LUX REST API  
  
Készítse el egy webalkalmazás backendjét, amely lehetővé teszi egy luxus autókölcsönző cég számára a

foglalások kezelését. Az alkalmazásnak biztosítania kell az autók nyilvántartását, foglalások kezelését és a

kölcsönzési díjak számítását.

Adatmodell:

Car{

id integer($int32)

brand\* string maxLength: 50

minLength: 1

model\* string maxLength: 50

minLength: 1

licensePlate\* string minLength: 1

pattern: ^[A-Z]{3}-\d{3}$

year integer($int32) maximum:

2025 minimum: 2000

dailyPrice integer($int32) maximum:

2000 minimum: 100

[JsonIgnore]

bookings [

nullable: true

Booking{

id integer($int32)

startDate\* string($date-time)

endDate\* string($date-time)

carId\* integer($int32)

totalPrice integer($int32) maximum:

2147483647 minimum: 0

}

userUID\*

}]

string minLength: 1

Booking{

id integer($int32)

startDate\* string($date-time)

endDate\* string($date-time)

carId\* integer($int32)

car Car{

id integer($int32)

brand\* string

maxLength: 50

minLength: 1

model\* string

maxLength: 50

minLength: 1

licensePlate\* string

minLength: 1

pattern: ^[A-Z]{3}-\d{3}$

year integer($int32)

maximum: 2025

minimum: 2000

dailyPrice integer($int32)

maximum: 2000

minimum: 100

}

[NotMapped]

totalPrice integer($int32)

}

Luxus Autó kölcsönző – Vizsga gyakorló feladat

Az adattartalommal kapcsolatos elvárások:

• Brand, Model: Max 50 karakter.

• LicensePlate: ABC-123 formátum.

• Year: 2000–2025 között.

• DailyPrice: 100–2000 között.

• StartDate >= mint az aktuális dátum.

• StartDate < EndDate.

Készítsen Seedert amely feltölti a kezdeti adatokat az adatbázisba, a cars.json felhasználásával.

Az API a következő végpontokat biztosítja:

• GET /api/cars - Autók listázása a rájuk leadott foglalásokkal együtt.

• POST /api/cars - Új autó hozzáadása.

• PUT /api/cars/{id} - Autó adatainak módosítása.

• DELETE /api/cars/{id} - Autó törlése.

• GET /api/bookings - Foglalások listázása

• POST /api/bookings - Új foglalás felvétele.

Ügyeljen rá, hogy ne lehessen foglalást felvenni, ha az adott autó már foglalt a kiválasztott időszakra.

• DELETE /api/bookings/{id} - Foglalás törlése

**1. LARAVEL PROJEKT LÉTREHOZÁSA**

composer create-project laravel/laravel luxury-car-rental

.env fájlban:

DB\_CONNECTION=mysql

DB\_DATABASE=painting

DB\_USERNAME=root

DB\_PASSWORD= # vagy amit használsz

**2. MIGRÁCIÓ ÉS MODELL: Rental**

php artisan make:model Rental -m (migrációt is)

php artisan make:controller RentalItemController –resource

php artisan make:model Rental -mcr /model, controller, migration

**3. migrations**

**CAR**

 public function up(): void

{

    Schema::create('cars', function (Blueprint $table) {

        $table->id();

        $table->string('brand', 50);

        $table->string('model', 50);

        $table->string('licensePlate')->unique();

        $table->year('year');

        $table->integer('dailyPrice');

        $table->timestamps();

    });

}

**BOOKING**

public function up(): void

{

    Schema::create('bookings', function (Blueprint $table) {

        $table->id();

        $table->date('start\_date');

        $table->date('end\_date');

        $table->foreignId('car\_id')->constrained()->onDelete('cascade');

        $table->integer('total\_price');

        $table->string('user\_uid');

        $table->timestamps();

    });

}

**4. Futtasd a migrációt:**

php artisan migrate

5. **. CAR model (app/Models/**

<?php

namespace App\Models;

use Illuminate\Database\Eloquent\Factories\HasFactory;

use Illuminate\Database\Eloquent\Model;

class Car extends Model

{

    use HasFactory;

    protected $fillable = ['brand', 'model', 'licensePlate', 'year', 'dailyPrice'];

    public function bookings()

    {

        return $this->hasMany(Booking::class);

    }

}

**6.Booking Modell**

<?php

namespace App\Models;

use Illuminate\Database\Eloquent\Model;

use Illuminate\Database\Eloquent\Factories\HasFactory;

class Booking extends Model

{

    use HasFactory;

    protected $fillable = ['start\_date', 'end\_date', 'car\_id', 'total\_price', 'user\_uid'];

    public function car()

    {

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

    }

}

}

**7. json fájl bemásolása**

luxury-car-rental\database\seeders\json

cars.json

**8. Seeder létrehozása:**

**php artisan make:seeder CarSeeder**

<?php

namespace Database\Seeders;

use Illuminate\Database\Seeder;

use App\Models\Car;

use Illuminate\Support\Facades\File;

class CarSeeder extends Seeder

{

    public function run(): void

    {

        $path = database\_path('seeders/json/cars.json');

        $cars = json\_decode(file\_get\_contents($path), true);

        foreach ($cars as $car) {

            Car::create($car);

        }

    }

}

HIBA ESETÉRE:

<?php

namespace Database\Seeders;

use Illuminate\Database\Seeder;

use App\Models\Car;

use Illuminate\Support\Facades\File;

class CarSeeder extends Seeder

{

public function run(): void

{

$path = database\_path('seeders/json/cars.json');

if (!File::exists($path)) {

$this->command->error("cars.json fájl nem található: $path");

return;

}

$cars = json\_decode(File::get($path), true);

if (!is\_array($cars)) {

$this->command->error("Hibás JSON formátum: $path");

return;

}

foreach ($cars as $car) {

Car::create($car);

}

$this->command->info("CarSeeder lefutott: " . count($cars) . " rekord.");

}

}

**9. DatabaseSeeder.php fájlban regisztrálni**

public function run(): void

{

$this->call(CarSeeder::class);

}

TELJES:

<?php

namespace Database\Seeders;

use Illuminate\Database\Seeder;

use App\Models\Car;

use Illuminate\Support\Facades\File;

class CarSeeder extends Seeder

{

    public function run(): void

    {

        $path = database\_path('seeders/json/cars.json');

        $cars = json\_decode(file\_get\_contents($path), true);

        foreach ($cars as $car) {

            Car::create($car);

        }

    }

}

10. Futatás seed

**php artisan db:seed**

esetleg: php artisan migrate:fresh –seed

**11. ROUTE-ok beállítása (routes/api.php)**

Route::apiResource('cars', CarController::class);

Teljes

<?php

use Illuminate\Support\Facades\Route;

use App\Http\Controllers\CarController;

use App\Http\Controllers\BookingController;

Route::apiResource('cars', CarController::class);

Route::apiResource('bookings', BookingController::class);

//Route::get('/cars', [CarController::class, 'index']);

//Route::post('/cars', [CarController::class, 'store']);

//Route::get('/cars/{id}', [CarController::class, 'show']);

//Route::put('/cars/{id}', [CarController::class, 'update']);

//Route::delete('/cars/{id}', [CarController::class, 'destroy']);

**12. CAR- CONTROLLER LÉTREHOZÁSA**

php artisan make:controller CarController –api

<?php

namespace App\Http\Controllers;

use App\Models\Car;

use Illuminate\Http\Request;

class CarController extends Controller

{

    // Összes autó lekérdezése

    public function index()

    {

        return response()->json(Car::all());

    }

    // Egy autó lekérdezése azonosító alapján

    public function show(string $id)

    {

        $car = Car::find($id);

        if (!$car) {

            return response()->json(['message' => 'Car not found'], 404);

        }

        return response()->json($car);

    }

    // Új autó létrehozása

    public function store(Request $request)

    {

        $validated = $request->validate([

            'brand' => 'required|string|max:255',

            'model' => 'required|string|max:255',

            'licensePlate' => 'required|string|max:20|unique:cars,licensePlate',

            'year' => 'required|integer|min:1900|max:' . date('Y'),

            'dailyPrice' => 'required|numeric|min:0',

        ]);

        $car = Car::create($validated);

        return response()->json($car, 201);

    }

    // Autó módosítása

    public function update(Request $request, string $id)

    {

        $car = Car::find($id);

        if (!$car) {

            return response()->json(['message' => 'Car not found'], 404);

        }

        $validated = $request->validate([

            'brand' => 'sometimes|required|string|max:255',

            'model' => 'sometimes|required|string|max:255',

            'licensePlate' => 'sometimes|required|string|max:20|unique:cars,licensePlate,' . $car->id,

            'year' => 'sometimes|required|integer|min:1900|max:' . date('Y'),

            'dailyPrice' => 'sometimes|required|numeric|min:0',

        ]);

        $car->update($validated);

        return response()->json($car);

    }

    // Autó törlése

    public function destroy(string $id)

    {

        $car = Car::find($id);

        if (!$car) {

            return response()->json(['message' => 'Car not found'], 404);

        }

        $car->delete();

        return response()->json(['message' => 'Car deleted successfully']);

    }

}

**14.BOOKING CONTROLLER**

**php artisan make:controller BookingController --api**

<?php

namespace App\Http\Controllers;

use App\Models\Booking;

use App\Models\Car;

use Illuminate\Http\Request;

class BookingController extends Controller

{

    public function index()

    {

        return response()->json(Booking::with('car')->get());

    }

    public function show(string $id)

    {

        $booking = Booking::with('car')->find($id);

        if (!$booking) {

            return response()->json(['message' => 'Booking not found'], 404);

        }

        return response()->json($booking);

    }

    public function store(Request $request)

    {

        $validated = $request->validate([

            'car\_id' => 'required|exists:cars,id',

            'user\_uid' => 'required|string|max:255',

            'start\_date' => 'required|date',

            'end\_date' => 'required|date|after\_or\_equal:start\_date',

        ]);

        // Ütközés ellenőrzése

        $conflict = Booking::where('car\_id', $validated['car\_id'])

            ->where(function ($query) use ($validated) {

                $query->whereBetween('start\_date', [$validated['start\_date'], $validated['end\_date']])

                      ->orWhereBetween('end\_date', [$validated['start\_date'], $validated['end\_date']])

                      ->orWhere(function ($query2) use ($validated) {

                          $query2->where('start\_date', '<=', $validated['start\_date'])

                                 ->where('end\_date', '>=', $validated['end\_date']);

                      });

            })->exists();

        if ($conflict) {

            return response()->json(['message' => 'This car is already booked for the selected period.'], 409);

        }

        // Ár kiszámítása

        $days = (new \DateTime($validated['start\_date']))->diff(new \DateTime($validated['end\_date']))->days + 1;

        $car = Car::find($validated['car\_id']);

        $totalPrice = $days \* $car->daily\_price;

        $booking = Booking::create([

            ...$validated,

            'total\_price' => $totalPrice,

        ]);

        return response()->json($booking, 201);

    }

    public function update(Request $request, string $id)

    {

        $booking = Booking::find($id);

        if (!$booking) {

            return response()->json(['message' => 'Booking not found'], 404);

        }

        $validated = $request->validate([

            'car\_id' => 'sometimes|required|exists:cars,id',

            'user\_uid' => 'sometimes|required|string|max:255',

            'start\_date' => 'sometimes|required|date',

            'end\_date' => 'sometimes|required|date|after\_or\_equal:start\_date',

        ]);

        // Ha változik a dátum/autó, ellenőrizzük az ütközést

        if (isset($validated['car\_id']) || isset($validated['start\_date']) || isset($validated['end\_date'])) {

            $carId = $validated['car\_id'] ?? $booking->car\_id;

            $start = $validated['start\_date'] ?? $booking->start\_date;

            $end = $validated['end\_date'] ?? $booking->end\_date;

            $conflict = Booking::where('car\_id', $carId)

                ->where('id', '!=', $booking->id)

                ->where(function ($query) use ($start, $end) {

                    $query->whereBetween('start\_date', [$start, $end])

                          ->orWhereBetween('end\_date', [$start, $end])

                          ->orWhere(function ($query2) use ($start, $end) {

                              $query2->where('start\_date', '<=', $start)

                                     ->where('end\_date', '>=', $end);

                          });

                })->exists();

            if ($conflict) {

                return response()->json(['message' => 'This car is already booked for the selected period.'], 409);

            }

            // Ár újraszámolás

            $days = (new \DateTime($start))->diff(new \DateTime($end))->days + 1;

            $car = Car::find($carId);

            $validated['total\_price'] = $days \* $car->daily\_price;

        }

        $booking->update($validated);

        return response()->json($booking);

    }

    public function destroy(string $id)

    {

        $booking = Booking::find($id);

        if (!$booking) {

            return response()->json(['message' => 'Booking not found'], 404);

        }

        $booking->delete();

        return response()->json(['message' => 'Booking deleted successfully']);

    }

}

**9. Teszt**

 GET /api/cars – az összes autó lekérése (pl. a seederből betöltöttek)

 GET /api/cars/{id} – autó lekérése ID alapján

 GET /api/bookings – összes foglalás

 POST /api/bookings – új foglalás (példa JSON-nal)

"car\_id": 1,

"user\_uid": "user123",

"start\_date": "2025-06-01",

"end\_date": "2025-06-05"

}

 DELETE /api/bookings/{id} – foglalás törlése ID alapján

PUT http://localhost:8000/api/bookings/1

{

"start\_date": "2026-07-10",

"end\_date": "2026-07-15"

}