



**Széchenyi István Katolikus Technikum és Gimnázium**

**Szoftverfejlesztő és -tesztelő projektfeladat**

**PÉKSÉG**

**Készítették:**

Vakherda Ádám,  
Bolyky Máté,  
Iróczki Regina

**Ózd, 2025**

## Tartalomjegyzék

RENDSZERKÖVETELMÉNY .....	4
Szerveroldali követelmények.....	4
Kliensoldali követelmények.....	4
WEBALKALMAZÁS HASZNÁLATA/INDÍTÁSA .....	5
ALKALMAZOTT FEJLESZTŐI ÉS CSOPORTMUNKA ESZKÖZÖK .....	6
Adatbázis .....	7
Frontend.....	7
Backend .....	7
ADATBÁZIS.....	8
Modell leírása .....	8
TÁBLÁK, KAPCSOLATOK .....	12
ER/EK DIAGRAMM .....	13
BACKEND DOKUMENTÁCIÓ.....	14
TESZTELES.....	20
getByid: .....	20
Keresés név és típus alapján:.....	21
Update/Destroy: .....	22
GetHigher/Lowerthan: .....	23
FilterByFizetesimod:.....	24
Seeder: .....	25
Bejelentkezési (login): .....	26
addFelhasznalo: .....	26
TOVÁBBFEJLESZTÉSI LEHETŐSÉGEK .....	27
IRODALOMJEGYZÉK .....	28

## BEVEZETÉS

Projektünk során egy olyan pékség weboldalának megtervezésén dolgoztunk, amelynek célja, hogy napi szinten friss, kiváló minőségű pékáruval lássa el az oktatási intézményeket. Az ötlet alapját saját tapasztalataink adták, hiszen diákokként sokszor szembesültünk azzal, hogy az iskolai menzán vagy büfében a kínált kenyér, kifli vagy zsemle nem volt már igazán friss, mire elfogyasztásra került.

Úgy gondoljuk, hogy egy problémát akkor lehet igazán hatékonyan megoldani, ha az ember maga is megtapasztalta azt. Ezért választottuk ezt a témát: személyesen is érezzük, mennyire fontos lenne a friss pékáru biztosítása az iskolákban, óvodákban, egyetemeken. Célunk, hogy a weboldalunk segítségével az intézmények egyszerűen és gyorsan tudjanak megrendeléseket leadni, és biztosak lehessenek abban, hogy az általuk kapott termékek mindig frissek és ízletesek lesznek.

A weboldal lehetővé teszi a rendelési folyamat digitalizálását, átláthatóságát és automatizálását. A kínálat könnyen böngészhető, a rendelés személyre szabható, a szállítás pedig megbízható és rendszeres.

Jövőbeli terveink között szerepel, hogy a pékséget tovább fejlesztjük: bővítjük a kínálatot (pl. gluténmentes vagy vegán opciókkal), és más intézményeket – például kórházakat vagy irodaházakat – is bevonunk a rendszerbe. Hosszú távon egy olyan hálózatot szeretnénk létrehozni, amely nemcsak friss pékárut biztosít, hanem hozzájárul egy egészségesebb, élhetőbb mindennapi környezet kialakításához is.

Github elérhetőségének a linkje: <https://github.com/bolykymate/Pekseg>

# RENDSZERKÖVETELMÉNY

## Szerveroldali követelmények

- **Operációs rendszer:** Linux vagy Windows
- **Backend:** Node.js/PHP/Python alapú szerver
- **Adatbázis:** MySQL
- **Tárhely:** Minimum 50GB SSD tárhely
- **RAM:** Minimum 4GB RAM
- **Hálózati kapcsolat:** Minimum 100 Mbps internetkapcsolat

## Kliensoldali követelmények

- **Böngésző támogatás:** Chrome, Firefox, Edge, Safari (legújabb verziók)

## Minimális hardverigény

- **Asztali gép/laptop:** Legalább 4GB RAM, 2 magos CPU, internetkapcsolat
- **Mobil eszköz:** Android 8.0+ vagy iOS 12.0+ támogatása

## Szoftver

- **VS Code** v1.96.0 vagy újabb
- **Node.js** v22.13.0 vagy újabb
- **XAMPP** 8.1.12 vagy újabb
- **Composer** 2.8.4 vagy újabb

# WEBALKALMAZÁS HASZNÁLATA/INDÍTÁSA

## **Előfeltételek a webalkalmazás futtatáshoz/telepítéshez:**

- XAMPP (Apache + MySQL) telepítése
- PHP (XAMPP részeként települ)
- Visual Studio Code telepítése
- Composer telepítése (Laravel függőségekhez)
- Angular CLI telepítése
- Node.js és npm telepítése (Angular futtatásához)

## **XAMPP konfigurálása:**

1. Indítsa el a XAMPP Control Panelt, az Apache és MySQL szolgáltatásokat
2. Nyissa meg a `http://localhost/phpmyadmin` felületet

## **Laravel backend indítása:**

1. Projektmappa megnyitása Visual studio Code-ban
2. Ellenőrizzük, hogy az env-ben a pekseg adatbázis van megadva
3. Új terminál nyitása
4. Adatbázis migrálása és feltöltése: „php artisan migrate –seed” parancssal
5. Laravel szerver indítása: „php artisan serve” parancssal
6. A terminálban ekkor megjelenik egy visszajelzés, amely mutatja, hogy a szerver fut  
Ezután a backend elérhető lesz

## **Angular frontend indítása:**

1. Projektmappa megnyitása Visual studio Code-ban
2. Új terminál nyitása
3. Függőségek telepítése: `npm install`
4. Fejlesztői szerver indítása: `ng serve`
5. A terminál visszajelzést ad a sikeres indításról  
Ezután a frontend a terminálban megadott címen (alapértelmezetten `http://localhost:4200`) lesz elérhető

# ALKALMAZOTT FEJLESZTŐI ÉS CSOPORTMUNKA ESZKÖZÖK

A munkánk során többféle eszközt is használtunk, amelyek segítettek a tervezésben, a fejlesztésben és abban, hogy csapatként hatékonyan tudjunk együtt dolgozni.

- **Figma**

A weboldal megtervezésénél a Figma nevű online tervezőprogramot használtuk. Ebben tudtuk előre megrajzolni/megtervezni, hogyan nézzen ki az oldal. Így mindenki látta, mi, hogy fog kinézni, és könnyebb volt megbeszélni az ötleteket.

- **Trello**

A feladatok nyomon követéséhez a Trello-t vettük igénybe. Ebben létre tudtunk hozni listákat és kártyákat, amikre felírtuk, kinek mi a dolga, és mi hol tart. Ez különösen jól jött, mert segített rendszerezni a feladatokat/teendőket.

- **VSC**

A fejlesztéshez a Visual Studio Code-ot használtuk, mivel jól támogatja a különböző programozási nyelveket és keretrendszereket, mint például a HTML, CSS, JavaScript, Angular és PHP. Egyszerű kezelhetősége miatt ideális választás volt számunkra a munka során.

- **GitHub**

A GitHubot a forráskód tárolására és verziókezelésére használtuk. Segített abban, hogy egyszerre tudjunk dolgozni, anélkül, hogy egymás munkáját felülírtuk volna. Minden változtatás követhető és biztonságosan kezelhető volt, így átlátható maradt a fejlesztés folyamata.

- **Discord**

A csapat közötti kommunikációra a Discordot használtuk. A platform/felület lehetőséget biztosított a gyors üzenetváltásra, kérdések gyors megválaszolására, valamint hang- és videóhívások lebonyolítására.

## **Adatbázis**

Az adatbázist MySQL-ben készítettük el, amit a XAMPP-on belül a phpMyAdmin felületen keresztül kezeltünk. Itt hoztuk létre a táblákat a termékek, rendelések, iskolák és egyéb adatok számára.

## **Frontend**

A frontend fejlesztése Angular keretrendszerben történt. Ez lehetővé tette a komponensalapú, fejlesztést és a dinamikus felhasználói élményt. A komponenseket TypeScript-ben írtuk, a stílusok pedig CSS segítségével lettek kialakítva.

## **Backend**

A backend rész Laravel keretrendszerrel készült, PHP nyelven. A Laravel segítségével hoztuk létre azokat a REST API-kat, amiken keresztül az Angular frontend adatokat kér le vagy küld el. A backend feladata az adatbázis kezelése, az adatok ellenőrzése (validálása), valamint minden olyan szerveroldali működés, ami a háttérben zajlik, de fontos a rendszer működése szempontjából.

## ADATBÁZIS

Az adatbázisunk azzal a céllal készült, hogy segítsen egy online rendelési rendszer működésében, amely pékáruk árusítására lett kitalálva. Az adatbázis segítségével könnyedén nyomon követhetők a rendelések fontos részletei, például a rendelés dátuma, a kiszállítás tervezett időpontja, valamint a választott fizetési mód. Ezen kívül a Pékáru tábla információkat tárol/nyújt a termékekről, beleértve azok nevét, típusát és árait. Az adatbázis tartalmazza a felhasználók adatait is, mint például a felhasználó nevét, jelszavát és email címét, melyek segítenek a vásárlók kezelésében. A címek tábla pedig azokat az információkat tartalmazza, amelyek a kézbesítéshez szükségesek, beleértve a felhasználók címét, a telefonszámukat és a számlázási nevet is, így biztosítva, hogy a rendelések pontosan és időben eljussanak a megfelelő címre.

### Modell leírása

#### Pékáru

Mezők	Kulcs fajtája	Adattípus	Rövid leírás
<b>Pid</b>	<b>PRIMARY KEY</b>	<b>int</b>	<b>A pékáruink elsődleges azonosítója</b>
<b>nev</b>		<b>varchar(20)</b>	<b>A pékáruk neve</b>
<b>típus</b>		<b>varchar(20)</b>	<b>A pékáruk fajtáját/típusát tárolja el</b>
<b>ar</b>		<b>int</b>	<b>A pékáruk ára</b>



## Felhasználók

Mezők	Kulcs fajtája	Adattípus	Rövid leírás
<b>Fid</b>	<b>PRIMARY KEY</b>	<b>int</b>	<b>A felhasználók elsődleges azonosítója</b>
<b>nev</b>		<b>varchar(20)</b>	<b>A felhasználók nevét tárolja</b>
<b>jelszo</b>		<b>varchar(30)</b>	<b>A felhasználók jelszavát tárolja</b>
<b>email</b>		<b>varchar(30)</b>	<b>A felhasználók email címeit tárolja</b>

## Cím

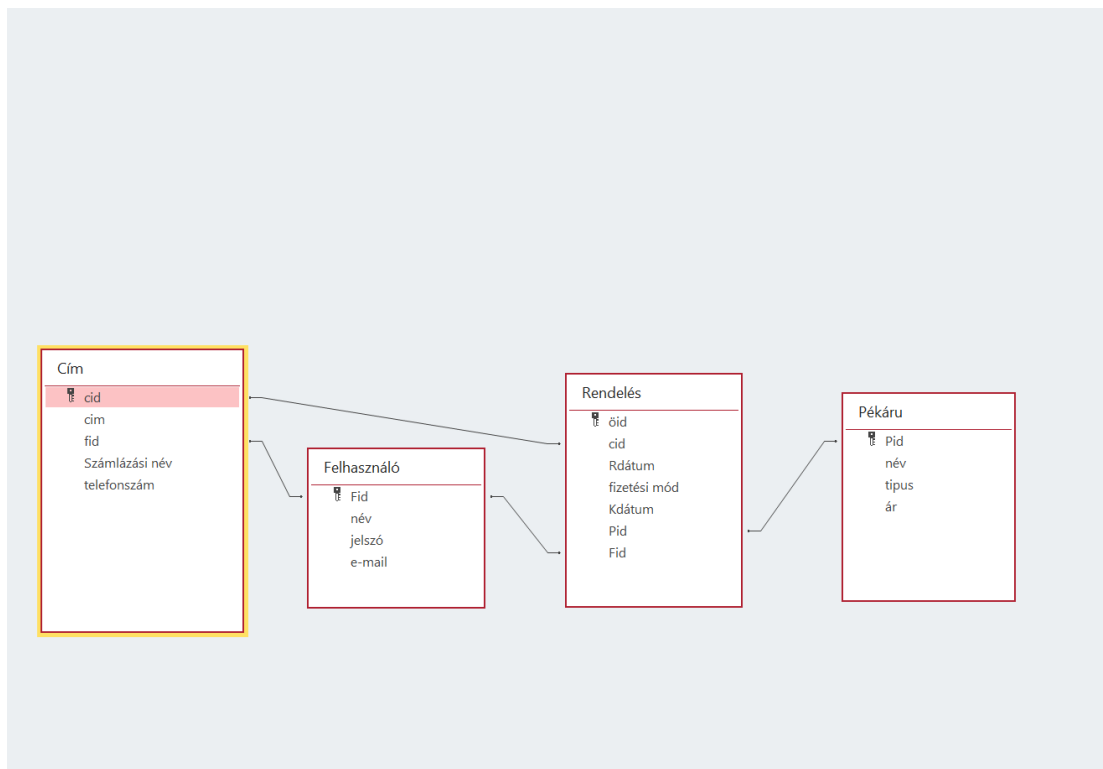
Mezők	Kulcs fajtája	Adattípus	Rövid leírás
<b>Cid</b>	<b>PRIMARY KEY</b>	<b>int</b>	<b>A címtábla elsődleges azonosítója</b>
<b>cim</b>		<b>varchar(30)</b>	<b>A felhasználók teljes címét tárolja el</b>
<b>Fid</b>	<b>FOREIGN KEY</b>		<b>A felhasználók tábla mezőit meghívó idegenkulcs</b>
<b>SzamlazasiNev</b>		<b>varchar(30)</b>	<b>A felhasználók számlázási nevét tünteti fel</b>
<b>telefonszam</b>		<b>varchar(15)</b>	<b>A felhasználók telefonszámait tárolja el</b>

## Rendelés

Mezők	Kulcs fajtája	Adattípus	Rövid leírás
<b>Rid</b>	<b>PRIMARY KEY</b>	<b>int</b>	<b>A rendelés azonosítója</b>
<b>Pid</b>	<b>FOREIGN KEY</b>	<b>int</b>	<b>A pékáru tábla mezőit meghívó idegenkulcs</b>
<b>Fid</b>	<b>FOREIGN KEY</b>	<b>int</b>	<b>A felhasználók tábla mezőit meghívó idegenkulcs</b>
<b>Cid</b>	<b>FOREIGN KEY</b>	<b>int</b>	<b>A címek tábla mezőit meghívó idegenkulcs</b>
<b>RDatum</b>		<b>varchar(20)</b>	<b>A rendelés dátumának ideje</b>
<b>KDatum</b>		<b>varchar(20)</b>	<b>A kiszállítás dátumának várható időpontja</b>
<b>FizetesiMod</b>		<b>boolean</b>	<b>A fizetési mód eldöntésére szolgál</b>

## TÁBLÁK, KAPCSOLATOK

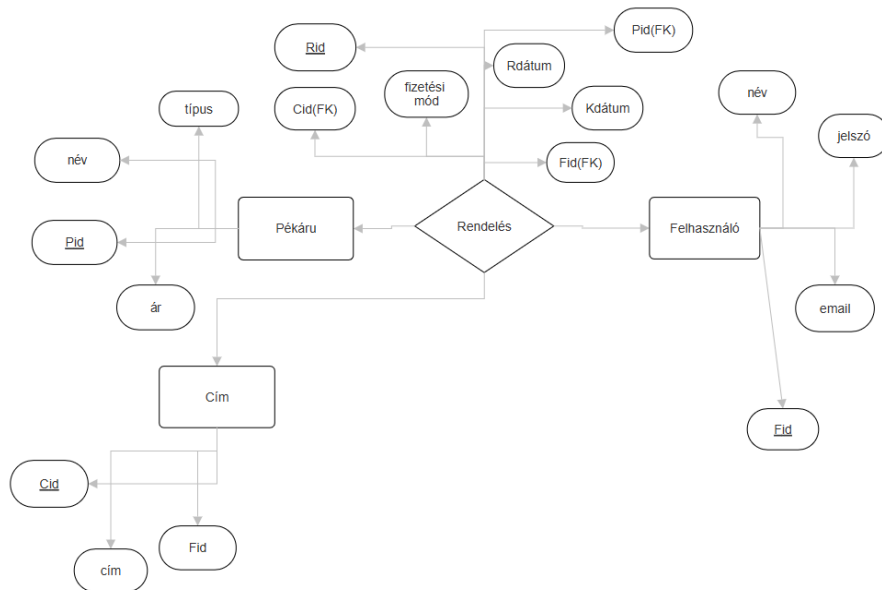
Az alábbi ábra az adatbázis táblái közötti kapcsolatokat mutatja be. Itt látható, hogyan kapcsolódnak egymáshoz a Cím, Felhasználó, Pékáru és Rendelés táblák idegen kulcsokon keresztül.



1. kép: A táblák közötti kapcsolatok ábrázolása.

## ER/EK DIAGRAMM

Az alábbi ER-diagram szemlélteti, hogy a rendszer milyen adatokat tárol, ezek hogyan kapcsolódnak egymáshoz, és miként épül fel az adatbázis logikailag. Ez segít könnyebben átlátni a működését.

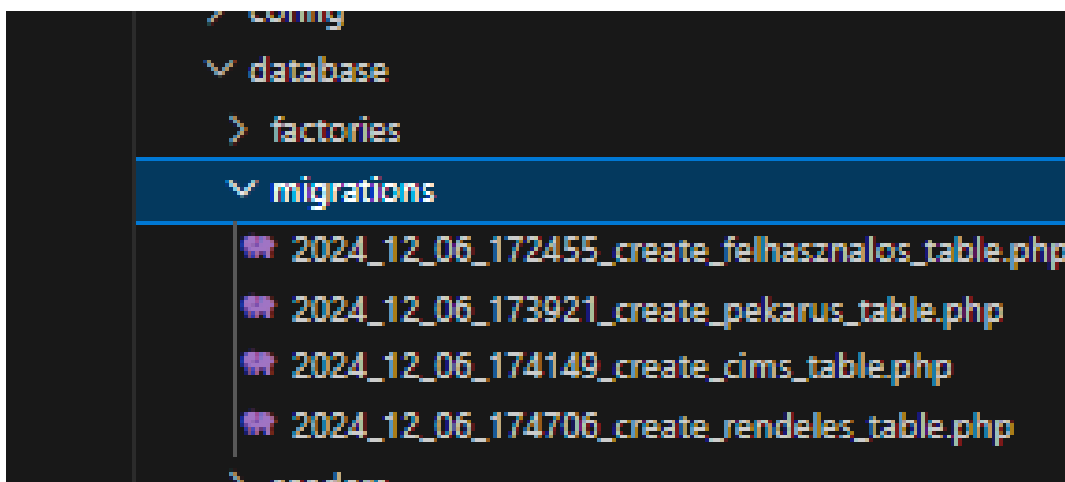


2. kép: Az entitások, azok jellemzői és a közöttük lévő kapcsolatok vizuális bemutatása.

## BACKEND DOKUMENTÁCIÓ

Ez a dokumentáció a pékségünkhöz tartozó API működését mutatja be, amely többek között lehetőséget biztosít új termékek hozzáadására, meglévő adatok lekérdezésére, frissítésére és törlésére is. Az API REST alapú, és JSON formátumban kommunikál. Az alábbi képen a Laravel migrációs fájlokat láthatjuk. Minden fájl egy-egy táblát hoz létre az adatbázisban pl.: Pékárúk, Felhasználók stb. A táblák közötti kapcsolatok is definiálva vannak, például a rendelések táblában a pékáru, felhasználó és cím külső kulcsként szerepel. A migrációk futtatásával (php artisan migrate --seed) az adatbázis automatikusan létrejön a megadott szerkezettel.

1.)



3. kép: Laravel migrációs fájlok

```
pekseg > database > migrations > 2024_12_06_172455_create_felhasznalos_table.php
1  <?php
2
3  use Illuminate\Database\Migrations\Migration;
4  use Illuminate\Database\Schema\Blueprint;
5  use Illuminate\Support\Facades\Schema;
6
7  return new class extends Migration
8  {
9      /**
10       * Run the migrations.
11       */
12     public function up(): void
13     {
14         Schema::create('felhasznalok', function
15             (Blueprint $table) {
16             $table->id();
17             $table->string('nev');
18             $table->string('jelszo');
19             $table->string('email')->unique();
20         });
21
22         /**
23          * Reverse the migrations.
24          */
25     public function down(): void
26     {
27         Schema::dropIfExists('felhasznalok');
28     }
29 };
30
```

4. kép A felhasználók táblájának migrációs fájlja

```

use Illuminate\Database\Schema\Blueprint;
use Illuminate\Support\Facades\Schema;
use Illuminate\Support\Facades\DB;

return new class extends Migration
{
    /**
     * Run the migrations.
     */
    public function up(): void
    {
        Schema::create('rendelessek', function (Blueprint
        $table) {
            $table->id();
            $table->foreignID('pekaru')->references('id')->on
            ('pekaruk');
            $table->foreignID('felhasznalo')->references('id')
            ->on('felhasznalok');
            $table->foreignID('cim')->references('id')->on
            ('cimek');
            $table->string('szamlazasiNev');
            $table->date('RDatum')->nullable()->default(DB::raw
            ('CURRENT_TIMESTAMP'));
            $table->date('KDatum');
            $table->boolean('fizetesiMod');
        });
    }

    /**
     * Reverse the migrations.
     */
    public function down(): void
    {
        Schema::dropIfExists('rendelessek');
    }
};

```

5. kép A rendelések táblájának migrációs fájlja



```
2024_12_06_173921_create_pekarus_table.php ×
pekseg > database > migrations > 2024_12_06_173921_create_pekarus_table.php

1  <?php
2
3  use Illuminate\Database\Migrations\Migration;
4  use Illuminate\Database\Schema\Blueprint;
5  use Illuminate\Support\Facades\Schema;
6
7  return new class extends Migration
8  {
9      /**
10       * Run the migrations.
11       */
12     public function up(): void
13     {
14         Schema::create('pekaruk', function (Blueprint
15             $table) {
16             $table->id();
17             $table->string('nev');
18             $table->string('tipus');
19             $table->integer('ar');
20         });
21     }
22
23     /**
24      * Reverse the migrations.
25      */
26     public function down(): void
27     {
28         Schema::dropIfExists('pekaruk');
29     }
30 };
```

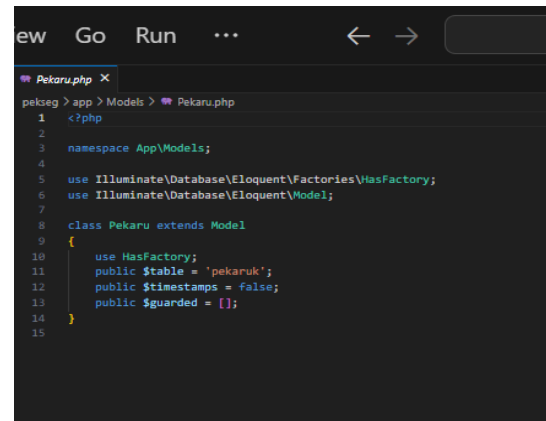
6. kép A pékáruk táblájának migrációs fájlja

```
2024_12_06_174149_create_cims_table.php X
pekseg > database > migrations > 2024_12_06_174149_create_cims_table.php
1  <?php
2
3  use Illuminate\Database\Migrations\Migration;
4  use Illuminate\Database\Schema\Blueprint;
5  use Illuminate\Support\Facades\Schema;
6
7  return new class extends Migration
8  {
9      /**
10       * Run the migrations.
11       */
12     public function up(): void
13     {
14         Schema::create('cimek', function (Blueprint $table)
15         {
16             $table->id();
17             $table->foreignID('felhasznalo')->references
18             ('id')->on('felhasznalok');
19             $table->string('cim')->unique();
20             $table->string('telSzam');
21         });
22     }
23
24     /**
25      * Reverse the migrations.
26      */
27     public function down(): void
28     {
29         Schema::dropIfExists('cimek');
30     }
31 };
```

7. kép A címek táblájának migrációs fájlja

2.)

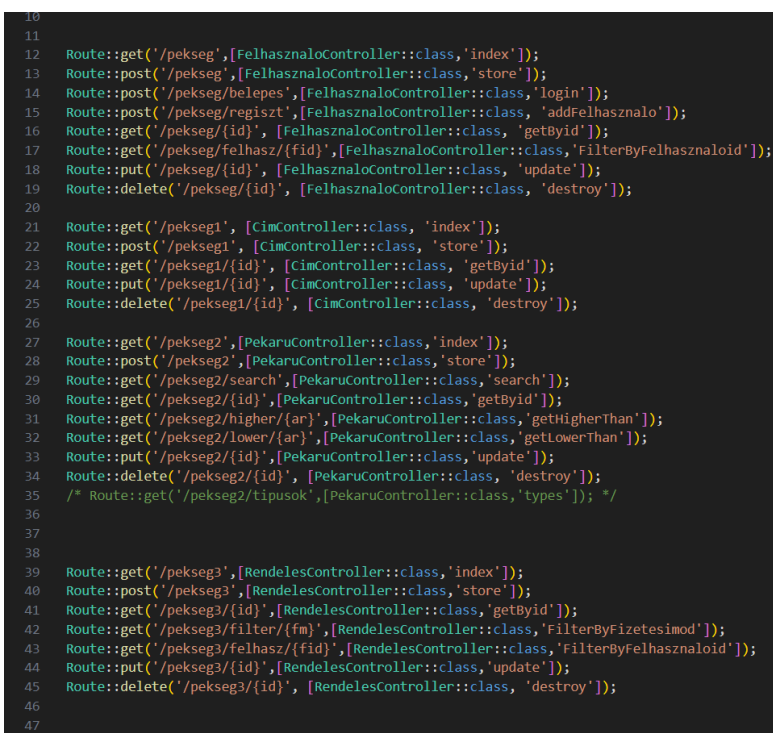
Ez a kép a Laravel modell rétegét mutatja be. A `$table` változó meghatározza a kapcsolódó táblát, a `$timestamps = false` kikapcsolja az automatikus időbélyeg mezőket, míg a `$guarded = [ ]` engedélyezi az össze mező tömeges kitöltését. (Ez az összes táblánál el lett végezve)



```
ew Go Run ...  
Pekar.php X  
peksseg > app > Models > Pekar.php  
1 <?php  
2  
3 namespace App\Models;  
4  
5 use Illuminate\Database\Eloquent\Factories\HasFactory;  
6 use Illuminate\Database\Eloquent\Model;  
7  
8 class Pekar extends Model  
9 {  
10     use HasFactory;  
11     public $table = 'pekaruk';  
12     public $timestamps = false;  
13     public $guarded = [];  
14 }  
15
```

8. kép Laravel modell osztály

3.)



```
10  
11  
12 Route::get('/pekseg',[FelhasznaloController::class,'index']);  
13 Route::post('/pekseg',[FelhasznaloController::class,'store']);  
14 Route::post('/pekseg/belepes',[FelhasznaloController::class,'login']);  
15 Route::post('/pekseg/registzt',[FelhasznaloController::class,'addFelhasznalo']);  
16 Route::get('/pekseg/{id}',[FelhasznaloController::class,'getByid']);  
17 Route::get('/pekseg/felhasz/{fid}',[FelhasznaloController::class,'filterByFelhasznaloid']);  
18 Route::put('/pekseg/{id}',[FelhasznaloController::class,'update']);  
19 Route::delete('/pekseg/{id}',[FelhasznaloController::class,'destroy']);  
20  
21 Route::get('/pekseg1',[CimController::class,'index']);  
22 Route::post('/pekseg1',[CimController::class,'store']);  
23 Route::get('/pekseg1/{id}',[CimController::class,'getByid']);  
24 Route::put('/pekseg1/{id}',[CimController::class,'update']);  
25 Route::delete('/pekseg1/{id}',[CimController::class,'destroy']);  
26  
27 Route::get('/pekseg2',[PekarController::class,'index']);  
28 Route::post('/pekseg2',[PekarController::class,'store']);  
29 Route::get('/pekseg2/search',[PekarController::class,'search']);  
30 Route::get('/pekseg2/{id}',[PekarController::class,'getByid']);  
31 Route::get('/pekseg2/higher/{ar}',[PekarController::class,'getHigherThan']);  
32 Route::get('/pekseg2/lower/{ar}',[PekarController::class,'getLowerThan']);  
33 Route::put('/pekseg2/{id}',[PekarController::class,'update']);  
34 Route::delete('/pekseg2/{id}',[PekarController::class,'destroy']);  
35 /* Route::get('/pekseg2/tipusok',[PekarController::class,'types']); */  
36  
37  
38  
39 Route::get('/pekseg3',[RendelesController::class,'index']);  
40 Route::post('/pekseg3',[RendelesController::class,'store']);  
41 Route::get('/pekseg3/{id}',[RendelesController::class,'getByid']);  
42 Route::get('/pekseg3/filter/{fm}',[RendelesController::class,'filterByFizetesimod']);  
43 Route::get('/pekseg3/felhasz/{fid}',[RendelesController::class,'filterByFelhasznaloid']);  
44 Route::put('/pekseg3/{id}',[RendelesController::class,'update']);  
45 Route::delete('/pekseg3/{id}',[RendelesController::class,'destroy']);  
46  
47
```

9. kép API végpontok

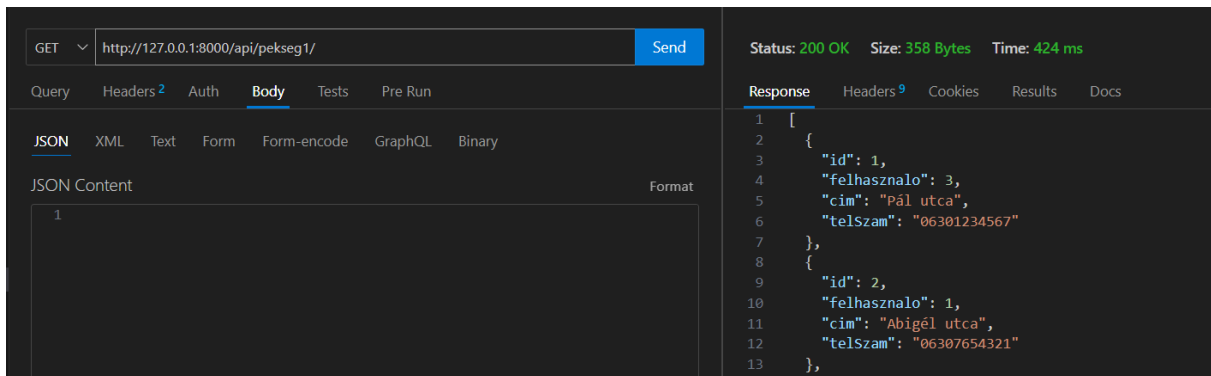
Itt a Laravel útvonalakat/útvonaldefiníciókat láthatjuk. Ezek az API végpontjait határozzák meg a különböző vezérők számára. A **Route::get**, **Route::post**, **Route::put** és **Route::delete** metódusok különböző műveleteket végeznek, például adatokat kérnek le, hoznak létre, módosítanak vagy törölnek.

Mindegyikben egyaránt el van helyezve egy végpont a `getByid`, az `Update`, és `Delete` metódusoknak.

Ezekon kívül található `SearchBy`, `Filterby`, és `Higher/LowerThan` metódus is. Viszont ezek nem mindegyik Controllerben lettek implementálva.

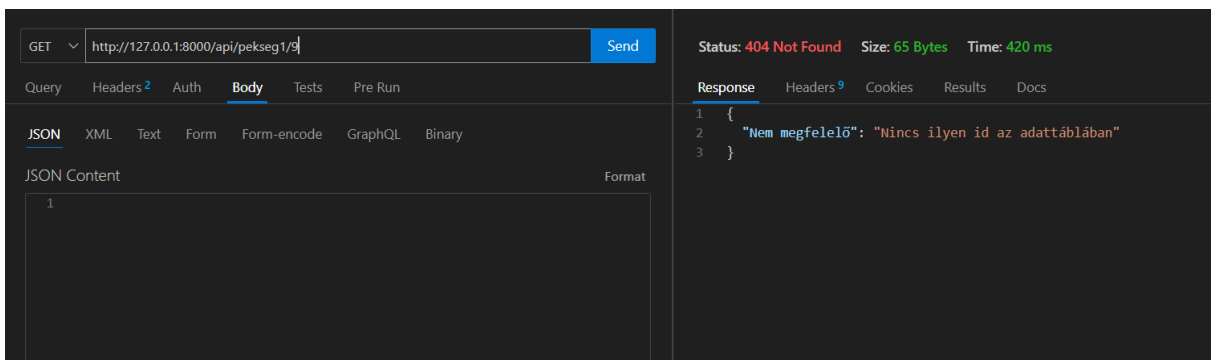
## TESZTELÉS

### getByid:



10. kép getByid metódus sikeres tesztelése

Reprezentáció a 4 getByid metódus egyikéről. A GET kérés a `http://127.0.0.1:8000/api/pekseg1/` URL-re történik, és egy JSON-formátumú választ ad vissza az adott id-jú felhasználó, pékárú, vagy cím részleteivel/adataival.



11. kép getByid metódus tesztje hibás adat/érték beírásakor

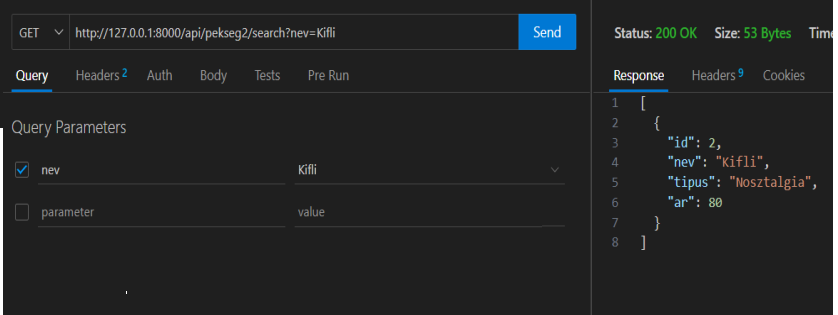
Itt pedig az a hibaüzenet látható, státuszkóddal együtt, amit akkor kapunk vissza, ha például nem létező vagy rossz id-t adunk meg keresésre.

## Keresés név és típus alapján:

A search függvény egy request (kérés) objektumot kap bemenetként, amely alapján keresést végez az adatbázisban. Ha a kérés tartalmaz egy 'nev' (név) paramétert, akkor az adott név alapján szűri az eredményeket, míg, ha a 'tipus' (típus) paraméter szerepel benne, akkor a termék típusa szerint keres. Ha egyik paraméter sem található a kérésben, a függvény egy 400-as hibakódú válaszüzenetet küld vissza.

```
public function search(Request $request)
{
    if ($request->has('nev'))
    {
        $result = Pekar::where('nev', $request->nev)->get();
    }
    elseif ($request->has('tipus'))
    {
        $result = Pekar::where('tipus', $request->tipus)->get();
    }
    else
    {
        return response()->json(['error' => 'Hiányzó keresési paraméter'], 400);
    }

    return response()->json($result);
}
```



12. kép Search metódus

13. kép Search metódus sikeres tesztelése

## Update/Destroy:

Ez a kódrészlet egy API-t mutat be, amely a Pekarú tábla adatainak a frissítését (update) és törlését (destroy) végzi.

(A többi kontrollerben is megtalálható az a két metódus)

### *Update:*

```
public function update(Request $request,$id)
{
    $pekaru=Pekaru::find($id);
    if(is_null($pekaru))
    {
        return response()->json(['Nem található'=>'Nincs ilyen id-jű sor az adattáblában'],404);
    }
    $validator=Validator::make($request->all(),
    [
        'nev'=>'required',
        'tipus'=>'required',
        'ar'=>'required'
    ]
    );
    if($validator->fails())
    {
        return response()->json(['Hiba!'=> 'Fontos adat hiányzik, nem lehet frissíteni'],406);
    }

    $pekaru->update($request->all());
    return response()->json(['Pekarú'=>$pekaru->nev],201);
}
```

14. kép Az update metódus

Ez a függvény egy megadott id alapján keres egy Pekarú rekordot, majd a kapott adatokkal frissíti azt.

Validálja a kérést, és ha hiányzik a nev, típus vagy ar mező, akkor 406 - Nem elfogadható hibát ad vissza.

Ha minden adat megfelelő, frissíti a rekordot.

### *Destroy:*

Ez a függvény pedig egy adott id-jű Pekarú rekordot töröl az adatbázisból. Ha sikeres a törlés, akkor egy 205 – Reset Content státuszkódot küld vissza.

```
public function destroy($id)
{
    $pekaru=Pekaru::find($id);
    if(is_null($pekaru))
    {
        return response()->json(['valami nem jó'=>'Nincs ilyen id-jű sor az adattáblában'],404);
    }
    $pekaru->delete();

    return response('',205);
}
```

15. kép A destroy metódus

## GetHigher/Lowerthan:

```
public function getHigherThan($ar)
{
    $pekaru=Pekaru::where('ar','>',$ar);
    if($pekaru->exists())
    {
        return $pekaru->get();
    }
    else
    {
        return response()->json(['Nem létező érték'=>'Nem lehet az keresett összegnél magasabb árú pékárut találni'],404);
    }
}

public function getLowerThan($ar)
{
    $pekaru=Pekaru::where('ar','<',$ar);
    if($pekaru->exists())
    {
        return $pekaru->get();
    }
    else
    {
        return response()->json(['Nem létező érték'=>'Nem lehet az keresett összegnél alacsonyabb árú pékárut találni'],404);
    }
}
```

16. kép A GetLower és GetHigher metódus

Ezek a függvények visszaadják azokat a pékárukat, amelyek ára magasabb és vagy alacsonyabb, mint a megadott érték (\$ar).

Lehetővé teszi a pékáruk ár szerinti keresését egy weboldalon.

18. kép  
A getHigher  
metódus sikeres  
tesztelése

17. kép  
A getLower  
metódus sikeres  
tesztelése

## FilterByFizetesimod:

```
public function FilterByFizetesimod($fm)
{
    $rendeles=Rendeles::where('fizetesimod','=',$fm);
    if($rendeles->exists())
    {
        return $rendeles->get();
    }
    else{
        return response()->json(['Nem létezik'=>'Nem létezik ilyen féle fizetési mód'],407);
    }
}
```

19. kép A FilterByFizetesimod metódusa

A FilterByFizetesimod lehetővé teszi a Rendelések szűrését fizetési mód alapján, és visszaadja a találatokat, ha léteznek. Két féle fizetési mód jöhet szóba: Kártyás és készpénzesHa a keresésnél 0-t adunk meg akkor a készpénzzel kívánt fizetési kérelmeket kapjuk meg. Az 1-es megjelölés pedig a kártyával való fizetést jelenti.

The screenshot shows a REST client interface with the following details:

- Method:** GET
- URL:** http://127.0.0.1:8000/api/pekseg3/filter/0
- Status:** 200 OK
- Size:** 265 Bytes
- Time:** 535 ms
- Response Body:** A JSON array containing two order objects.

```
[
  {
    "id": 1,
    "pekaru": 7,
    "felhasznalo": 2,
    "cim": 5,
    "szamlazasiNev": "Vicc Elek",
    "RDatum": "2024-11-25",
    "KDatum": "2024-11-26",
    "fizetesimod": 0
  },
  {
    "id": 3,
    "pekaru": 6,
    "felhasznalo": 5,
    "cim": 4,
    "szamlazasiNev": "Para Zita",
    "RDatum": "2024-11-27",
    "KDatum": "2024-11-28",
    "fizetesimod": 0
  }
]
```

20. kép A FilterByFizetesimod sikeres tesztelése



## Seeder:

A DatabaseSeeder osztály feladata az adatbázis feltöltése kezdeti adatokkal a Laravel keretrendszerben. A seeder biztosítja, hogy a rendszer mindig rendelkezzen alapadatokkal, amelyeket a fejlesztés és tesztelés során felhasználhatunk.

Az alábbi modelleket használja:

- **Felhasznalo:** Felhasználói adatok tárolása
- **Pekaru:** Pékáruk tárolása
- **Cim:** Felhasználókhoz tartozó címek kezelése
- **Rendeles:** Megrendelések kezelése

```
public function run(): void
{
    $felhasznalo = ['Kovács Sarolta', 'Vicc Elek', 'Pál Pál', 'Molnár Ödön', 'Para Zita'];
    $jelszo = ['S1234', 'V1234', 'P1234', 'Ö1234', 'Z1234'];
    $email = ['kovacs sarolta@gmail.com', 'viccelek@gmail.com', 'palpal@gmail.com', 'molnarodon@gmail.com', 'parazita@gmail.com'];
    $fRows = count($felhasznalo);
    for ($i=0; $i < $fRows; $i++) {
        Felhasznalo::create(['nev'=> $felhasznalo[$i], 'jelszo'=>$jelszo[$i], 'email'=>$email[$i]]);
    }

    $pnev = ['Kenyér', 'Kifli', 'Zsömlé', 'Kenyér', 'Brios', 'Fánk', 'Pizzás csiga'];
    $tipus = ['Fehér', 'Nostalgia', 'Rozsos', 'Barna', 'Fehér', 'Barackos', 'Csiga'];
    $ar = [800, 80, 120, 820, 200, 250, 275];
    $pRows = count($pnev);
    for ($i=0; $i < $pRows; $i++) {
        Pekaru::create(['nev'=> $pnev[$i], 'tipus'=>$tipus[$i], 'ar'=>$ar[$i]]);
    }

    $felid = [3, 1, 4, 5, 2];
    $cim = ['Pál utca', 'Abigél utca', 'Fecske utca', 'Petőfi utca', 'Luther utca'];
    $telszam = ['06301234567', '06307654321', '06307564231', '06303125467', '06309273638'];
    $cRows = count($felid);
    for ($i=0; $i < $cRows; $i++) {
        Cim::create(['felhasznalo'=> $felid[$i], 'cim'=>$cim[$i], 'telszam'=>$telszam[$i]]);
    }

    $pekid = [7, 5, 6, 2, 3];
    $felid = [2, 3, 5, 1, 4];
    $cimid = [5, 1, 4, 2, 3];
    $szamlazasinev = ['Vicc Elek', 'Pál Pál', 'Para Zita', 'Kovács Sarolta', 'Molnár Ödön'];
    $rendelesD = ['2024.11.25', '2024.11.26', '2024.11.27', '2024.11.28', '2024.11.29'];
    $rendelesK = ['2024.11.26', '2024.11.27', '2024.11.28', '2024.11.29', '2024.11.30'];
    $fizetesimod = [false, true, false, true, true];
    $rRows = count($pekid);
    for ($i=0; $i < $rRows; $i++) {
        Rendeles::create(['pekaru'=> $pekid[$i], 'felhasznalo'=>$felid[$i], 'cim'=>$cimid[$i], 'szamlazasiNev'=>$szamlazasinev[$i],
            'RDatum'=>$rendelesD[$i], 'KDatum'=>$rendelesK[$i], 'fizetesimod'=>$fizetesimod[$i]]);
    }
}
```

21. kép DatabaseSeeder kódja

A run() metódus feltölti az adatbázist tesztadatokkal. Először létrehozza a felhasználókat névvel, jelszóval és email-címmel, majd hozzáadja a pékárukat típussal és árral. Ezután a felhasználók címei és telefonszámjai kerülnek rögzítésre. Végül a rendelések kerülnek mentésre a kapcsolódó adatokkal, beleértve a pékáru azonosítót, számlázási adatokat, dátumokat és fizetési módokat. Az adatokat a megfelelő modellek create() metódusa menti el.

## Bejelentkezési (login):

A login metódus ellenőrzi a felhasználó hitelesítő adatait. Megkeresi a felhasználót az email cím alapján, majd összehasonlítja a megadott jelszót a tárolt jelszóval. Ha minden egyezik, visszaadja a felhasználó adatait.

## addFelhasznalo:

Az addFelhasznalo metódus új felhasználót regisztrál. Ellenőrzi, hogy minden kötelező mező (név, email, jelszó) ki legyen töltve. Ha valami hiányzik, hibát jelez. Ha minden rendben van, elmenti az adatokat és visszaadja az új felhasználót.

```
public function login(Request $request){
    $user = Felhasznalo::where('email','=', $request->email)->first();
    if ($user->exists()) {
        if ($request->email == $user->email && $request->jelszo == $user->jelszo) {
            return $user;
        }
    }
}

public function addFelhasznalo(request $request){
    $validator = Validator::make($request->all(),[
        'vezeteknev' => 'required',
        'keresztnev' => 'required',
        'email' => 'required',
        'jelszo' => 'required'
    ]);
    if($validator->fails()){
        return response()->json($validator->errors(),400);
    }

    $felhasznalok = Felhasznalo::create($request->all());
    return response($felhasznalok,201);
}
```

22. kép A login és addFelhasznalo metódusok

## TOVÁBBFEJLESZTÉSI LEHETŐSÉGEK

A jelenlegi verzió megfelelően használható, de a fejlesztés során több olyan funkció ötlete is felmerült, amit a jövőben érdemes lenne beépíteni:

- A főoldalon lévő legfelkapottabb termékek megjelenítése megvásárolt mennyiség alapján
- Intézmények értékelhetik a kiszállítást és a termékeket
- Csak elérhető termékek szerepeljenek a kínálatban
- Mobilbarát felület vagy külön app fejlesztése

# IRODALOMJEGYZÉK

1. <https://angular.io/docs>
2. <https://developer.mozilla.org/en-US/docs/Web/HTTP/Reference/Methods>
3. [https://inf.u-szeged.hu/~gnemeth/adatbgyak/exe/EK\\_diagram/az\\_egyedkapcsolat\\_diagram\\_elemei.html](https://inf.u-szeged.hu/~gnemeth/adatbgyak/exe/EK_diagram/az_egyedkapcsolat_diagram_elemei.html)
4. <https://laravel.com/docs>
5. <https://martinjoo.dev/layered-architectures-with-laravel>
6. <https://www.fornetti.hu>