**Backend útmutató**

***Laravel***

1. **Projekt létrehozása**

* **composer create-project laravel/laravel [projekt neve]**
* **php artisan install:api** (laravel 11 esetén)

1. **Projekt kezdeti beállításai**
   * .env fájl konfigurálása

* DB\_CONNECTION = mysql/mariadb
* DB\_DATABASE = [A használt adatbázis neve]

1. **Modell osztályok létrehozása**

* **php artisan make:model [név] --all**
* Szükségtelen fájlok törlése
* **Megkapott adatbázis tábla seederét fontos törölni!**

1. **Migráció megvalósítása** 
   * a táblához, amit nekünk kell létrehozni és feltölteni
   * ha nem az **-all** parancsal hoztuk létre a modell osztályunkat:
     + **php artisan make:migration create\_[tabla\_neve]\_table**
   * Az **up()** metódusban: **$table** változó használata.
   * **$table->bigIncrements('id')** 🡪 auto-incrementing ID
   * VARCHAR**: $table->string('\*')** 🡪 \* az oszlop neve
   * INT: az **$table->integer(\*)**🡪 \* az oszlop neve
   * BOOLEAN**: $table->boolean('\*')** 🡪 \* az oszlop neve
   * DATE**: $table->date('\*')** 🡪 \* az oszlop neve
   * Indexek és egyedi kulcsok
     + **Az $table->primary(\*)** meghatározza az elsődleges kulcsot.
     + **Az $table->unique(\*)** egy egyedi indexet ad hozzá az '\*' oszlophoz.
   * Külső kulcsok és kapcsolatok
     + **$table->unsignedBigInteger('foreign\_id')**
     + **$table->foreign('foreign\_id')->references('id')->on('other\_table')**
     + **Egyszerűbb: $table->foreignIdFor([ModelClass]::class)->constrained();**
   * migráció futtatása:
     + **php artisan migrate**
2. **Teszt adatok generálása**

* DatabaseSeeder-ben seederek hívása:
  + A DatabaseSeeder osztály run metódusában: **$this->call(SeederName::class);.**
* Egyéni seeder létrehozása (**SajatSeeder.php**):
  + A run metódusban használd a modell factory-ját adatok generálásához: **ModelName::factory()->count(darabszam)->create()**;, ahol ModelName a modell neve és darabszam az előállítani kívánt adatok száma.
* Factory létrehozása (**SajatFactory.php**):
  + A definition metódusban határozd meg az adatstruktúrát, visszatérve egy tömbbel, ami az adott modell attribútumait és azok generált értékeit tartalmazza.
* Példák gyakori factory attribútumokra:
  + **Egész számok:** 'age' => $this->faker->numberBetween(18, 65),
  + **Szöveg:** 'name' => $this->faker->name(),
  + **E-mail:** 'email' => $this->faker->unique()->safeEmail(),
  + **Dátum/idő:** 'created\_at' => $this->faker->dateTimeBetween('-1 years', 'now'),
  + **Buleán értékek:** 'is\_active' => $this->faker->boolean(),
  + **Véletlenszerű elemek:** 'status' => $this->faker->randomElement(['new', 'active', 'suspended']),
  + **Idegen kulcsok generálása:** 'some\_foreign\_key\_id' => $this->faker->randomElement(ModelName::pluck('id')->toArray()).
  + Egyedi esetben a **hivatalos dokumentációt** itt találod: [**https://fakerphp.github.io**](https://fakerphp.github.io)
* Seeder futtatása**:** 
  + **php artisan db:seed**

1. **Controller létrehozása**
   * **php artisan make:controller API/ControllerName --api**
   * **CRUD metódusok a controllerben:**
     + **Index:** A **GET** kérések kezelése az összes entitás listázásához

**public function index() {**

**$list = ModelName::all();**

**return response()->json(['data' => $list]);**

**}**

* + - **Store:** Új entitás létrehozása **POST** kéréssel.

**public function store(Request $request) {**

**$validatedData = $request->validate([...]);**

**$item = ModelName::create($validatedData);**

**VAGY**

**$item = new ModelName($request->all());**

**$item-> save();**

**return response()->json($item, 201);**

**}**

* + - **Show:** Egy adott entitás megjelenítése **GET** kéréssel.

**public function show($id) {**

**$item = ModelName::findOrFail($id);**

**if(is\_null($item)){ return response()->json([’message’=> „Item not found with id: $id”], 404); }**

**return response()->json(['data' => $item]);**

**}**

* + - * Alternatíva, ha csak bizonyos attribútumokra van szükség: **ModelName::find($id)->only([...]);**.
    - **Update:** Entitás frissítése **PUT**/**PATCH** kéréssel.

**public function update(Request $request, $id) {**

**$item = ModelName::findOrFail($id);**

**$validatedData = $request->validate([...]);**

**$item->update($validatedData);**

**return response()->json($item);**

**}**

* + - **Destroy:** Entitás törlése **DELETE** kéréssel.

**public function destroy($id) {**

**$item = ModelName::findOrFail($id);**

**$item->delete();**

**return response()->json(null, 204);**

**}**

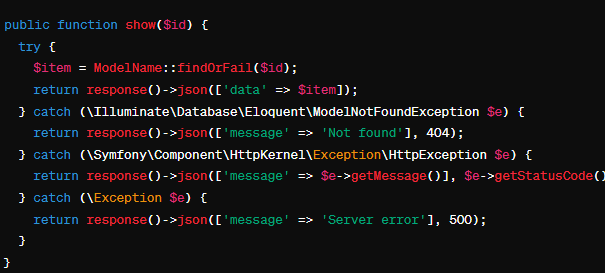
* **Módosításnál vagy új adat felvételénél**
  + Model osztályban 🡪 protected $fillable = [’oszlop\_neve’,…]
  + **Validálás a controllerben:**

**$validatedData = $request->validate([**

**'field\_name' => 'required|string|max:255',**

**'another\_field' => 'required|integer|min:1'**

**]);**

* + **Válaszkódok és üzenetek kezelése:**
    - Sikeres létrehozás esetén 201-es (Created) válaszkódot adj:
      * **return response()->json($item, 201);**
    - Hiba esetén:
      * **return response()->json(['message' => 'Not found'], statuscode);**
  + **Hibakezelés:**
    - **try-catch** blokk használata
    - **Specifikus esetek lekezelése, ha kell:**

1. **Api végpontok létrehozása**
   * Az API végpontok definiálása az **api.php** fájlban
   * API Resource útvonalak:
     + standard **CRUD** útvonal-készletet

**Route::apiResource('entities', ModelNameController::class);**

* + Csak bizonyos útvonalak megírása:
    - **Route::post('entities', [ModelNameController::class, ’store’]);**
    - Itt a post helyett lehet bármi más (get/put/patch/delete)
    - Itt a store helyett a megfelelőt kiválasztani
  + Egyedi útvonal létrehozása:
    - **Route::get('/entities/special-action', [ModelNameController::class, functonName]);**

**©Garai Gábor, 2024**