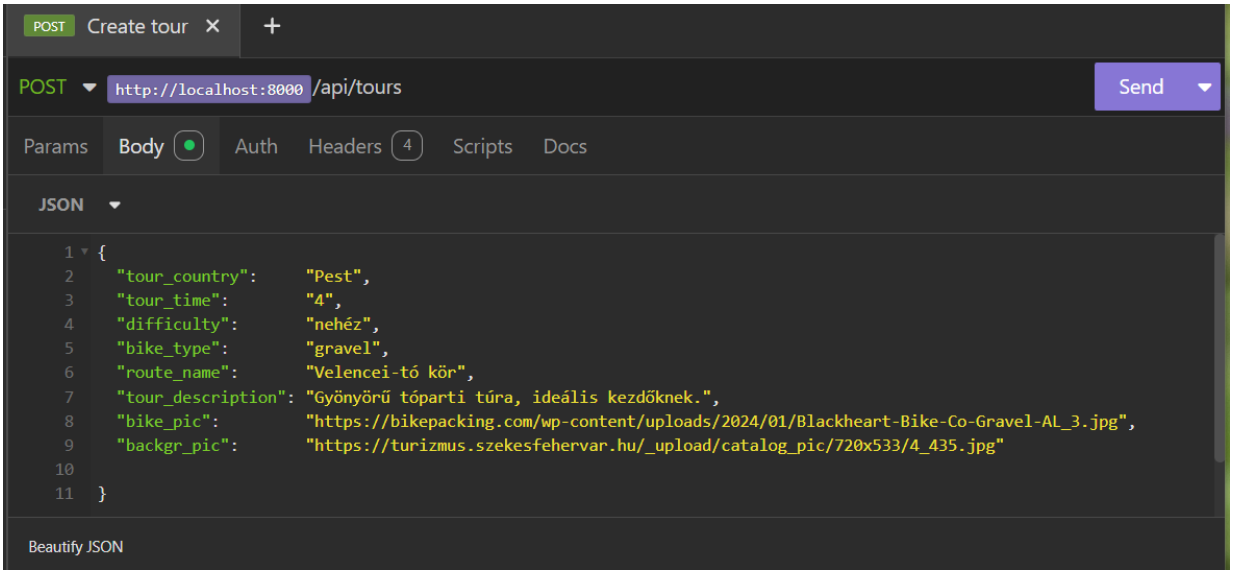
Regisztrált felhasználók listázása

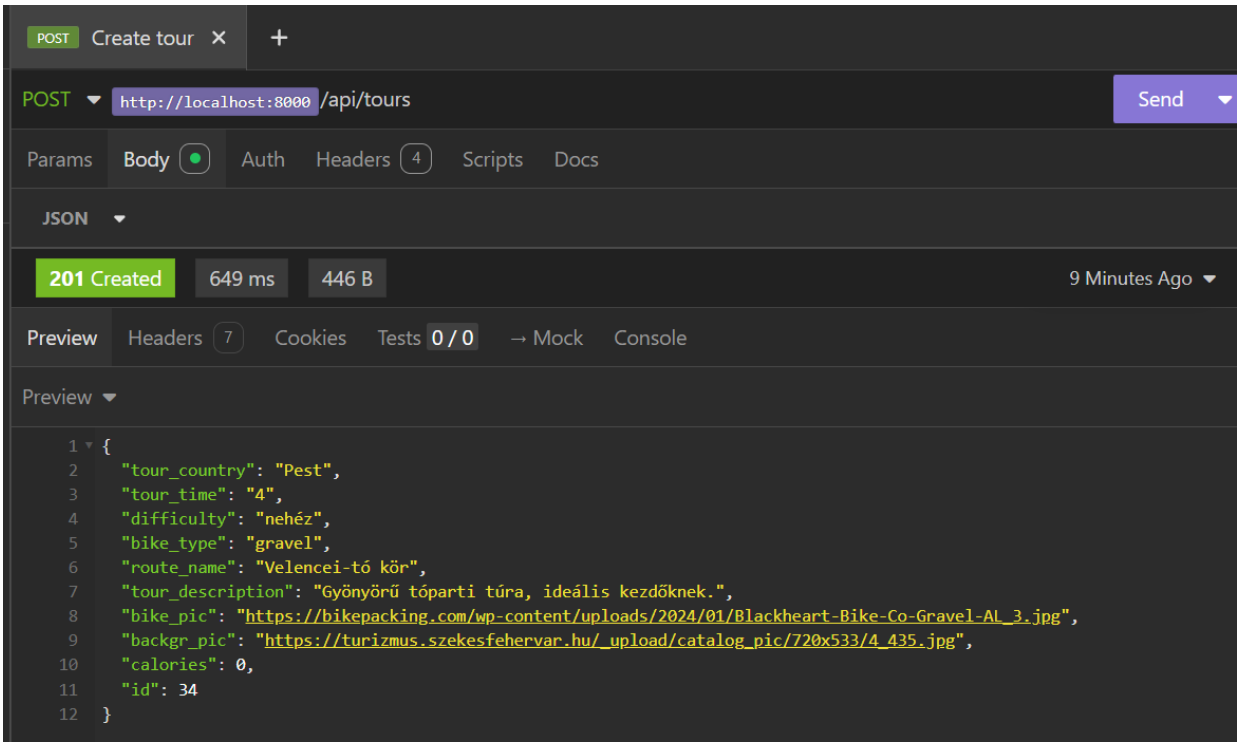


Felhasználó törlése

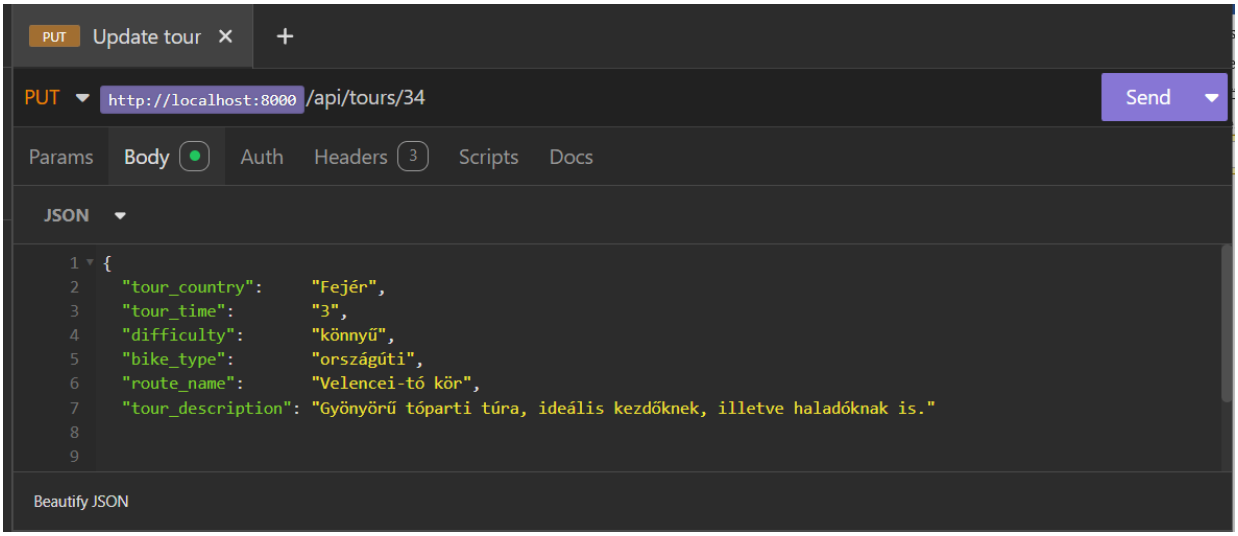


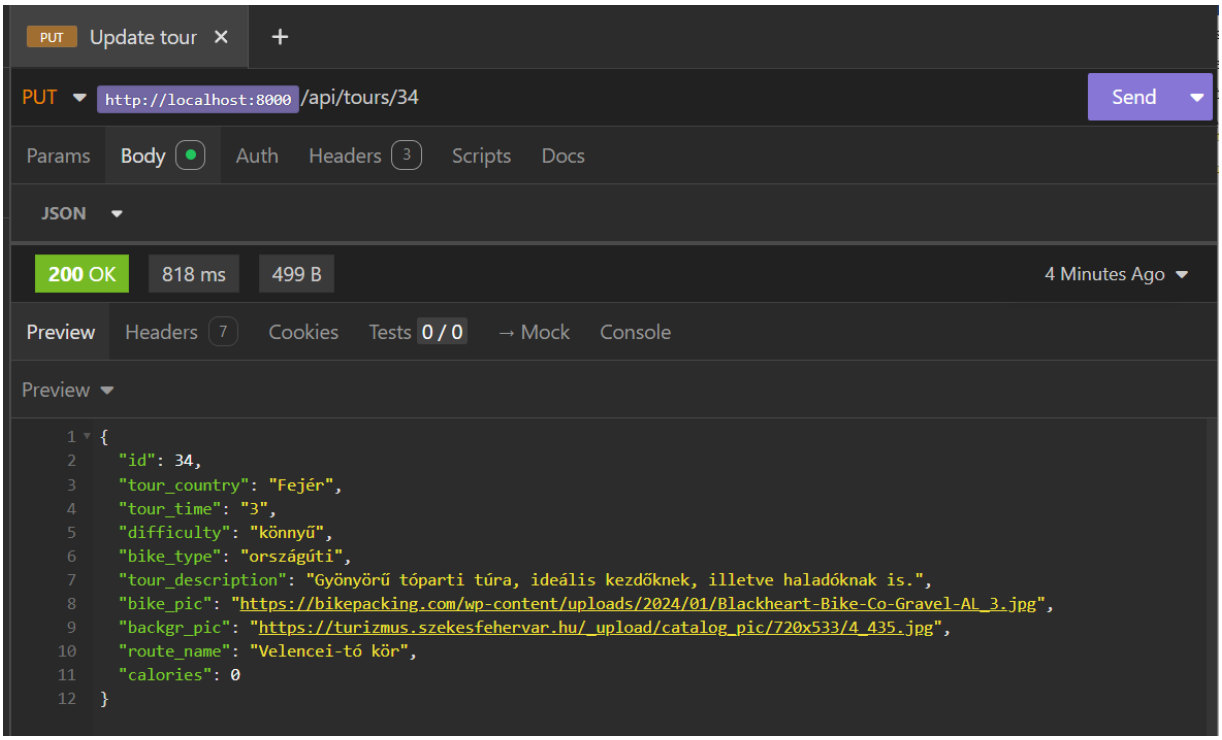
Új túra hozzáadása



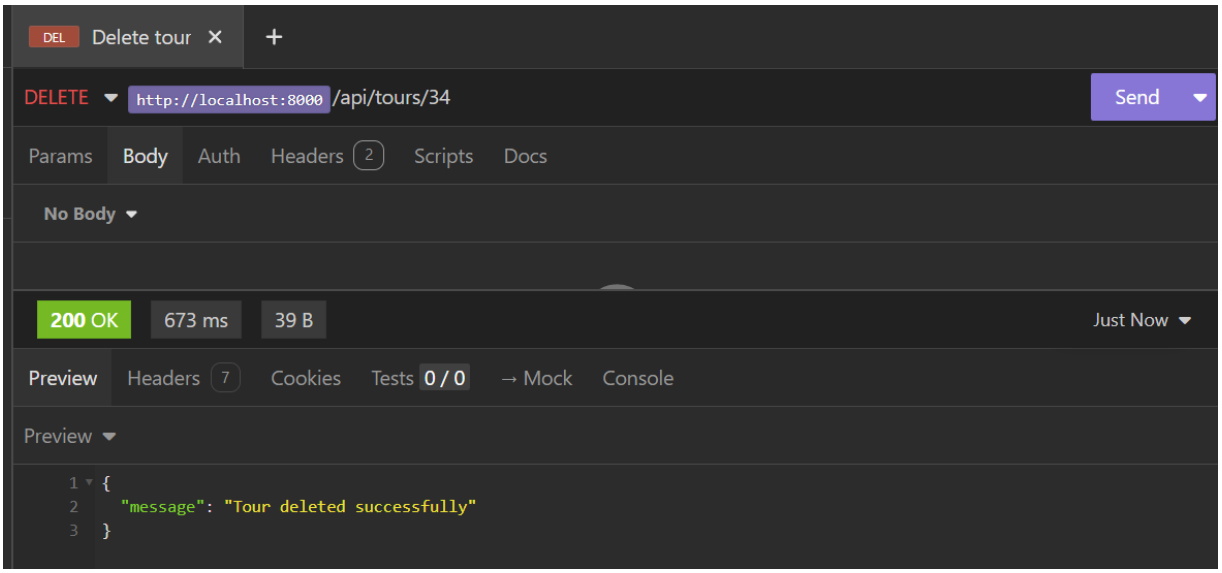


Túra szerkesztése

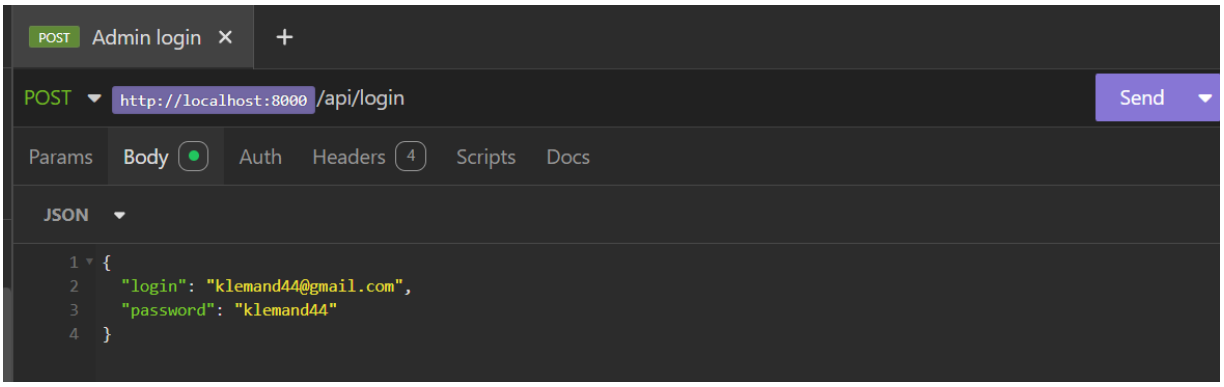


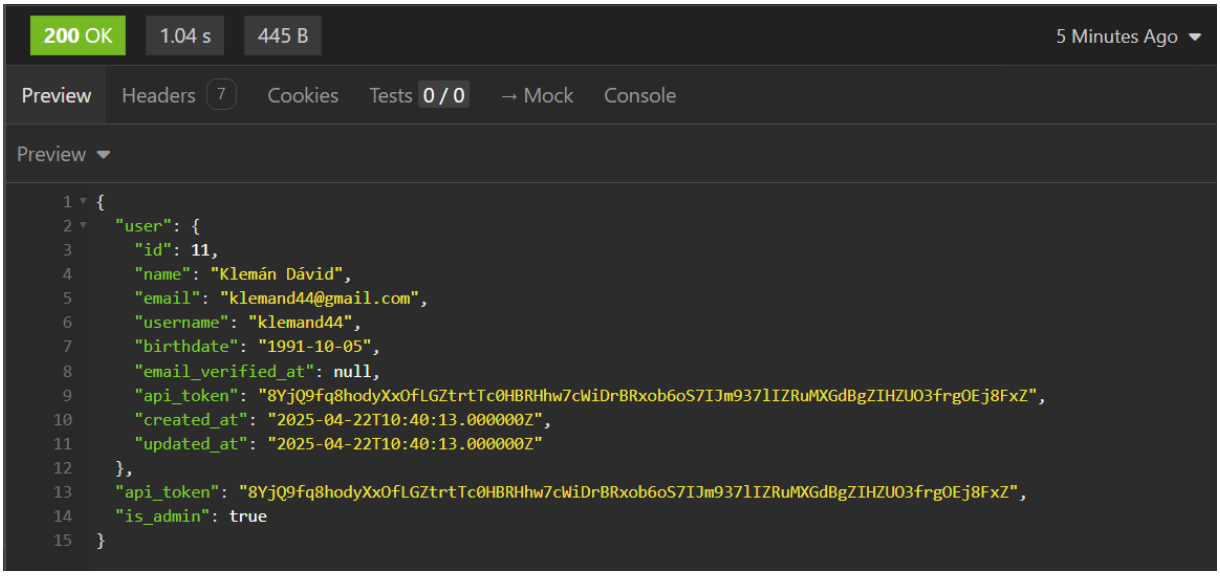


Túra törlése



Admin belépés





További fejlesztési lehetőségek

1. Szerepkör- és jogosultságkezelés (RBAC)

- Admin/Moderator/Ügyfél szerepkörök bevezetése, különböző engedélyekkel (ki hozhat létre, szerkeszthet vagy törölhet túrákat és javaslatokat).
- Laravel-ben a spatie/laravel-permission csomaggal egyszerűen menedzselhetők a jogosultságok.

2. API verziózás és OpenAPI dokumentáció

- A jelenlegi /api/* végpontok mellett vezessétek be az v1, v2 verziókat (/api/v1/tours).
- Generáljatok Swagger/OpenAPI dokumentációt a kód alapján (pl. laravel-swagger), így a kliensfejlesztők mindig naprakész, géppel is olvasható docs-ot kapnak.

3. Konténerizáció és CI/CD

- Docker Compose fájl írása a backendhez és az adatbázishoz (MySQL/MariaDB, Redis).
- GitHub Actions vagy GitLab CI használata:
    - Minden PR-nél lefut a lint, unit/integrációs tesztek és a migrációk.
- Deployment automatizálás (Heroku, DigitalOcean, AWS) Docker-rel.

4. Automata tesztelés és kódminőség

- Unit és Feature tesztek bővítése Laravel PHPUnit-tel, modellekre és controllerekre.
- ESLint, Prettier frontend-hez, PHP_CodeSniffer/ PHP-CS-Fixer backendhez a konzisztens kódformázáshoz.
- Futtasd a teszteket CI pipeline-ban minden commitnál.

5. Caching és teljesítményoptimalizálás

- Redis vagy Memcached bevezetése:
    - Gyakori lekérdezések (pl. /api/tours/filters) cache-elése.
    - Rate limiting Laravel Throttle middleware-rel.
- Adatbázis-oldali indexek finomhangolása a nagyobb táblákon.

6. Térképes és geolokációs funkciók

- A túrákhoz GPS-koordináták tárolása és megjelenítése Google Maps vagy Leaflet használatával.
- Útvonaltervezés, távolság- és magasságprofil integráció (pl. OpenStreetMap API).

7. Realtime értesítések és WebSocket

- Pusher vagy Laravel WebSockets alkalmazása, ha valaki új túrajavaslatot küld, admin azonnal látja valós időben.
- Push értesítések (mobilon/felhasználónak emailben) új túrákról, elfogadott javaslatokról.

8. Felhasználói profil és közösségi elemek

- Saját profiloldal, ahol a felhasználó korábbi túráit, kedvencek listáját, statisztikáit (összes megtett távolság, elégetett kalória) látja.
- Rating és kommentelés: megadhatók a túrákhoz, értékelések és visszajelzések gyűjtése.

9. Offline és mobil támogatás

- PWA megvalósítása Service Worker-rel, hogy hálózat nélkül is működjön az alapfunkció (korábban betöltött túralista).
- Dedikált mobilalkalmazás fejlesztése Ionic/Angular vagy React Native/Flutter használatával, a REST API-ra építve.

10. Harmadik fél integrációk

- OAuth2/Social login (Google, Facebook, GitHub) a könnyebb regisztrációért.
- Email küldő szolgáltatások (Mailgun, SendGrid) beállítása a production környezetben.