Békéscsabai SZC Nemes Tihamér Technikum és Kollégium

Szakképesítés megnevezése: Szoftverfejlesztő és -tesztelő

Azonosító száma: 5 0613 12 03

# VIZSGAREMEK

PeePal – Nyilvános mosdókereső

Készítették:

| | | |
|---|---|---|
| Magony Sándor Máté | Osztály: 13.B | Oktatási azonosító: 72581371703 |
| Horváth Zsolt | Osztály: 13.B | Oktatási azonosító: 72563016770 |
| Börcsök István Milán | Osztály: 13.B | Oktatási azonosító: 72554726794 |

Békéscsaba, 2024/2025

# Tartalomjegyzék

# Ábrajegyzék

# 1. Bevezető és rövid ismertető

Vizsgaremekünk dokumentációjának célja, hogy egy átfogó és széleskörű ismertetőt biztosítson a projekt teljes menetéről, annak életszerű, valós problémákra megoldást nyújtó jellegéről és használatáról, valamint a szoftver hátterét bemutatva a programozási folyamatokról. Célunk, hogy az elkészült alkalmazás a jövőben nyitott legyen új funkciókra, fejlesztésekre és kielégítse a felhasználói élményt. Átlátható kódolás, szép megjelenés. Ez az alapelvünk.

Munkánk első és legfontosabb eleme volt, hogy találjunk egy olyan, a mindennapokban felmerülő, nehézségeket okozó problémát, amelyre valódi megoldást tudunk nyújtani programunk segítségével. Több ötlet és gondolat közül végül a nyilvános illemhelyek helyzetére esett a választás.

Az alkalmazás ötlete abból indult ki, hogy sokan -főként fiatalok, turisták vagy a fővárosban kevésbé jártas emberek- gyakran szembesülnek azzal a problémával, hogy sürgősen szükségük lenne egy közeli illemhelyre, de nem tudják merre induljanak, melyiket válasszák. Bár a gondolat egy ilyen program megalkotására nem új, hiszen léteznek weboldalak, alkalmazások, melyek térképes keresést biztosítanak, de kutatómunkánk során azt tapasztaltuk, hogy többsége elavult, fejlesztése nem biztosított vagy nehezen kezelhető. Ezért döntöttünk úgy, hogy egy modern, felhasználóbarát, közösségi alapon bővíthető platformot hozunk létre.

A projekt keretében egy olyan mobilra és asztali böngészőre egyaránt optimalizált alkalmazást készítettünk, amely lehetőséget nyújt arra, hogy a felhasználók megtekinthessék az aktuális tartózkodási helyükhöz legközelebb eső nyilvános mosdókat, azok helyét, állapotát, nyitvatartását, valamint, hogy biztosítottak-e a mozgáskorlátozottak számára szükséges feltételek.

A fejlesztés során különös figyelmet fordítottunk az átlátható és hatékony kódstruktúrára, a kornak megfelelő webes szabványok betartására, valamint arra, hogy az alkalmazás nyitott legyen a jövőbeni továbbfejlesztések és bővítések irányába. A projekt során React keretrendszert használtunk a felhasználói felület megvalósításához, míg a Backend rész elkészítése Laravel alapokon történt. Az adatok rögzítéséhez és az adatbázis kezeléséhez a MySQL nyújtott segítséget.

A dokumentáció célja tehát nem csak az elkészült alkalmazás bemutatása, hanem az odáig tartó fejlesztési folyamatnak az ismertetése, amely során eljutottunk egy valóban működő, formaiságban megfelelő szoftverhez.

# 2. Téma indoklása

A projekt témájának kiválasztásakor elsődleges szempontunk az volt, hogy olyan valós problémára találjunk megoldást, amely a hétköznapi életben gyakran előfordul, mégis kevés figyelem irányul rá gyakorlati szempontból. A nyilvános mosdók megtalálása pontosan ilyen, egy látszólag egyszerű feladat, amely azonban – különösen nagyvárosi környezetben – gyakran komoly kihívást jelenthet. A probléma nem a mosdók hiányából fakad, hanem azok elérhetőségének, elhelyezkedésének, állapotának vagy nyitvatartásának ismeretéből. Egy idegen környezetben járva a legnagyobb nehézséget az információ hiánya okozza.

Ezzel a felismeréssel született meg a **PeePal** nevű alkalmazásunk ötlete, amely egy nyilvános illemhely kereső rendszer, térképes útvonaltervezővel, aktuális helymeghatározással és egyéb felhasználói funkciókkal. Bár az interneten már fellelhetők hasonló oldalak, alkalmazások, de azok kevésbé kiforrottak, hiányosak több területen is. Mi viszont egy olyan korszerű, felhasználóbarát platformot szerettünk volna létrehozni, amely helymeghatározás útján releváns találatokat kínál, továbbá felhasználói fiók létrehozásával egyfajta közösséget hoz létre.

Fiatalokként gyakran utazunk Budapestre szórakozás, városnézés vagy egyéb utazások miatt és tapasztalatból tudjuk, mennyire kellemetlen tud lenni, amikor gyors megoldásként illemhelyet kell találnunk és nem állnak rendelkezésre megfelelő információk az mosdókról. Egy ilyen helyzetben az ember nem akar hosszú perceket eltölteni kereséssel egy nehezen kezelhető felületen, amely még hiányos is funkciókban. A cél egy gyors, pontos és megbízható szoftver, amin ezt pár kattintással el tudjuk érni.

A **PeePal** célcsoportja így nem csupán a fővárosba látogatóknak szól, hanem a helyi lakosok számára is állandó megoldás jelenthet hosszú távon. A jövőben tervben van a szolgáltatás kibővítése országos szintre, ezt több lépésben képzeljünk el. Elsősorban mindenképpen a megyeszékhelyek a célpont, őket követnék a megyei jogú és közepes méretű városok. Ebben a tekintetben szerencsénkre vagy szerencsétlenségünkre még sok idő áll rendelkezésre, hiszen vidéken még elmaradottnak tekinthető a mosdókkal ellátott körzetek mennyisége.

Összeségében tehát a projektünk nem csak technikailag nyújt kihívást és fejlődési lehetőséget számunkra, hanem társadalmilag is hasznos és talán elvárt, és egy olyan hétköznapi problémára reflektál, amelyhez eddig kevés korszerű és hatékony digitális megoldás született.

# 3. Téma kifejtése

A következő fejezetben ismertetjük a követelményeknek megfelelően létrehozott projektünk részleteit, így szó esik az alkalmazáshoz felhasznált keretrendszerekről, rendszertervről, kifejtjük adatbázisunk működését, foglalkozunk biztonsági kérdésekkel és természetesen a tesztesetek bemutatása is megjelenik.

## 3.2. Követelmények

A fejlesztés megkezdése előtt meghatároztuk azokat a funkcionális és nem funkcionális követelményeket, amelyeket a szoftvernek teljesíteni kell:

**Funkcionális követelmények:**

-Felhasználói regisztráció és bejelentkezés.

-Új mosdók hozzáadása regisztrációhoz kötötten.

-Aktuális helyzetmeghatározás.

-Admin felület új/meglévő mosdók kezelésére.

**Nem funkcionális követelmények:**

-Egyszerű, modern és dizájnos felhasználói felület.

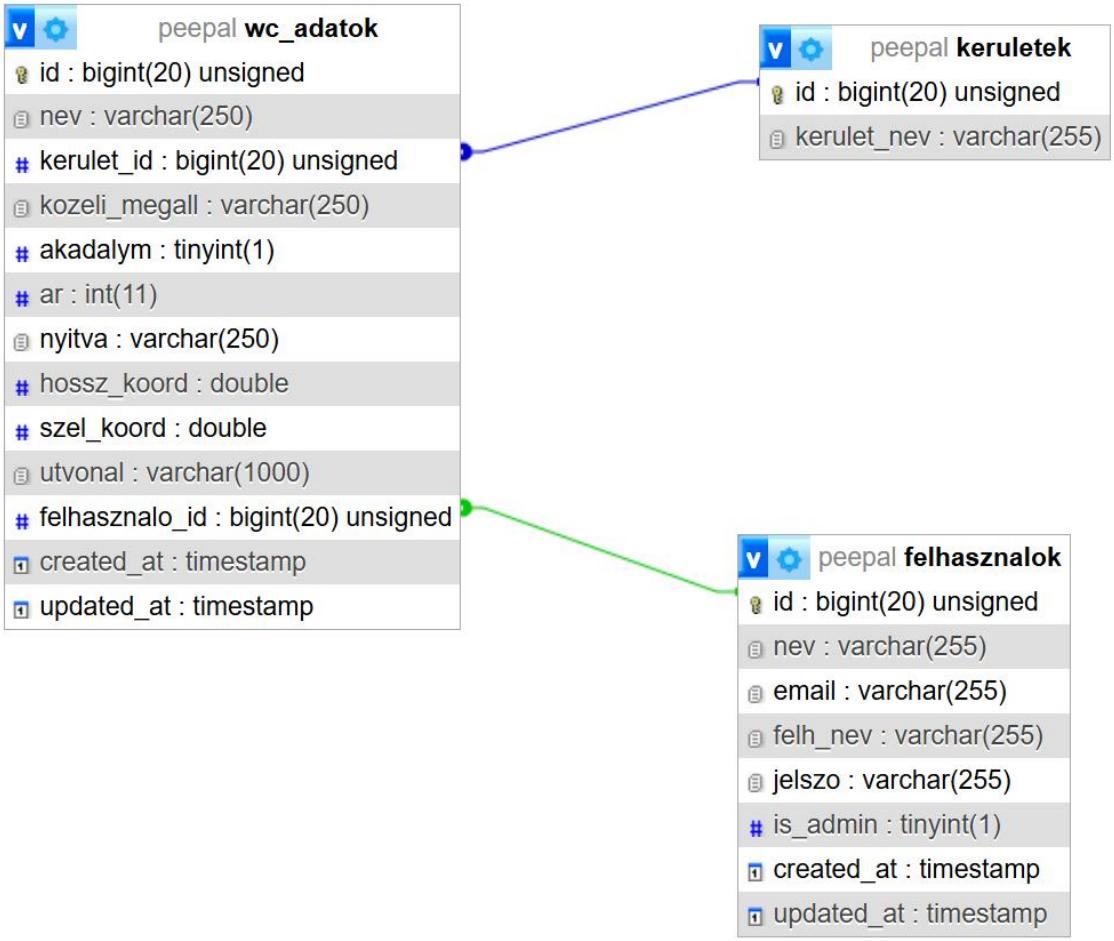
-Reszponzív megvalósítás, asztali- és mobilos felhasználásra.

## 3.3. Adatbázis, Backend és Frontend kapcsolata, algoritmusok ismertetése

A szoftver három fő alkotóelemből áll, ezek a Frontend, a Backend és az adatbázis. A következőkben az adatbázis, a Frontend és Backend komponensek működéséről és kapcsolatáról lesz szó.

## 1. Adatbázis

Elsőként az adatbázisunkat mutatjuk be, a táblák és adatszerkezetek részletes ismertetésével, diagrammal és kódrészlettel ábrázolva a leírtakat.




*1. ábra - Adatbázis diagram*

## A `keruletek` tábla


*2. ábra - `keruletek` tábla*

**-id (Kerület azonosító):** Ez a mező az adott kerület egyedi azonosítója a táblában. Ez az elsődleges kulcs, amely automatikusan növekszik minden új kerület létrehozásakor, és egyedi az adott kerülethez. Az egyedi azonosító segít megkülönböztetni és azonosítani a különböző kerületeket a táblában.

**-kerulet_nev (Kerület neve):** Ez a mező tárolja a kerület nevét, amely segít azonosítani és megkülönböztetni az egyes kerületeket. A kerület neve megjelenik a kerületek listázásakor, és segít a felhasználóknak megtalálni a kívánt kerületet.

**A `felhasznalok` tábla**

**-id (Felhasználó azonosító):** Ez a mező az adott felhasználó egyedi azonosítója a táblában. Az **id** mező egy szám típusú, és az elsődleges kulcsa a táblának. Automatikusan növekszik minden új felhasználó regisztrációja során, és segíti az egyedi felhasználók azonosítását és megkülönböztetését.

**-nev (Felhasználó neve):** Ez a mező tárolja a felhasználó által választott nevet. A **nev** mező VARCHAR típusú, és legfeljebb 255 karakter hosszú lehet. Ezt a nevet használja a rendszer a felhasználók azonosítására és megkülönböztetésére.

**-email (Email cím):** Ez a mező tárolja a felhasználó által megadott e-mail címet. Az **email** mező VARCHAR típusú, és legfeljebb 255 karakter hosszú lehet. Az e-mail címek segítenek a felhasználók azonosításában és a kommunikációban velük.

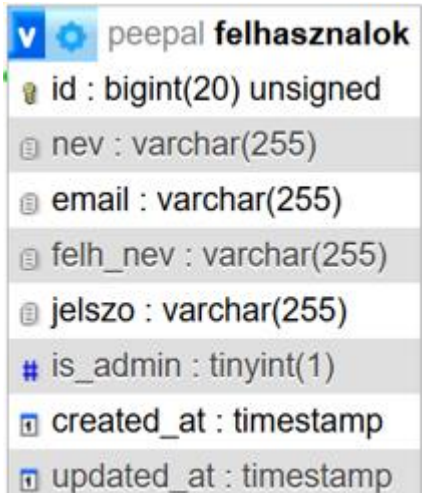
**-felh_nev (Felhasználónév):** Ez a mező tárolja a felhasználó által választott felhasználónevet. A **felh_nev** mező VARCHAR típusú, és legfeljebb 255 karakter hosszú lehet. Ezt a felhasználónevet használja a rendszer a felhasználók azonosítására és megkülönböztetésére.

**-jelszo (Jelszó):** Ez a mező tárolja a felhasználó jelszavát. A **jelszo** mező VARCHAR típusú, és legfeljebb 255 karakter hosszú lehet. Fontos, hogy a jelszavakat biztonságosan tároljuk az adatbázisban, hogy minimalizáljuk a biztonsági kockázatokat.

**-is_admin (Adminisztrátor):** Ez a mező jelzi, hogy a felhasználó adminisztrátor-e. A **is_admin** mező BOOLEAN típusú, és alapértelmezés szerint false értéket kap. Ez segít megkülönböztetni az adminisztrátorokat a normál felhasználóktól.

**-created_at (Létrehozás ideje):** Ez a mező rögzíti a felhasználó létrehozásának idejét. A **created_at** mező DATETIME típusú, és az adott időpontot rögzíti minden új felhasználó létrehozása során.

**-updated_at (Frissítés ideje):** Ez a mező rögzíti a felhasználó adatainak utolsó frissítésének idejét. Az **updated_at** mező DATETIME típusú, és az adott időpontot rögzíti minden alkalommal, amikor a felhasználó adatai frissülnek.




*3. ábra - `felhasznalok` tábla*

**A `wc_adatok` tábla**


*4. ábra - `wc_adatok` tábla*

**-id (WC adat azonosító):** Ez a mező az adott WC adat egyedi azonosítója a táblában. Ez az elsődleges kulcs, amely automatikusan növekszik minden új WC adat létrehozásakor, és egyedi az adott WC adat számára.

**-nev (WC neve):** Ez a mező tárolja a WC nevét, amely segít azonosítani és megkülönböztetni az egyes WC adatokat. A WC neve megjelenik a WC adatok listázásakor, és segít a felhasználóknak megtalálni a kívánt WC-t.

**-kerulet_id (Kerület azonosító):** Ez a mező az idegen kulcs a táblában, ami összekapcsolja a WC adatokat a kerületekkel. Az **kerulet_id** alapján lehet kapcsolatot kialakítani a két tábla között, és lehet látni, hogy mely kerületekhez tartoznak az egyes WC adatok.

**-kozeli_megall (Közeli megálló):** Ez a mező tárolja a WC-hez legközelebb eső megálló nevét. A **kozeli_megall** mező VARCHAR típusú, és legfeljebb 250 karakter hosszú lehet.

**-akadalym (Akadálymentes):** Ez a mező jelzi, hogy a WC akadálymentes-e. Az **akadalym** mező BOOLEAN típusú, és alapértelmezés szerint null értéket kap.

**-ar (Ár):** Ez a mező tárolja a WC használatának árát. Az **ar** mező INTEGER típusú, és alapértelmezés szerint null értéket kap.

**-nyitva (Nyitvatartás):** Ez a mező tárolja a WC nyitvatartási idejét. A **nyitva** mező VARCHAR típusú, és legfeljebb 250 karakter hosszú lehet.

**-hossz_koord (Hosszúsági koordináta):** Ez a mező tárolja a WC hosszúsági koordinátáját. A **hossz_koord** mező DOUBLE típusú.

**-szel_koord (Szélességi koordináta):** Ez a mező tárolja a WC szélességi koordinátáját. A **szel_koord** mező DOUBLE típusú.

**-utvonal (Útvonal):** Ez a mező tárolja a WC-hez vezető útvonalat. Az **utvonal** mező VARCHAR típusú, és legfeljebb 1000 karakter hosszú lehet.

**-felhasznalo_id (Felhasználó azonosító):** Ez a mező az idegen kulcs a táblában, ami összekapcsolja a WC adatokat a felhasználókkal. Az **felhasznalo_id** alapján lehet kapcsolatot kialakítani a két tábla között, és lehet látni, hogy mely felhasználók hozták létre az egyes WC adatokat.

**-created_at (Létrehozás ideje):** Ez a mező rögzíti a WC adat létrehozásának idejét. A **created_at** mező DATETIME típusú, és az adott időpontot rögzíti minden új WC adat létrehozása során.

**-updated_at (Frissítés ideje):** Ez a mező rögzíti a WC adat utolsó frissítésének idejét. Az **updated_at** mező DATETIME típusú, és az adott időpontot rögzíti minden alkalommal, amikor a WC adat frissül.

```php
public function up(): void
{
    Schema::create('keruletek', function (Blueprint $t) {
        $t -> id();
        $t -> string('kerulet_nev');
    });

    Schema::create("felhasznalok", function (Blueprint $t){
        $t -> id();
        $t -> string("nev");
        $t -> string("email");
        $t -> string("felh_nev");
        $t -> string('jelszo');
        $t -> boolean('is_admin')->default(false);
        $t -> timestamps();
    });

    Schema::create('wc_adatok', function (Blueprint $table) {
        $table->id();
        $table->string('nev', 250);
        $table->foreignId('kerulet_id') -> references('id') -> on('keruletek');
        $table->string('kozeli_megall', 250);
        $table->boolean('akadalym')->nullable();
        $table->integer('ar')->nullable();
        $table->string('nyitva', 250)->nullable();
        $table->double('hossz_koord');
        $table->double('szel_koord');
        $table->string('utvonal', 1000);
        $table->foreignId('felhasznalo_id')->nullable()->references('id')->on('felhasznalok');
        $table->timestamps();
    });
}
```

*5. ábra - Adatbázis létrehozásának kódja*

## 2. Backend

Először a Backend oldalát tekintjük át, bemutatva a fontosabb algoritmusok működését. Ahhoz, hogy érthetőbb és áttekinthetőbb legyen, kommentelt kódrészletekkel ábrázoljuk a leírtakat.

## 2.1. Controllerek

```php
public function __construct()
{
    $this->middleware('auth:api', ['except' => ['login', 'register']]);
}

public function register(Request $request)
{
    $validator = Validator::make($request->all(), [
        'nev' => 'required|string',
        'email' => 'required|email|unique:felhasznalok',
        'felh_nev' => 'required|string|unique:felhasznalok',
        'jelszo' => 'required|string|min:6',
        'jelszo_confirmation' => 'required|same:jelszo',
        'is_admin' => 'boolean'
    ]);

    if ($validator->fails()) {
        return response()->json($validator->errors(), 422);
    }

    $user = User::create([
        'nev' => $request->nev,
        'email' => $request->email,
        'felh_nev' => $request->felh_nev,
        'jelszo' => $request->jelszo,
        'is_admin' => $request->is_admin ?? false
    ]);

    return response()->json([
        'message' => 'Sikeresen regisztrált felhasználó',
        'user' => $user
    ], 201);
}
```

*6. ábra - Validálás, Profil létrehozása*

```php
protected function respondWithToken($token)
{
    return response()->json([
        'access_token' => $token,
        'token_type' => 'bearer',
        'expires_in' => config('jwt.ttl') * 60,
        'user' => auth('api')->user()
    ]);
}
```

*7. ábra - JSON Web Token*

### AuthController.php – Felhasználói hitelesítés és regisztráció

Ez a **Controller** kezeli a felhasználók regisztrációját, bejelentkezését, kijelentkezését és a JWT tokenek (JSON Web Token) frissítését. Két részt emeltünk ki, az első ábrám a validálással ellátott regisztrációt, a regisztrációs adatok kikötéseivel. A hitelesítés sikeressége után létrejön az új profil, amelyről a rendszer üzenetben értesíti a felhasználót.

A második ábrán, a már említett JWT token visszajelzése látható, amelyet azonosításra és a testreszabott mezőnevek (pl.: **felh_nev, jelszo**) kezelésére használunk. Röviden a token rendszer egyfajta biztonságos „belépőkártyaként” működik, aki rendelkezik vele - regisztrált profil - az elér bizonyos – mosdó hozzáadása – funkciókat.

**WcController.php – Mosdók kezelése**

```php
class WcController extends Controller
{
    public function index()
    {
        $mosdok = WcAdatok::with('kerulet')->get();
        return response()->json($mosdok);
    }
```

*9. ábra - Adatok lekérése kerülettel*

```php
        if (!$user->is_admin) {
            return response()->json(['message' => 'Nincs jogosultság'], 403);
        }
```

*8. ábra - Admin jogosultság*

Ebben a vezérlőben két lényegesebb dolgot érdemes kiemelni. Fontos tudnivaló az első ábrán látható lekérdezés. Egyetlen lekérdezéssel jutunk hozzá a mosdó adataihoz és a hozzá kapcsolódó kerülethez is, köszönhetően az azonosítónak. Ez azért hasznos, mert az API Frontendnek nem kell külön kérni a kerületet minden egyes illemhely adathoz.

A második ábra pedig már az említett admin jogosultság használatát mutatja. Új mosdót bármely felhasználó hozzá tud adni, de azt törölni csak admin jogosultsággal lehet, ha ezzel nem rendelkezik, kap egy **403**-as hibakódot, amely a jogosultság hiányát jelenti. Bejelentkezés hiányában **401**-es hibakód jelenik meg.

**UserController.php – Felhasználók kezelése**

```php
class UserController extends Controller
{
    public function store(Request $request)
    {
        $validatedData = $request->validate([
            'nev' => 'required|string',
            'email' => 'required|email|unique:felhasznalok',
            'felh_nev' => 'required|string|unique:felhasznalok',
            'jelszo' => 'required|string|min:6',
            'is_admin' => 'boolean'
        ]);

        try {
            $felhasznalo = User::create([
                'nev' => $validatedData['nev'],
                'email' => $validatedData['email'],
                'felh_nev' => $validatedData['felh_nev'],
                'jelszo' => $validatedData['jelszo'],
                'is_admin' => $validatedData['is_admin'] ?? false
            ]);

            return response()->json([
                'message' => 'Sikeres rögzítés',
                'data' => $felhasznalo
            ], 201);
        } catch (\Exception $e) {
            return response()->json([
                'message' => 'Hiba történt a mentés során',
                'error' => $e->getMessage()
            ], 500);
        }
    }
}
```

*10. ábra - Felhasználók kezelése*

Utolsó fontos **Controller** a felhasználók kezelésére szolgál, amely első sorban új profilok létrehozását teszi lehetővé a **store()** segítségével. Ha a szokásos validálásnak megfelel, felveszi az új adatbázisba, fontos, hogy a jelszó mező nem közvetlenül van hash-elve, azt a **User** modellben található mutator (módosító) végzi el, így automatikusan biztonságos jelszót mentünk.

Nagyon lényeges kiemelni, hogy miden vezérlő hibakezelése biztosított!

A folytatásban még két lényeges részről esik szó, ezek a modellek, illetve a routes útvonalak/végpontokról.

### 2.2. Modellek

A modellek szerepe elengedhetetlen az adatbázis táblái és az adatok közti kapcsolatok definiálásban. Három modellt hoztunk létre, ezek a **Keruletek.php**, a **WcAdatok.php** és **User.php** osztályok. Most tekintsük át a fontosabb kódrészeket, amelyeket ezek a fájlok tartalmaznak:

**Keruletek.php**

```php
<?php

namespace App\Models;

use Illuminate\Database\Eloquent\Factories\HasFactory;
use Illuminate\Database\Eloquent\Model;

class Keruletek extends Model
{
    public function wcAdatok(){
        return $this->hasMany(WcAdatok::class);
    }

    use HasFactory;

    protected $table = 'keruletek';

    public $timestamps = false;
}

```

*11. ábra - Egy kerülethez több mosdó tartozhat*

A modell a `keruletek` táblához kapcsolódik, a **hasMany()** kapcsolat segítségével érjük el, hogy egy kerülethez több mosdó is tartozhat. Mivel ez egy statikus táblának számít nincs szükség időbélyegekre, így egy false értékkel azt kiiktattuk.

**WcAdatok.php**

A **WcAdatok** modell az azonos nevű tábla rekordjait képviseli, olyan formában, hogy egy **belongsTo** kapcsolattal szabályozzuk, hogy minden illemhelyet egy kerülethez és egy felhasználóhoz soroljon. Így nem fordulhat elő az, hogy ugyanaz a mosdó több kerületben is megjelenjen, ami fizikailag lehetetlen.

```php
class WcAdatok extends Model
{
    use HasFactory;

    public function kerulet(){
        return $this->belongsTo(Keruletek::class);
    }

    public function felhasznalo(){
        return $this->belongsTo(User::class);
    }

    protected $table = 'wc_adatok';

    protected $fillable = [
        'id',
        'nev',
        'kerulet_id',
        'kozeli_megall',
        'akadalym',
        'ar',
        'nyitva',
        'hossz_koord',
        'szel_koord',
        'utvonal',
        'felhasznalo_id'
    ];

}
```

*12. ábra - Kerületek deifiniálása mosdóhoz és felhasználóhoz*

Meg kell említeni a **fillable** tömböt, amely használata nélkül egy felhasználó vagy egy támadó szabadon módosíthatna olyan mezőket, amihez nem férhetne hozzá (pl.: **is_admin, password**). Ennek használatával pedig le van szabályozva a tömegesen módosítható mezők száma, így biztonságosabb, tisztább és könnyebb az adatokat kezelni.

**User.php**

A **User.php** modell, amely a **felhasznalok** nevű adatbázistáblát reprezentálja, és egyben hitelesíthető felhasználóként is működik. Ez azt jelenti, hogy ez a modell alapja minden olyan műveletnek, amely egy regisztrált felhasználóhoz kapcsolódik (pl.: bejelentkezés, hitelesítés, jogosultságkezelés). A modell az **Authenticatable** osztályból származik, így a Laravel beépített Auth rendszerével teljesen kompatibilis. A JWT által a token alapú bejelentkezést is lehetővé teszi API-khoz. A **jelszo** mező automatikusan titkosítva (hash-elve) kerül mentésre egy módosítón keresztül - **setJelszoAttribute()** -, így a jelszavak kódolt formában tárolódnak.

```
5   use Illuminate\Database\Eloquent\Factories\HasFactory;
6   use Illuminate\Foundation\Auth\User as Authenticatable;
7   use Illuminate\Notifications\Notifiable;
8   use Laravel\Sanctum\HasApiTokens;
9   use Tymon\JWTAuth\Contracts\JWTSubject;
```

*14. ábra - Implementációk*

```
15      public function getJWTIdentifier()
16      {
17          return $this->getKey();
18      }
```

*13. ábra - JWT token visszaadása*

```
50      public function setJelszoAttribute($value)
51      {
52          $this->attributes['jelszo'] = \Illuminate\Support\Facades\Hash::make($value);
53      }
```

*15. ábra - Jelszó titkosítás (Hash)*

### 2.3. Routes (api.php)

```php
use App\Http\Controllers\UserController;
use App\Http\Controllers\AuthController;
use Illuminate\Http\Request;
use Illuminate\Support\Facades\Route;
use App\Http\Controllers\WcController;

Route::get('/mosdok', [WcController::class, 'index']);

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

Route::post('/hozzaadas', [WcController::class, 'store']);

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

Route::prefix('auth')->group(function () {
    Route::post('register', [AuthController::class, 'register']);
    Route::post('login', [AuthController::class, 'login']);
    Route::post('logout', [AuthController::class, 'logout']);
    Route::post('refresh', [AuthController::class, 'refresh']);
    Route::get('me', [AuthController::class, 'me']);
});

Route::post('/users', [UserController::class, 'store']);
```

*16. ábra - Útvonalak, Végpontok*

Utoljára maradt a végpontok, útvonalak bemutatása Laravel REST API használatával. Három fontos elemet érdemes vizsgálni, ezek a:

1. Mosdókezelés (**WcController**):

    -Mosdók lekérése (index, show), létrehozása (store), és törlése (destroy).

2. Felhasználói hitelesítés (**AuthController**):

    -Regisztráció, bejelentkezés, kijelentkezés, token frissítés és aktuális felhasználói adatok lekérése. Mindezek az auth/ útvonalon alatt érhetők el.

3. Felhasználók kezelése (**UserController**):

    -Új felhasználó hozzáadása a rendszerhez (store).

# 3. Frontend

Ábrával szemléltetve mutatjuk meg, hogyan épülnek fel a komponensek. A lehető legtisztábbra szerettük volna elkészíteni, így elérve a részletesen kidolgozott és legfőképpen könnyen hibakezelhető kódot. Ahogy a példa mutatja egy fő mappát hoztunk létre **WC_Komponens** néven az **src** mappán belül és ebben sorakoznak fel a fontos fájlok. Ezek a következők:

**-1. Bejel_Regisz**

**-2. Csempe**

**-3. Fejlec**

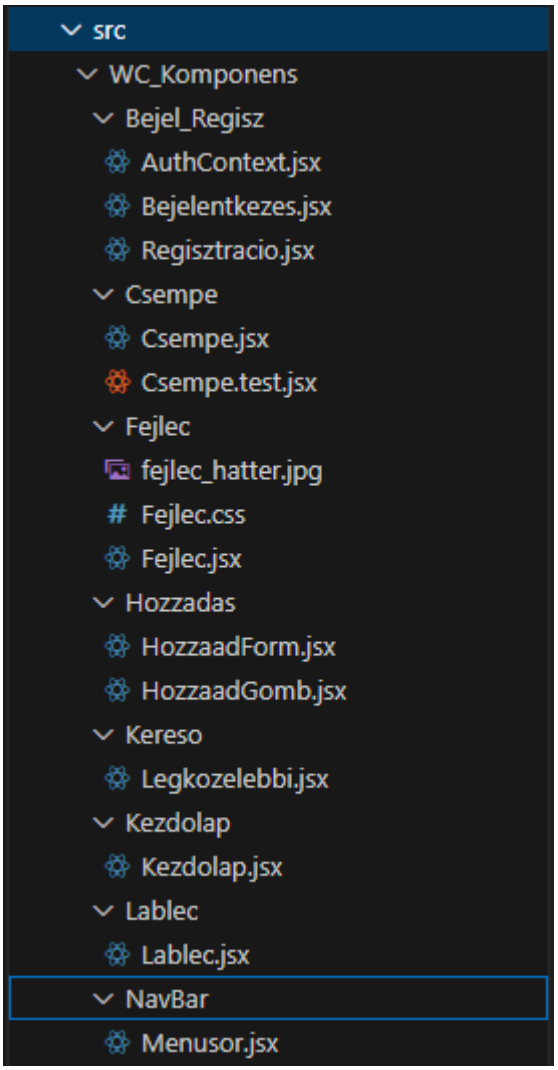
**-4. Hozzaadas**

**-5. Kereso**

**-6. Kezdolap**

**-7. Lablec**

**-8. NavBar**




*17. ábra - Frontend komponensek*

### 3.1. Bejel_Regisz – Bejelentkezés, Regisztráció, Hitelesítés

**Bejelentkezes.jsx**

Ez a komponens egy felhasználói bejelentkezési űrlapot valósít meg, amely lehetővé teszi a regisztrált felhasználók számára, hogy bejelentkezzenek az alkalmazásba. A React **useState()** függvényt használja a felhasználónév, jelszó és hibaüzenet állapotának kezelésére. A **useAuth()** hookkal a komponens eléri az **AuthContext** által biztosított „login” függvényt és a betöltési állapotot (loading), amely azt jelzi, hogy éppen zajlik-e egy bejelentkezési kísérlet.

Amikor a felhasználó rákattint a „Bejelentkezés” gombra, a **handleSubmit()** függvény fut le. Ez először ellenőrzi, hogy a szükséges mezők ki vannak-e töltve:

```
17      if (!felhNev || !jelszo) {
18        setError('Minden mező kitöltése kötelező');
19        return;
20      }
```

*19. ábra – Szükséges mezők kitöltve*

Ha minden mező rendben van, akkor meghívja az **AuthContext**-ből származó login függvényt. Sikeres bejelentkezés esetén (helyes felhasználónév és jelszó) az oldal automatikusan átirányít a főoldalra (navigate('/')):

```
22      const success = await login(felhNev, jelszo);
23      if (success) {
24        navigate('/');
25      }
```

*18. ábra – Sikeres bejelentkezés, főoldalra navigálás*

Hibás bejelentkezés esetén hibaüzenet jelenik meg az űrlap tetején, egy piros háttérrel kiemelve. A dizájn Tailwind CSS osztályokat használ, hogy modern és reszponzív megjelenést biztosítson. Erre itt egy példa:

```
29    <div className="flex justify-center items-center min-h-screen bg-yellow-100">
30      <form
31        className="bg-white p-6 rounded-lg shadow-lg w-80 space-y-4"
32        onSubmit={handleSubmit}
33      >
34        <h2 className="text-2xl font-bold text-center">Bejelentkezés</h2>
```

*20. ábra - Tailwind CSS példa*

**Regisztracio.jsx**

A **Regisztracio.jsx** komponens felhasználói fiók létrehozását teszi lehetővé. Több űrlapmezőt is kezel: név, e-mail cím, felhasználónév, jelszó és jelszó megerősítés. Ezek értékeit egy objektumként tárolja a **formData** állapotban, amely minden mező változásakor frissül.

A **handleSubmit()** függvény végzi el az alapvető mezőellenőrzéseket:

```
30    const newErrors = {};
31    if (!formData.nev) newErrors.nev = 'A név megadása kötelező';
32    if (!formData.email) newErrors.email = 'Az email megadása kötelező';
33    if (!formData.felh_nev) newErrors.felh_nev = 'A felhasználónév megadása kötelező';
34    if (!formData.jelszo) newErrors.jelszo = 'A jelszó megadása kötelező';
35    if (formData.jelszo !== formData.jelszo_confirmation) {
36      newErrors.jelszo_confirmation = 'A jelszavak nem egyeznek';
37    }
```

*21. ábra - Mezőellenőrzés*

Ha nem talál problémát, meghívja a **register** függvényt az **AuthContext**-ből. Sikeres regisztráció esetén az oldal a bejelentkezési oldalra navigál. Amennyiben a szerver Laravel típusú validációs hibákat küld vissza, a komponens ezeket olvasható formában szétválogatja:

```
49        const serverErrors = {};
50        for (const key in result.errors) {
51          if (Array.isArray(result.errors[key])) {
52            serverErrors[key] = result.errors[key][0];
53          } else {
54            serverErrors[key] = result.errors[key];
55          }
56        }
57        setErrors(serverErrors);
58      }
```

*22. ábra – Hibakezelés*

Az űrlap vizuálisan is jelzi a hibás mezőket piros kerettel, és a mező alatt megjeleníti a kapcsolódó hibaüzenetet.

**AuthContext.jsx**

Az **AuthContext.jsx** egy központi állapotkezelő, amely a hitelesítésért felelős az egész alkalmazásban. A React Context API segítségével biztosítja, hogy bármelyik komponens, amely a **useAuth()** hookot használja, elérje a hitelesítési funkciókat és állapotokat.

Főbb funkciói:

**-login(felhNev, jelszo)**: Elküldi a bejelentkezési adatokat az API-nak. Ha a válasz pozitív, elmenti a tokent a localStorage-be, majd beállítja a user állapotot.

**-register(userData)**: Elküldi a regisztrációs adatokat a Backendnek. Hibák esetén a választ visszaadja a komponensnek.

**-logout()**: Kiküldi a kijelentkezési kérést, törli a tokent, és visszaállítja a user állapotot nullára.

Az inicializálás során az **useEffect()** meghívásával megpróbálja automatikusan betölteni a felhasználói adatokat, ha már van token tárolva a böngészőben:

```
11    useEffect(() => {
12      const token = localStorage.getItem('token');
13      if (token) {
14        fetchUserData(token);
15      } else {
16        setLoading(false);
17      }
18    }, []);
```

*23. ábra - Inicializálás*

### 3.2. Csempe

**Csempe.jsx**

A **Csempe.jsx** komponens az alábbi fontos elemekkel rendelkezik:

-Adatokat kér le és jelenít meg a nyilvános mosdókról.

-Felhasználó jogosultság alapján eltérő UI-t mutat (admin jogosultság megléte).

-Kezeli a betöltési és hibás állapotokat.

-Tartalmaz egy törlési funkciót, kommunikál a Backenddel.

Pár kódrész kiemelve:

```
12    useEffect(() => {
13        const getData = async () => {
14            setLoading(true);
15            setError(null);
16            const data = await mosdokFetch();
17            if (data) {
18                setMosdok(data);
19            } else {
20                setError("Nem sikerült betölteni az adatokat.");
21            }
22            setLoading(false);
23        };
24        getData();
25    }, []);
```

*25. ábra - useEffect(), Adatok betöltése*

```
27    function mosdoTorles(mosdoId){
28        fetch("http://localhost:8000/api/mosdotorles/" + mosdoId, {
29            method: "DELETE",
30            header: {
31                "Accept" : "application/json",
32                "Content-Type" : "application/json"
33            }
34        })
35        .then(response => {
36            if (response.ok) {
37                setMosdok(prev => prev.filter(m => m.id !== mosdoId));
38            } else {
39                console.error("Törlés sikertelen");
40            }
41        })
42        .catch(error => {
43            console.error("Hiba a törlés során:", error);
44        });
45    }
```

*24. ábra - Törlés admin felhasználók számára*

```
48      if (loading) {
49          return(
50            <div className="flex justify-center items-center h-32">
51                <div
52                  role="status"
53                  className="w-12 h-12 border-4 border-gray-300 border-t-yellow-500 rounded-full animate-spin"
54                ></div>
55            </div>
56          );
57      }
58
59      if (error) {
60          return <div className="text-center text-red-600 p-4">{error}</div>;
61      }
```

*28. ábra - Mosdó betöltése*

```
63      if(user && user.is_admin){
64          return (
65              <div className="grid grid-cols-1 md:grid-cols-2 lg:grid-cols-3 gap-6 p-4">
66                  {mosdok.map((mosdo, index) => (
67                      <div key={index} className="bg-white shadow-lg rounded-xl p-6 border border-gray-200 relative">
68                          <h1 className="text-xl font-semibold text-gray-800 mb-3">{mosdo.nev}</h1>
69 >                        <div className="text-gray-600 space-y-1">···
74                          </div>
75                          {mosdo.akadalym === 1 ? <TfiWheelchair className="absolute bottom-10 right-10 w-8 h-8"/> : ""}
76 >                        <a ···
83                          </a>
84                          <button
85 >                            onClick={() => mosdoTorles(mosdo.id)}···
87                          >
88                              Törlés
89                          </button>
90                      </div>
91                  ))}
92              </div>
93          );
```

*27. ábra – Admin törlési lehetősége*

```
94      } else {
95          return(
96              <div className="grid grid-cols-1 md:grid-cols-2 lg:grid-cols-3 gap-6 p-4">
97                  {mosdok.map((mosdo, index) => (
98                      <div key={index} className="bg-white shadow-lg rounded-xl p-6 border border-gray-200 relative">
99                          <h1 className="text-xl font-semibold text-gray-800 mb-3">{mosdo.nev}</h1>
100 >                       <div className="text-gray-600 space-y-1">···
105                         </div>
106                         {mosdo.akadalym === 1 ? <TfiWheelchair className="absolute bottom-10 right-10 w-8 h-8"/> : ""}
107                         <a
108 >                           href={mosdo.utvonal}···
112                         >
113                             Útvonalterv
114                         </a>
115                     </div>
116                 ))}
117             </div>
118         )
119     }
```

*26. ábra - Útvonaltervezés*

### 3.3. Fejlec – Fejléc kódja, formázása, háttérkép

Ebben a mappában két fájlt hoztunk létre, ezek a **Fejlec.jsx** és a **Fejlec.css**, emellett ide tettük a **fejlec_hatter.jpg** fájlt, amely a főoldalon látható.

Ezekben a fájlokban főként formáztuk meg a fejlécet, köszöntő szöveget helyeztünk el és ami a legfontosabb, hogy reszponzívvá tettük az oldalt. Kisebb képernyőkön (pl.: mobil, tablet) az oldal **background-attachment** értéke **scroll**-ra változik. A navigációs sáv pedig hamburger menüvé, amely kattintással lenyílik. A köszöntő szöveg kapott egy sötétebb rétegű hátteret a könnyebb olvashatóság miatt, továbbá létrehoztunk egy **parallax** osztályt, amely annyit tesz, hogy a görgetéskor a háttérkép lassabban mozog, a tartalom pedig normál sebességen, így dizájnos és modern élményt nyújt a felhasználók számára.

```jsx
import styles from "./Fejlec.css";
import fejlecHatter from "./fejlec_hatter.jpg";

export default function Fejlec() {
  return (
    <header className={`${styles.parallax} relative w-full h-[60vh] bg-no-repeat bg-cover bg-top z-10`}
      style={{ backgroundImage: `url(${fejlecHatter})` }}
    >
      <div className="absolute inset-0 bg-black/40 flex flex-col items-center justify-center text-white text-center p-6">
        <h1 className="text-3xl font-bold mb-2 bg-black/50 p-3 rounded-lg inline-block">
          Üdvözöljük a PeePal weboldalán!
        </h1>
        <h2 className="text-lg font-medium max-w-md bg-black/40 p-3 rounded-lg">
          Találja meg az Önhöz legközelebb eső nyilvános mosdót egy kattintással (vagy kettővel)
        </h2>
      </div>
    </header>
  );
}
```

*30. ábra - Fejléc felépítése*

```css
.parallax {
    background-attachment: fixed;
    background-position: center;
    background-repeat: no-repeat;
    background-size: cover;
  }

  @media only screen and (max-width: 1366px) {
    .parallax {
      background-attachment: scroll;
    }
  }
```

*29. ábra - Reszponzivitás, parallex hatás*

### 3.4. Hozzaadas – Új mosdó hozzáadása, Hozzáadás gomb

Főbb elemek:

**-Űrlap mezők:** nev, kerulet_id, kozeli_megall, akadalym, ar, nyitva, utvonal, koordinatak.

**-Hitelesítés:** Kötelező mezők ellenőrzése, koordináták ellenőrzése.

**-POST kérés:** Fetchelés (http://localhost:8000/api/hozzaadas) felé.

**-ToolTip:** Értelmező segítség a mezők kitöltésére.

**-Római számok konvertálása:** A kerületek római számmal jelöljük, hogy dolgozni tudjunk velük egy **convertToRoman()** függvénnyel ezt átváltjuk.

**-Navigáció:** Sikeres hozzáadás után automatikusan a főoldalra navigál.

**HozzaadForm.jsx**

```jsx
export default function HozzaadForm() {
    const [nev, setNev] = useState("");
    const [kerulet_id, setKeruletId] = useState("");
    const [kozeli_megall, setKozeli] = useState("");
    const [akadalym, setAkadalym] = useState(false);
    const [ar, setAr] = useState("");
    const [nyitva, setNyitva] = useState("");
    const [utvonal, setUtvonal] = useState("");
    const [koordinatak, setKoordinatak] = useState("");
    const [errors, setErrors] = useState({});

    const navigate = useNavigate();

    const handleSubmit = async (event) => {
        event.preventDefault();

        const newErrors = {};
        if (!nev.trim()) newErrors.nev = "A név megadása kötelező.";
        if (!kerulet_id) newErrors.kerulet_id = "Kerület kiválasztása kötelező.";
        if (!kozeli_megall.trim()) newErrors.kozeli_megall = "A megálló megadása kötelező.";
        if (!koordinatak.trim()) newErrors.koordinatak = "A koordináták megadása kötelező.";
        if (!/^-?\d+\.\d+, ?-?\d+\.\d+$/.test(koordinatak)) newErrors.koordinatak = "Helytelen formátum. (pl. 47.1234, 19.1234)";

        if (Object.keys(newErrors).length > 0) {
            setErrors(newErrors);
            return;
        }

        const wcInfo = {
            nev,
            kerulet_id,
            kozeli_megall,
            akadalym,
            ar,
            nyitva,
            utvonal,
            koordinatak,
            felhasznalo_id: null
        };
```

*31. ábra - Mosdó hozzáadása*

```
45          try {
46              const response = await fetch("http://localhost:8000/api/hozzaadas", {
47                  method: "POST",
48                  headers: { "Content-Type": "application/json" },
49                  body: JSON.stringify(wcInfo)
50              });
51
52              if (!response.ok) {
53                  const errorData = await response.json();
54                  console.error("Hiba a POST kérésnél:", errorData);
55                  alert("Hiba történt az adatok mentésekor.");
56                  return;
57              }
58
59              await response.json();
60              setNev("");
61              setKeruletId("");
62              setKozeli("");
63              setAkadalym(false);
64              setAr("");
65              setNyitva("");
66              setUtvonal("");
67              setKoordinatak("");
68              setErrors({});
69              navigate("/");
70
71          } catch (error) {
72              console.error("Hálózati hiba:", error);
73              alert("Nem sikerült csatlakozni a szerverhez.");
74          }
75      };
```

*32. ábra - Fetchelés*

```
86      const convertToRoman = (num) => {
87          const roman = { M: 1000, CM: 900, D: 500, CD: 400, C: 100, XC: 90, L: 50, XL: 40, X: 10, IX: 9, V: 5, IV: 4, I: 1 };
88          let str = "";
89          for (let i in roman) {
90              const q = Math.floor(num / roman[i]);
91              num -= q * roman[i];
92              str += i.repeat(q);
93          }
94          return str;
95      };
```

*33. ábra - Kerületszám konvertálása*

**HozzaadGomb.jsx**

```jsx
export default function HozzaadGomb() {
  const { user } = useAuth();

  if (user) {
    return (
      <Link
        to="/hozzaadas"
      >
        <Plus size={24} />
      </Link>
    );
  } else {
    return (
      <Link
        to="/bejelentkezes"
      >
        <Plus size={24} />
      </Link>
    );
  }
}
```

*34. ábra – Hozzáadás gomb*

### 3.5. Kereso – Legközelebbi mosdó keresése

**Legkozelebbi.jsx**

Fontos elemei:

-Helymeghatározás - **geolocation API**.

-Mosdók lekérése - **mosdokFetch().**

-Távolság kiszámítása – **calculateDistance()**.

-Legközelebbi mosdó meghatározása.

-Állapotkezelés – **useState**.

-**nearestRestroom:** Legközelebbi illemhely adatai

-**userLocation:** a felhasználó pozíciója

-**error:** hibaüzenetek

-**loading:** betöltés állapota

A komponens tartalmának lényege a legközelebbi mosdó meghatározása, annak adatainak megjelenítése.

```
const calculateDistance = (lat1, lon1, lat2, lon2) => {
  const dx = lat1 - lat2;
  const dy = lon1 - lon2;
  return dx * dx + dy * dy;
};
```

*35. ábra - Koordináta-alapú távolság*

```
      if (nearest) {
        setNearestRestroom(nearest);
      } else {
        setError("Nem találtunk elérhető mosdót a közelben.");
      }
    } catch (err) {
      setError("Hiba történt a mosdók lekérésekor.");
      console.error(err);
    } finally {
      setLoading(false);
    }
  };
```

*36. ábra - Hibakezelés*

### 3.6. Kezdolap

**Kezdolap.jsx:** Komponensek importálása: **Fejlec**, **Csempe**, **HozzaadGomb**.

*37. ábra - Legközelebbi mosdó*

```
import Fejlec from "../Fejlec/Fejlec";
import HozzaadGomb from "../Hozzadas/HozzaadGomb";

export default function Kezdolap(){
    return(
        <>
            <Fejlec />
            <Csempe />
            <HozzaadGomb />
        </>
    )
}
```

*38. ábra - Importálás*

### 3.7. Lablec

**Lablec.jsx:** A komponens feladatának főbb részei a következők:

Tartalom:

-Logó – PeePal logó a bal oldalon.

-Linkek – elérhetőségek, minden jog fenntartva.

Háttér és elrendezés

A fájl lényegi tartalma formázás, tehát Tailwind CSS-el készült footer (lábléc).

```jsx
export default function Lablec(){
    return(
        <footer className="bg-white dark:bg-gray-800">
            <div className="mx-auto w-full max-w-screen-xl p-4 py-6 lg:py-8">
                <div className="md:flex md:justify-between">
                <div className="mb-6 md:mb-0">
                    <img src="/Logo.png" className="h-12 mt-7" alt="PeePal Logo" />
                </div>
                <div className="grid grid-cols-2 gap-8 sm:gap-6 sm:grid-cols-3">
                    <div>
                        <h2 className="mb-6 text-sm font-semibold text-gray-900 uppercase dark:text-white">Kövess minket</h2>
                        <ul className="text-gray-500 dark:text-gray-400 font-medium">
                            <li className="mb-4">
                                <a href="https://git.gszi.edu.hu/magonysandormate/PeePal" className="hover:underline ">Nemes Gitea</a>
                            </li>
                            <li>
                                <a href="https://facebook.com/peepal" className="hover:underline">Facebook</a>
                            </li>
                        </ul>
                    </div>
                    <div>
                        <h2 className="mb-6 text-sm font-semibold text-gray-900 uppercase dark:text-white">Elérhetőségek</h2>
                        <ul className="text-gray-500 dark:text-gray-400 font-medium">
                            <li className="mb-4">
                                <label className="hover:underline">peepal@gmail.com</label>
                            </li>
                            <li>
                                <label className="hover:underline">+36 30 123 4567</label>
                            </li>
                        </ul>
                    </div>
                </div>
                <hr className="my-6 border-gray-200 sm:mx-auto dark:border-gray-700 lg:my-8" />
                <div className="sm:flex sm:items-center sm:justify-between">
                    <span className="text-sm text-gray-500 sm:text-center dark:text-gray-400">© 2025 PeePal™ Minden jog fenntartva.
                    </span>
                </div>
                </div>
        </footer>
    )
}
```

*39. ábra - Logó, Linkek, Tailwind CSS*

### 3.8. NavBar - Menusor.jsx:

Az utolsó komponens a Menusor.jsx, amely az oldalon található navigációs sávot foglalja magában. Manapság talán minden weboldal legfontosabb eleme ez, főként, ha szépen van elkészítve. Működése:

Felhasználói állapot szerint megjelenő menüpontok:

-Ha felhasználó be van jelentkezve, akkor a „Kijelentkezés” menüpont jelenik meg, ez a fordított esetben is igaz.

-Navigálás oldalak közt.

Mobilnézet:

-A reszponzivitásnak köszönhetően mobilon egy úgynevezett hamburger (☰) menü ikon jelenik meg, ezt lenyitva hozzáférünk minden ponthoz.

```
            <li>
              <Link to="/kereso" className="hover:bg-yellow-500 px-4 py-2 rounded-lg transition">
                Legközelebbi mosdó
              </Link>
            </li>
            <li class="hidden md:block border-l-2 border-amber-800 h-10"></li>
            <li>
              <Link to="/bejelentkezes" className="hover:bg-yellow-500 px-4 py-2 rounded-lg transition">
                Bejelentkezés
              </Link>
            </li>
            <li>
              <Link to="/regisztracio" className="hover:bg-yellow-500 px-4 py-2 rounded-lg transition">
                Regisztráció
              </Link>
            </li>
```

*41. ábra - Alap állapot (Legközelebbi mosdó, Bejelentlkezés, Regisztráció)*

```
return (
  <nav className="bg-yellow-200 text-amber-900 py-4 px-6 flex justify-between items-center shadow-md sticky top-0 w-full z-50">
    <Link to="/" className="text-2xl font-bold flex items-center">…
    </Link>

    <button
      className="md:hidden text-amber-900"
      onClick={() => setMenuOpen(!menuOpen)}
    >
      {menuOpen ? <X size={32} /> : <Menu size={32} />}
    </button>

    <ul
      className={`md:flex md:space-x-6 md:items-center text-lg ${…
      }`}
    >
      <li>
        <Link
          to="/kereso" …
          onClick={() => setMenuOpen(false)}
        >
          Legközelebbi mosdó
        </Link>
      </li>
      <li class="hidden md:block border-l-2 border-amber-800 h-10"></li>
      <li>
        <Link
          to="/bejelentkezes" …
          onClick={() => setMenuOpen(false)}
        >
          Bejelentkezés
        </Link>
      </li>
      <li>
        <Link
          to="/regisztracio" …
          onClick={() => setMenuOpen(false)}
        >
          Regisztráció
        </Link>
      </li>
    </ul>
  </nav>
);
```

*40. ábra - Bejelentkezett állapot (Legközelebbi mosdó, Kijelentkezés)*

**apiFetch.js, App.js**

Utolsó sorban a két talán legfontosabb fájlról lesz szó a Frontenden belül. Ezek az **apiFetch.js** és az **App.js**.

**apiFetch.js**

-Fetch API lekéri a mosdók listáját a http://localhost:8000/api/mosdok címről.

-Hibakezelést is tartalmaz: ha a kérés sikertelen, jegyzi a hibát a konzolra, és egy üres listát ad vissza.

Szerepe azért fontos, mert a függvény innen biztosítja a mosdók adatainak a listáját. Elengedhetetlen minden olyan oldalon, ahol keresőt, térképes nézetet használunk, ahol a mosdók listája szükséges. A hibakezelés pedig felügyeli, hogy az oldal ne omoljon össze egy API hiba esetén.

```
export async function mosdokFetch() {
    try {
        const response = await fetch('http://localhost:8000/api/mosdok');
        if (!response.ok) {
            throw new Error(`Hiba: ${response.status}`);
        }
        return await response.json();
    } catch (error) {
        console.error("Hiba történt az adatok lekérésekor:", error);
        return [];
    }
}
```

*42. ábra - apiFetch*

**App.js**

Lényegében egy **root** komponens, amely betölti az oldalakat (pl.: kezdőlap, kereső, bejelentkezés). A menüsor és a lábléc minden oldalon megjelenik. Fontos az **AuthProvider** szerepe, amely biztosítja a teljes alkalmazásban a bejelentkezett felhasználó elérését.

```
import Bejelentkezes from './WC_Komponens/Bejel_Regisz/Bejelentkezes';
import Regisztracio from './WC_Komponens/Bejel_Regisz/Regisztracio';
import Menusor from './WC_Komponens/NavBar/Menusor';
import { Route, Routes } from 'react-router-dom';
import Kezdolap from './WC_Komponens/Kezdolap/Kezdolap';
import LegkozelebbiMosdo from './WC_Komponens/Kereso/Legkozelebbi';
import HozzaadForm from './WC_Komponens/Hozzadas/HozzaadForm';
import Lablec from './WC_Komponens/Lablec/Lablec';
import { AuthProvider } from './WC_Komponens/Bejel_Regisz/AuthContext';
```

*43. ábra - Import*

```jsx
export default function App() {
  return (
    <>
      <AuthProvider>
        <Menusor />
        <div className=''>
          <Routes>
            <Route index element={<Kezdolap />}/>
            <Route path="/kereso" element={<LegkozelebbiMosdo />}/>
            <Route path="/bejelentkezes" element={<Bejelentkezes />}/>
            <Route path="/regisztracio" element={<Regisztracio />} />
            <Route path="/hozzaadas" element={<HozzaadForm />}/>
          </Routes>
        </div>
        <Lablec />
      </AuthProvider>
    </>
  );
```

*44. ábra - Routes*

# 4. Funkcionális tesztesetek, Teszteredmények dokumentációja

**Csempe komponens (Frontend Teszt)**

Fájlnév: src/WC_Komponens/Csempe/Csempe.test.jsx

**Cél:**

-A tesztelés célja annak ellenőrzése, hogy a Csempe React-komponens helyesen jeleníti meg a felületet különböző felhasználói állapotok és adatlekérési eredmények esetén (admin jogosultság, hiba, betöltés).

**Használt technológiák:**

-React Testing Library

-Jest

-Role-alapú elérhetőség ellenőrzés (pl. role="status" a spinnerhez)

**Tesztesetek:**

**1. Adminnak megjelenik a Törlés gomb:**

**Cél:** Ellenőrzi, hogy egy admin jogosultságú felhasználó számára megjelenik a „Törlés” gomb.

**Lépések:**

-Mockolt admin felhasználóval rendereljük a komponenst

-Ellenőrizzük, hogy a törlés gomb megjelenik a DOM-ban

**2. Nem adminnak nem jelenik meg a Törlés gomb:**

**Cél:** Biztosítja, hogy egy nem admin jogosultságú felhasználó ne lássa a törlés gombot.

**Lépések:**

-Mockolt sima felhasználóval rendereljük a komponenst

-Ellenőrizzük, hogy a törlés gomb nem található meg

**3. Betöltés alatt megjelenik spinner:**

**Cél:** Ellenőrzi, hogy adatlekérés során, amíg a komponens betöltődik, egy vizuális visszajelző (spinner) jelenik meg.

**Lépések:**

-A komponens lassított/mesterségesen késleltetett fetch hívással kerül renderelésre

-Várt eredmény: a DOM tartalmaz egy role="status" attribútummal rendelkező elemet

**4. Hiba esetén megjelenik hibaüzenet:**

**Cél:** Teszteli, hogy ha az adatlekérés sikertelen, megfelelő hibaüzenetet kap a felhasználó.

**Lépések:**

-A komponens hibát szimuláló fetch válasszal kerül renderelésre

-Ellenőrizzük, hogy a hibaüzenet megjelenik

**Mockolás:** A useAuth kontextus és az adatlekérő függvények külön komponensekben kerültek mockolásra. Ez lehetővé teszi, hogy minden teszteset különálló, előre meghatározott állapotból fusson le.

**Tesztelési parancs:** npm test

**Mosdó törlés teszt (Backend Teszt)**

**Tesztesetek**:

**1. test_admin_torolhet_mosdot():**

**Cél:** Ellenőrzi, hogy admin felhasználó képes-e sikeresen törölni egy mosdót.

**Lépések:**

- Admin felhasználó és egy mosdó létrehozása
- API kérés (DELETE) hitelesített adminmal
- Várt válasz: 200 OK
- Ellenőrzés, hogy a mosdó valóban törlődött az adatbázisból

**2. test_nem_admin_nem_torolhet():**

**Cél:** Biztosítja, hogy nem admin felhasználó ne tudjon mosdót törölni.

**Lépések:**

- Nem admin felhasználó és egy mosdó létrehozása
- API kérés a nem admin hitelesített felhasználóval
- Várt válasz: 403 Forbidden
- Ellenőrzés, hogy a mosdó nem törlődött az adatbázisból

**3. test_anonim_felhasznalo_nem_torolhet():**

**Cél:** Ellenőrzi, hogy be nem jelentkezett (anonim) felhasználó ne férhessen hozzá a törléshez.

**Lépések:**

- Létrehozás: egy mosdó és egy nem admin felhasználó

- API kérés hitelesítés nélkül

- Várt válasz: 401 Unauthorized

**Adatbázis frissítés:**

A **use RefreshDatabase** gondoskodik róla, hogy minden teszt tiszta adatbázis-állapotból induljon, így a tesztek egymástól függetlenül futtathatók.

**Tesztelési parancs:** php artisan test

## 5. Adatfeltöltés

Az adatbázishoz tartozó táblákat Laravel migrációval hoztuk létre (2025_04_10_091601_tabla_feltoles.php).

A tesztadatok feltöltéséhez seeder-eket használtunk (FelhasznaloFeltoltes.php, KeruletekFeltoltes.php, WcAdatokFeltoltes.php) és ezeket a DatabaseSeeder.php fájlban hívtuk meg.

A felhasználók táblát feltöltő seeder tartalmaz egy normál felhasználót és egy admint a tesztelhetőség érdekében.

```php
class FelhasznaloFeltoltes extends Seeder
{
    public function run(): void
    {
        DB::table('felhasznalok')->insert([[
            'nev' => 'Admin',
            'email' => 'admin@gmail.com',
            'felh_nev' => 'admin',
            'jelszo' => Hash::make('admin'),
            'is_admin' => true,
            'created_at' => now(),
            'updated_at' => now()
        ], [
            'nev' => 'Felhasználó',
            'email' => 'felhasznalo@gmail.com',
            'felh_nev' => 'felhasznalo',
            'jelszo' => Hash::make('felhasznalo'),
            'is_admin' => true,
            'created_at' => now(),
            'updated_at' => now()
        ]]);
    }
}
```

*45. ábra – Felhasználó feltöltés*

```php
class KeruletekFeltoltes extends Seeder
{
    public function run(): void
    {

        function arabicToRoman($number) {
            $map = array('M' => 1000, 'CM' => 900, 'D' => 500, 'CD' => 400, 'C' => 100, 'XC' => 90, 'L' => 50, 'XL' => 40, 'X'
            => 10, 'IX' => 9, 'V' => 5, 'IV' => 4, 'I' => 1);
            $returnValue = '';
            while ($number > 0) {
                foreach ($map as $roman => $int) {
                    if($number >= $int) {
                        $number -= $int;
                        $returnValue .= $roman;
                        break;
                    }
                }
            }
            return $returnValue;
        }

        for ($i = 1; $i <= 23 ; $i++) {
            DB::table('keruletek') -> insert([
                'kerulet_nev' => arabicToRoman($i) . '. kerület'
            ]);
        }
    }
}
```

*46. ábra - Kerületfeltöltés*

A kerületek táblát feltöltő seeder-ben egy arab számokat római számmá konvertáló függvényt használtunk (arabicToRoman) az esztétika érdekében.

# 6. Felhasználói dokumentáció

Üdvözöllek!

A PeePal egy olyan weboldal, amely segít abban, hogy gyorsan és egyszerűen megtaláld a hozzád legközelebbi nyilvános mosdót – bárhol is járj. Legyen szó városi sétáról, kirándulásról vagy utazásról!

Weboldalunk alapvető funkciói közé tartozik a térképes keresés, ahol megtekintheted a környékeden található nyilvános WC-k pontos helyét, nyitvatartását, legközelebbi megállót, kerületet, valamint azt is, hogy az adott mosdó akadálymentesített-e.

A PeePal lehetőséget nyújt arra is, hogy regisztrálj, és te magad is segítsd embertársaid! Ha új mosdót találsz, ami még nem szerepel az adatbázisban, könnyedén feltöltheted annak adatait, hogy a közösség is profitálhasson belőle.

A regisztrációval személyre szabott élményben lehet részed: Hozzájárulhatsz a közösségi WC-térkép bővítéséhez.

A PeePal célja, hogy megkönnyítse a mindennapokat, és biztosítsa, hogy mindig tudd, hol van a legközelebbi tiszta, biztonságos és elérhető nyilvános illemhely.

Köszönjük, hogy minket választottál – velünk mindig tudni fogod, merre van a legközelebbi megoldás!

Üdvözlettel,
*A PeePal csapata*

**Minden futtatáshoz szükséges információt a README.md fájl tartalmaz.**

**Szükséges eszközök a PeePal weboldal használatához:**

A weboldal használatához szükséges lesz egy eszköz, ami internetkapcsolattal rendelkezik, mint például:

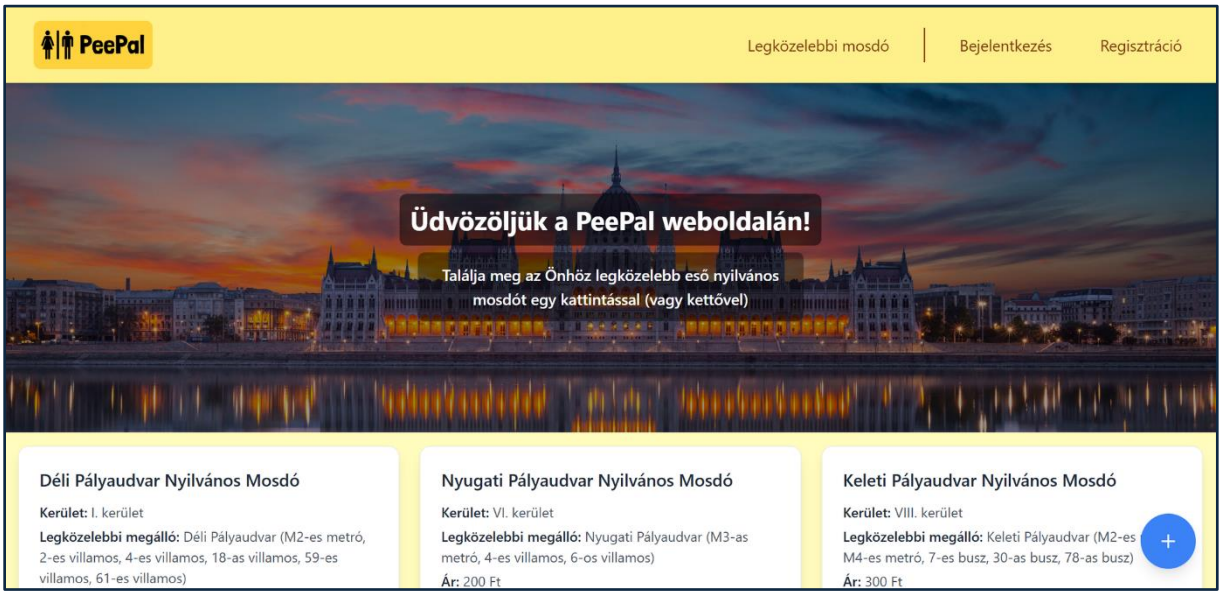
- Számítógép
- Okostelefon
- Táblagép

Ezek az eszközök mind képesek a PeePal oldal böngészésére, nyilvános illemhelyek feltöltésére.

**Weboldal eléréséhez használható böngészők:**

Az eszköz és internet mellett szükséges egy webböngésző is. Ez lehet például:

- Mozilla Firefox
- Google Chrome
- Microsoft Edge




*47. ábra - PeePal főoldal*

**Weboldal használatának részletes bemutatása:**

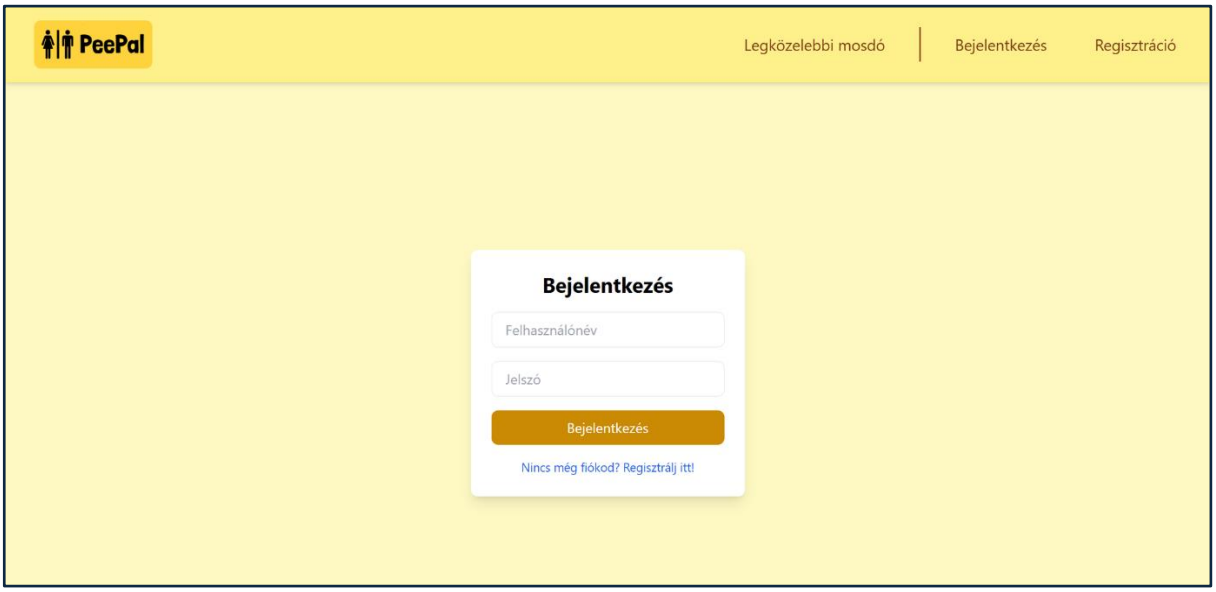
A PeePal főoldala, ahonnan tudsz navigálódni másik oldalakra. Megtalálható a PeePal logó, Legközelebbi mosdók, Bejelentkezés, Regisztráció gombok.

- A logóra kattintva vissza lehet térni a főoldalra bármikor, ha éppen egy másik oldalon jársz.
- A „Legközelebbi mosdó” gombra kattintva, az oldal megmutatja a hozzád legközelebb lévő mosdót.
- A „Bejelentkezés” gombra kattintva a beléphetsz fiókodba, ami által élvezheted a többi funkciót, amit kínál az oldal.
- A „Regisztráció” gombra kattintva regisztrálhatsz, ami szimplán pár percet vesz igénybe, és teljes valójában élvezheted az oldal használatát.
- Illetve a főoldalon megtalálod az összes nyilvános mosdót.
- A jobb alsó sarokban lévő kék ikonra kattintva lehet illemhelyet felvenni.

Minden funkció szabadon élvezhető, kivéve az illemhely felvétele, ahhoz regisztrálnia kell a felhasználónak!

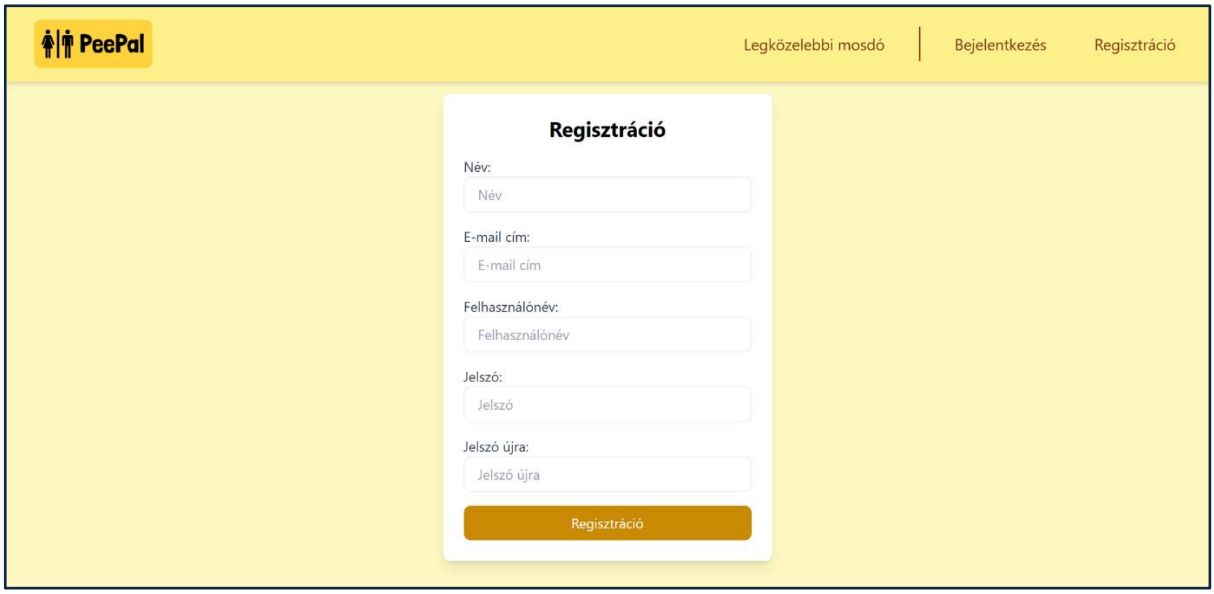
**Bejelentkezés és Regisztráció:**

A főoldalon a „Bejelentkezés” gombra nyomva lehet belépni a fiókodba.




*48. ábra - Bejelentkezés*

Meg kell adnod a felhasználónevedet, a jelszódat, és ezután rányomni a bejelentkezésre. Ha esetleg nincsen még fiókod, akkor a kékkel írt szövegre kell rákattintanod, ami átvisz a regisztrációhoz.




*49. ábra - Regisztráció*

A regisztrációnál meg kell add a személyes, és felhasználói adataidat. Név, e-mail cím, felhasználónév, jelszó, és jelszó újra. Miután ezek meg vannak regisztrálhatsz is. Fontos, hogy a főoldalról is el tudsz jutni erre az oldalra, ha rányomsz a „Regisztráció” gombra! A regisztráció által tudsz felvenni új illemhelyeket a listába. Ha ez kész, élvezheted az oldal adta lehetőségeket! Ugyan azokkal az adatokkal nem lehet többször regisztrálni, csak egyszer!

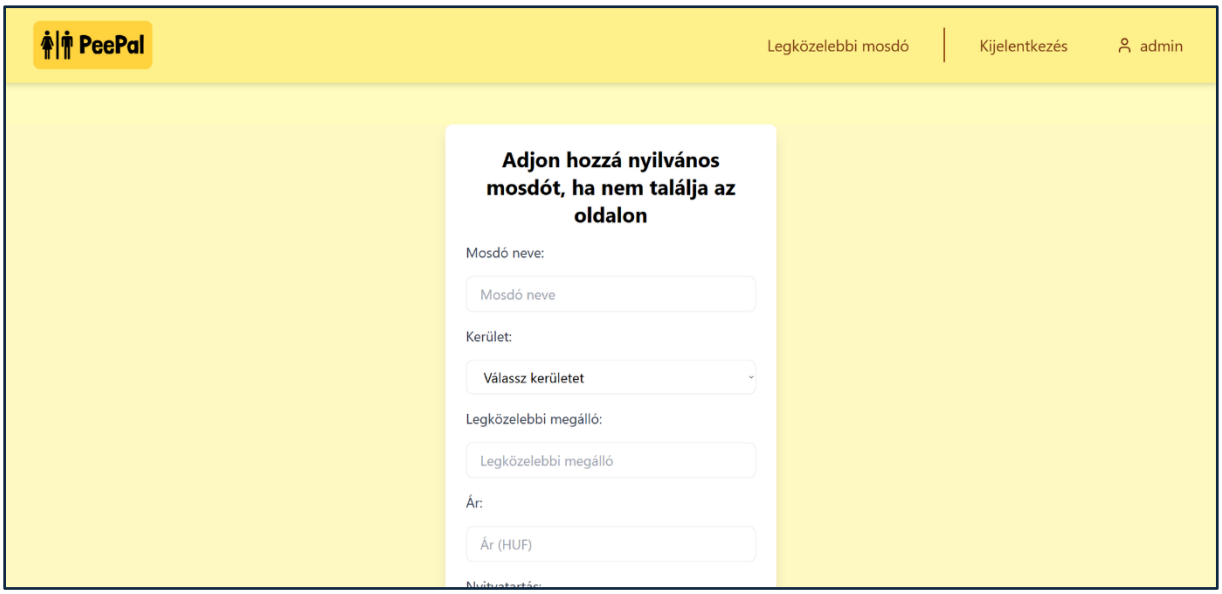
**A regisztráció előnye:**

A fiók létrehozásának rengeteg előnye van. A legfontosabb funkció az az, hogy nyilvános mosdókat tudsz hozzáadni az adatbázishoz!

Ennek a funkciónak a használatához rá kell kattintani a jobb alsó sarokban lévő kék „+” jelre, ami átvisz a mosdó felvételhez.

Az alábbi dolgokat kell megadnia a felhasználónak:

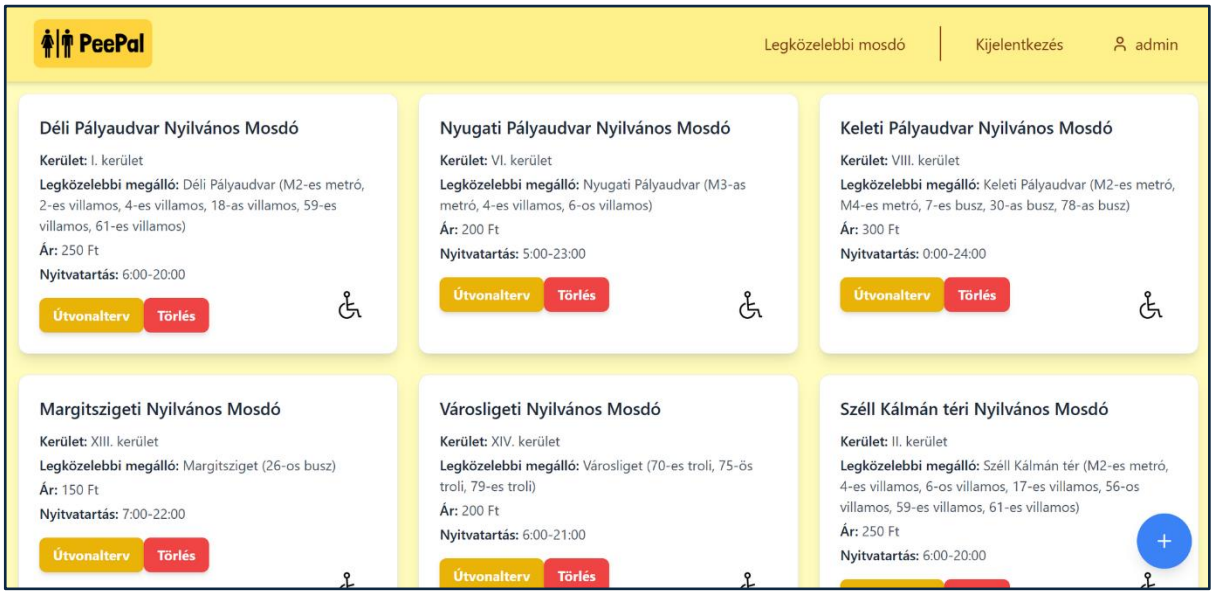
- Mosdó neve, Kerület, Legközelebbi megálló, Ár (Ft), Nyitvatartás, Akadálymentes-e, Útvonalterv link, Koordináták




*50. ábra - Mosdó hozzáadása*

Fontos, hogy muszáj mindent kitölteni, mert anélkül nem lehet működik a funkció! Mindent helyesen adjunk meg, hiszen nem lenne szép dolog félrevezetni a másikat!

Ezen az oldalon megtalálható az összes meglévő, és hozzáadott illemhely. Minden fontos információ itt megtalálható az adott helyszínről. Hol található, meddig van nyitva, mennyibe kerül, legközelebbi megállók. A mosdók jobb sarkában lévő kis ikon azt jelzi, hogy akadálymentesített.



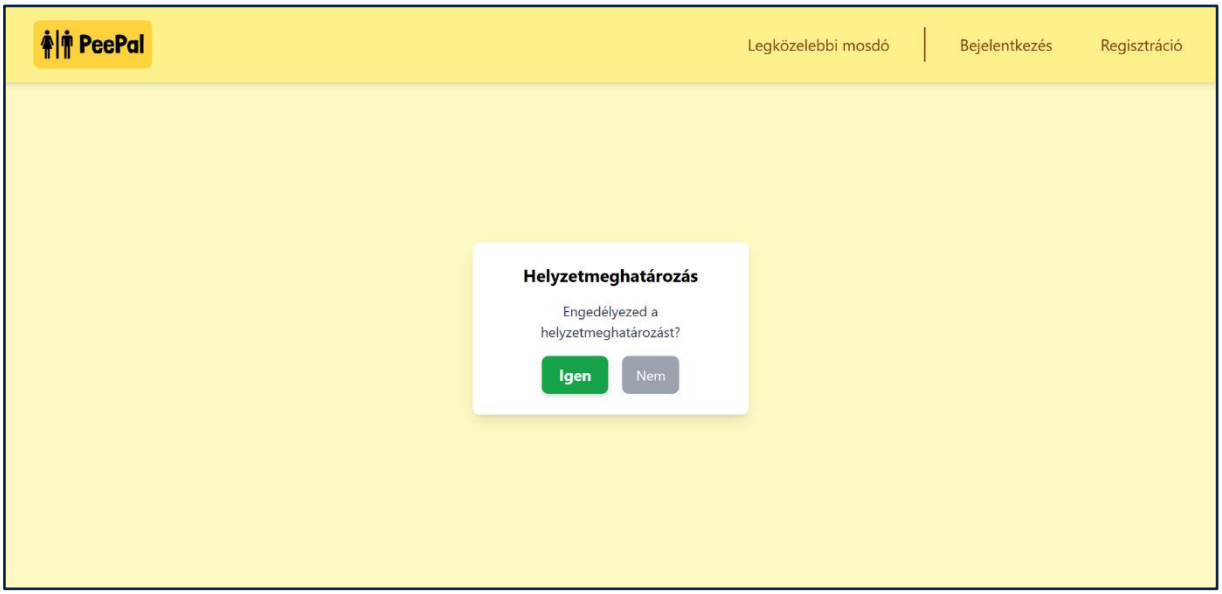

*51. ábra - Mosdók csempéi*

Két gomb is található:

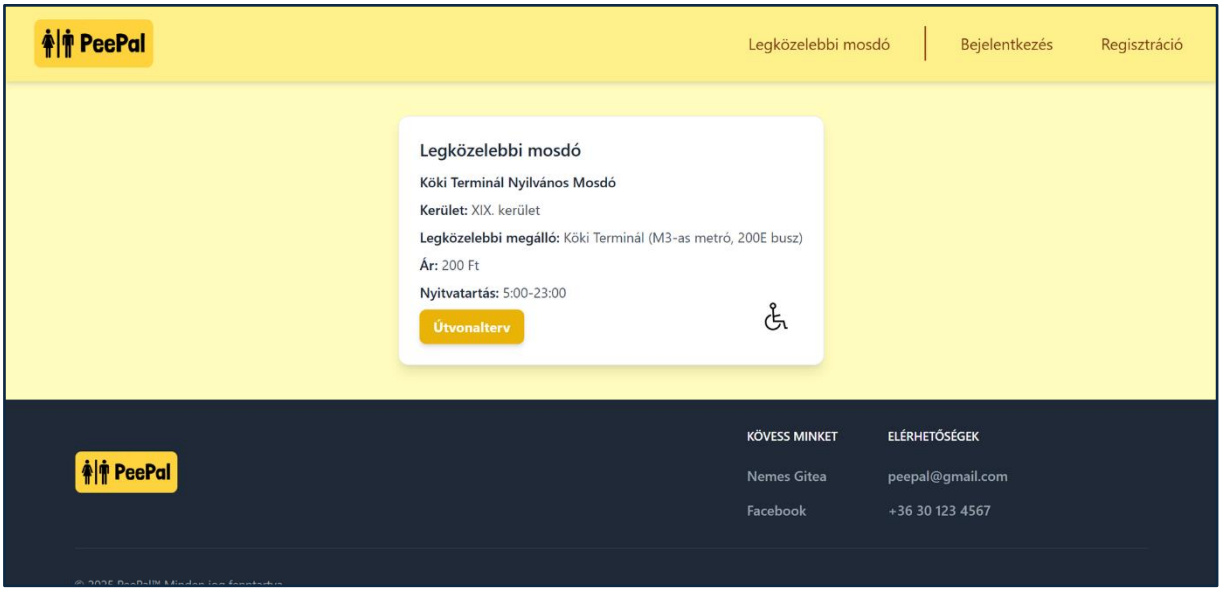
- Útvonalterv: Erre kattintva, az oldal átirányít a Google Maps-re, és megtervezi az utadat a mosdóhoz.
- Törlés: Ezzel a gombbal törölni lehet az adott illemhelyet. Ezt csak az oldal adminja éri el.

**Legközelebbi mosdó keresése:**

Az oldal rendelkezik helyzetmeghatározással. Ezáltal meg lehet találni a hozzád legközelebb lévő mosdót.




*53. ábra - Helyzetmeghatározás*




*52. ábra - Legközelebbi mosdó*

**További tudnivalók:**

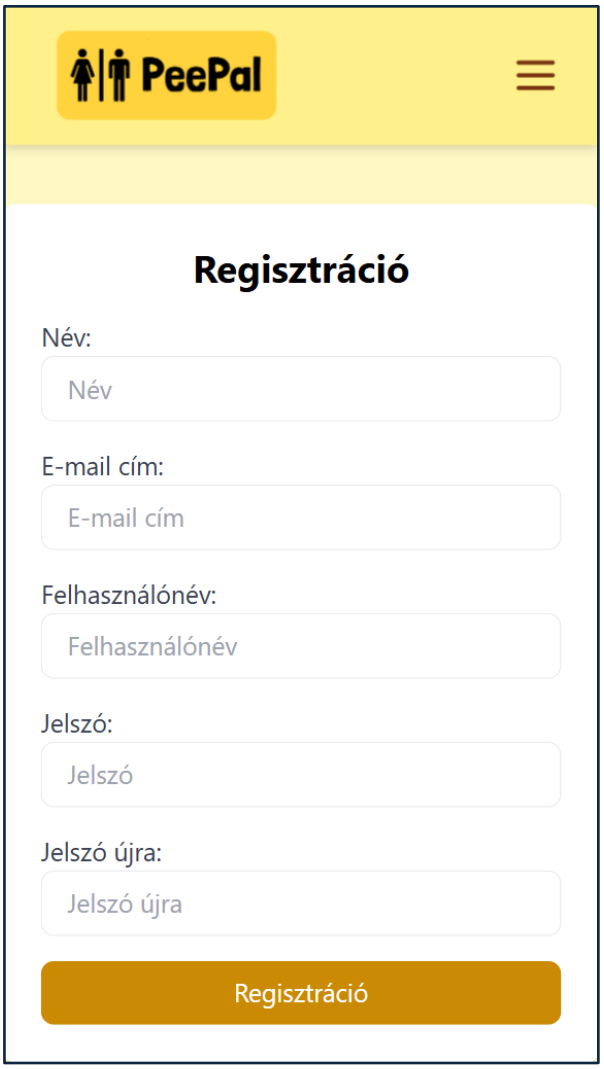
Az oldal reszponzív ezáltal működik minden eszközön! Legyen az számítógép, telefon vagy táblagép. Mindegyik eszközön tudja élvezni szolgáltatásunkat!


*54. ábra - Mobilos felület*


*55. ábra - Hamburger menü*

A telefonos menünél, hamburger menü vehető észre, így megoldva a navigációs sáv menüpontjainak elérését. Ez egyrészt szép és letisztult kinézetet biztosít, másrészt helytakarékos, nem zavarja a képernyőn való böngészést.

Így néz ki a regisztráció folyamata mobilon:




*56. ábra - Regisztráció mobilon*

# 7. Összegzés

Munkánk befejeztével elmondhatjuk, hogy büszkénk vagyunk magunkra! Elképzelésünket sikerült teljesíteni, és létrehozni. Ez egy remek alap lesz a jövőbeli munkáinkhoz, projektjeinkhez. A weboldal pontosan úgy működik ahogy elterveztük a legeleje óta. Rengeteg idő lett belefektetve, és ez reméljük meg is látszik a projekten!

Sokat tanultunk, és fejlődtünk a fejlesztés alatt. A munka alatt megszámlálhatatlan kihívás, probléma jött velünk szembe, de megoldottuk őket. Az oldalunk, ami egy nyilvános illemhely kereső, ahol az ember megtalálhatja a hozzá legközelebb lévő mosdót, természetesen nem tökéletes. Van funkció, amit gyorsan, könnyen megoldottunk, de volt, ami sajnos sok időt vett igénybe.

Megszámlálhatatlan ötletünk van arra, hogyan lehetne még jobb ez a weboldal. Jelen pillanatban a weboldal Budapesti illemhelyekre van kihegyezve. A legfontosabb, és legelső újítás egyértelműen ezt bővítené, hogy minél több városban nyújthasson segítséget az oldal. Tervezünk értékelés, és kommentelési lehetőséget is hozzáadni az mosdókhoz, hogy az ember könnyen eldönthesse a vélemények alapján, melyik az amelyik rendes, és ápolt környezet, és melyik, ami koszos, és kicsit kockázatosabb. Emellett a felhasználók jogosultságait is tervezzük alakítgatni. Fontos az is, hogy szeretnénk mindig meghallgatni az oldal használóinak a véleményeit, kritikáikat, és visszajelzéseiket. Legyen bármiről is szó, mi figyelünk, és egyből lépéseket teszünk, ha szükséges. Ezek segíthetnek az oldal fennmaradásában és sikerében. Bízunk benne, hogy a kitartó munkánk, és fejlesztéseink hozzájárulnak az oldal használatához és növekedéséhez.

# 8. Irodalomjegyzék

## Segítségül vett oldalak, tanulmányok:

*Laravel*. (2024/2025). Forrás: https://laravel.com

*React*. (2024/2025). Forrás: https://react.dev

*Stackoverflow*. (2024/2025). Forrás: https://stackoverflow.com/questions

*Szit*. (2024/2025). Forrás: https://szit.hu/doku.php?id=oktatas

*W3Schools*. (2024/2025). Forrás: https://www.w3schools.com

Tailwind. (2024/2025). Forrás: https://tailwindcss.com