A képen clipart, macska, illusztráció, rajzfilm látható

Automatikusan generált leírás

**Széchenyi István Katolikus Technikum és Gimnázium**

**Szoftverfejlesztő és -tesztelő projektfeladat**

**„REpont vizsgafeladat”**

Készítették:

Kriston-Széles Flórián,

Krizsai Tamás,

Kanta Áron

**Ózd, 2025**

Tartalomjegyzék:

1. Bevezetés
2. Felhasználói dokumentáció
   1. Rendszerkövetelmény
      1. Hardver
      2. Szoftver
   2. Webalkalmazás indítása
   3. Webalkalmazás használata
      1. Általános információk a webalkalmazásról
      2. Főoldal leírása + képernyőképek
      3. Aloldal1 leírása + képernyőkép
      4. Aloldal2…
3. Fejlesztői dokumentáció
   1. Alkalmazott fejlesztői és csoportmunka eszközök
   2. Adatbázis
   3. Frontend
   4. Backend
4. Adatbázis
   1. Modell leírása (adatmezők – típusok - leírásuk)
   2. Táblák, kapcsolatok
   3. E-K diagram
5. Frontend
   1. Alkalmazás frontend részének részletes leírása + képernyőfotók
6. Backend
   1. Alkalmazás backend részének részletes leírása + képernyőfotók
7. Tesztelés
8. Továbbfejlesztési lehetőségek
9. Irodalomjegyzék
10. **Bevezetés**
11. **Felhasználói dokumentáció**
12. **Fejlesztői dokumentáció**
13. **Alkalmazott fejlesztői és csoportmunka eszközök**

**Git/GitHub:** Project tárolása és verziómenedzselés

**Discord:** Kommunikáció

1. **Adatbázis**

**Visual Studio Code:** SQL kód megírása

**XAMPP:** SQL kód tesztelése, Apache webszerver futtatása

**MySQL:** Itt van tárolva maga az adatbázis

1. **Frontend**
2. **Backend**

**Visual Studio Code**: Laravel api, kontrollerek megírása és tesztelése

**XAMPP**: Adatbázis lokális futtatása és kezelése

**Laravel**: Keretrendszer az apihoz

1. **Adatbázis**
2. Az adatbázis azzal a céllal lett létrehozva, hogy a REpont gépeknek a nyomon követésében, és a végfelhasználóknak ezeknek a gépeknek a hely alapú mutatásában segítsen. Könnyen meg lehet tudni egy gép jelenlegi adatait, például melyik településen van, a pontos címét, illetve milyen szinten van megtelítve. Azon kívül, van még egy másik tábla, melyben különböző teljesítményekért ún. ’achievement’-eket lehet elérni, amelyek x mennyiségű üveg beváltásáért ajándékokkal jutalmazzák meg a felhasználót, pl. kuponok különböző boltoknál. Van egy tábla, melyben Magyarország települései és azoknak az irányítószámai vannak eltárolva, ez a többi adattábla könnyebb összeköttetésére szolgál. Véglegesen pedig a felhasználók is el vannak tárolva az adatbázisban, felhasználónév, jelszó, a felhasználó címe, hogy tudjuk melyik gépek vannak hozzá a legközelebb stb.
3. **Táblák Bemutatása:**
4. **Települések**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Mező | Rövid Leírás | Adattípus | Kulcs típusa | Egyéb specialitások |
| zipcode | Itt a települések irányítószáma van tárolva | int | PRIMARY KEY | NOT NULL, UNIQUE |
| settlementname | A települések neve | varchar  (75) |  | NOT NULL |

1. **Gépek**

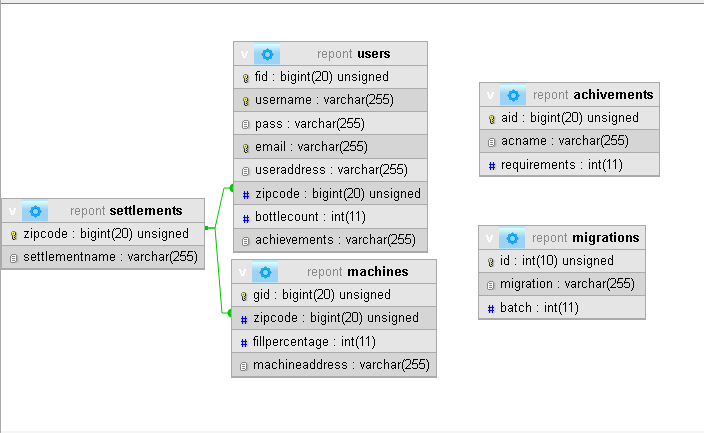
|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Mező | Rövid Leírás | Adattípus | Kulcs típusa | Egyéb specialitások |
| gid | Itt a gép azonosító száma van tárolva | int | PRIMARY KEY | NOT NULL, AUTO\_INCREMENT |
| zipcode | A gépnek a tartozási helyének az irányítószáma | int | FOREIGN KEY | NOT NULL |
| fillpercentage | A gépnek a töltöttségi százaléka | int |  |  |
| machineaddress | A gépnek a pontos címe | varchar (100) |  | NOT NULL |

1. **Felhasználók**

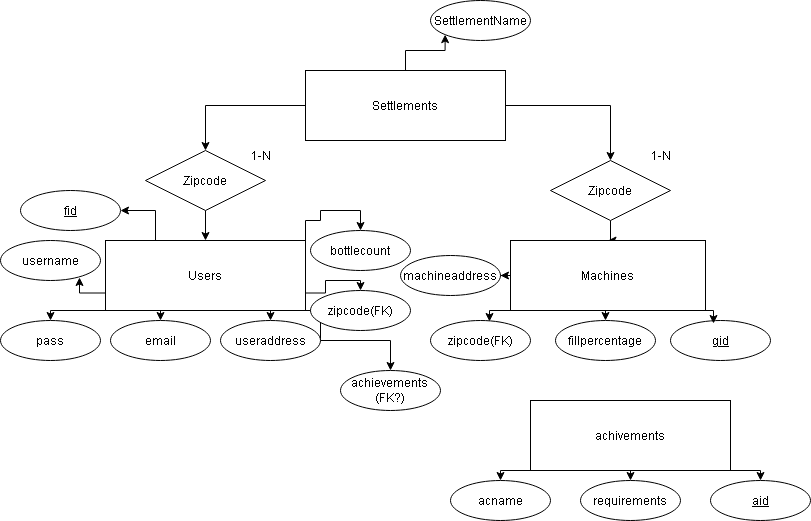
|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Mező | Rövid Leírás | Adattípus | Kulcs típusa | Egyéb specialitások |
| fid | **Itt a felhasználó azonosító száma van tárolva** | **int** | **PRIMARY KEY** | **NOT NULL, AUTO\_INCREMENT** |
| username | **A végfelhasználóknak a felhasználóneve** | **varchar**  **(30)** |  | **NOT NULL, UNIQUE** |
| pass | **A felhasználónak a jelszava van itt eltárolva** | **varchar**  **(125)** |  | **NOT NULL** |
| useraddress | A felhasználó a pontos címe | varchar  (100) |  |  |
| zipcode | A felhasználó irányítószáma | int | FOREIGN KEY | NOT NULL |
| bottlecount | Az újrahasznosított üvegek száma | int |  | NOT NULL |
| achivements | Az elért achivementek listája | varchar (250) |  |  |
| email | A felhasználó e-mail címe | varchar (65) |  | NOT NULL UNIQUE |

1. **Achievementek**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Mező | Rövid Leírás | Adattípus | Kulcs típusa | Egyéb specialitások |
| aid | Az achievement azonosítója van itt | int | PRIMARY KEY | NOT NULL, AUTO\_INCREMENT |
| acname | Az achievement neve | varchar  (75) |  |  |
| requirements | A kötelező üvegszám az eléréshez | int |  | NOT NULL |

**Kapcsolat ábra:**

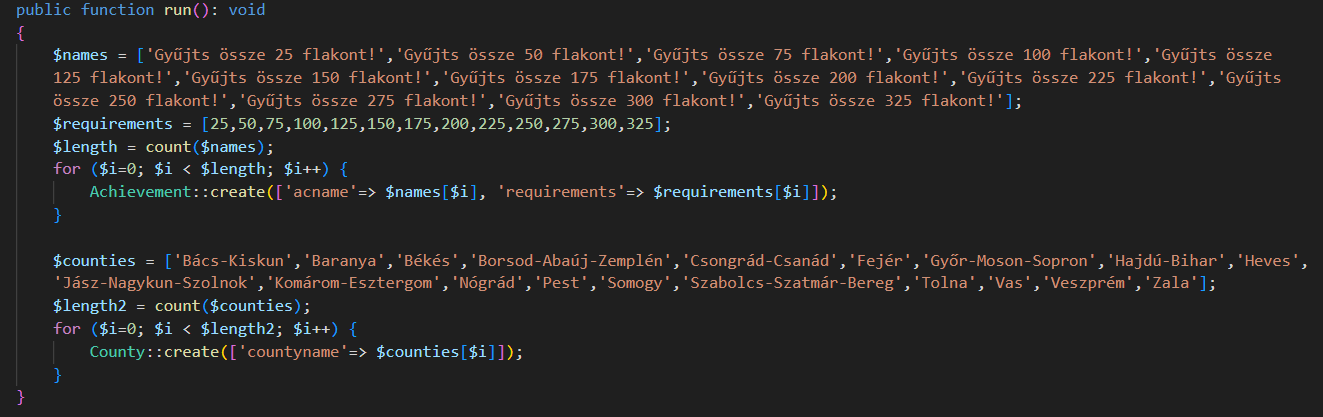
1. **ER diagram:**

****

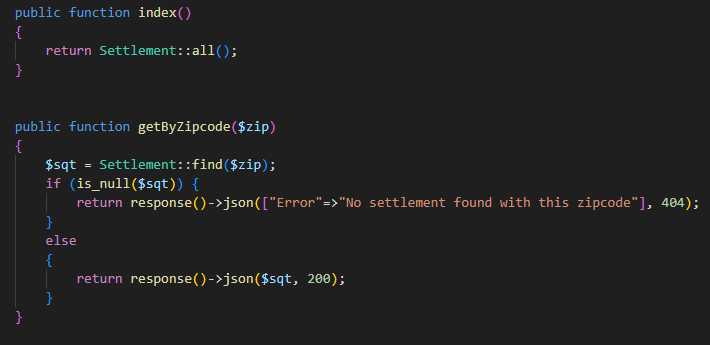
1. **Frontend**
2. **Backend**
   1. A screen shot of a computer program

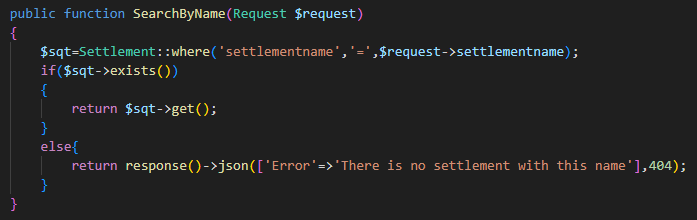
      Description automatically generatedIde valamit találjunk majd ki mer nem tudok mást szülni most.

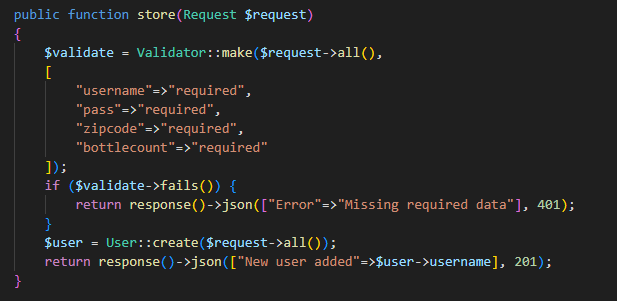
Itt látható az api.php, amiben benne vannak a megírt végpontokhoz vezető elérési útvonalak. A frontend kódja erre hivatkozik és a felhasználó vagy adminisztrátor által elrendelt műveletet végrehajtja.



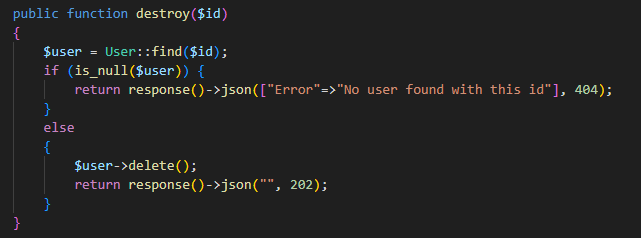
Itt látható a databaseseeder, ennek a feladata alapvetően a keretrendszer, jelen esetben a Laravel által létrehozott adattáblák feltöltése adatokkal. Ez esetben az achievementek adattáblát tölti fel előre meghatározott értékekkel. A többi adattáblát külső sql fájlból importáljuk az adatbázisba, az adatok nagy mennyisége miatt.

Az index és a getById metódus (itt getByZipcode) feladata az adatok lekérése az adatbázisból és azok továbbítása JSON formátumban. E jelenlegi specifikus szituációban a településekhez tartozó Settlement Controller kódrészlete látható. Ennek segítségével hívható le az összes adat a települések adattáblából, vagy ha csak egy adott elemre van szükség, kivitelezhető az arra való keresés, táblától függ, hogy milyen adat alapján, ez lehet irányítószám, azonosító, cím stb. Utóbbi esetben a metódus kap egy paramétert, ebben az esetben egy irányítószámot, amire aztán rákeres az adatbázisban, ha megtalálja akkor leküldi az adott település adatait és egy 200-as státuszkódot, ellenkező esetben visszaad egy hibaüzenetet, hogy nem található ilyen irányítószámú település, és ezt egy 404-es státuszkóddal is jelzi. Ez a metódus továbbítja a településeket a frontend számára, ahol megjelenítődnek.

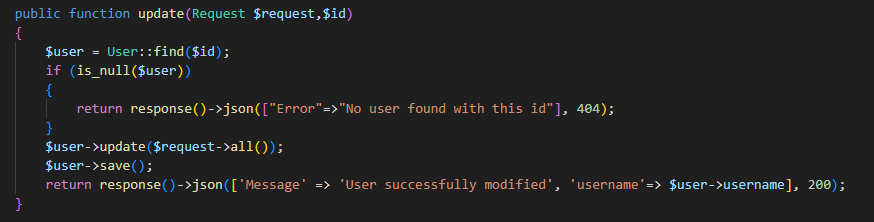
****Lent látható a search metódus. Célja ugyanaz, mint a getById eljárásnak, a különbség, hogy itt nem az elérési úton keresztül kell átadni a paramétert a metódus számára, hanem query paraméterben vagy JSON formátumban a request törzsén keresztül kapja meg. Itt sikeres kérés esetén csak az adott település továbbítódik, státuszkód nem, hiba esetén a response majdnem teljesen azonos, a hibaüzenet igazodik a metódus által keresett értékhez, itt a névhez, a státuszkód ebben az esetben is 404-es.



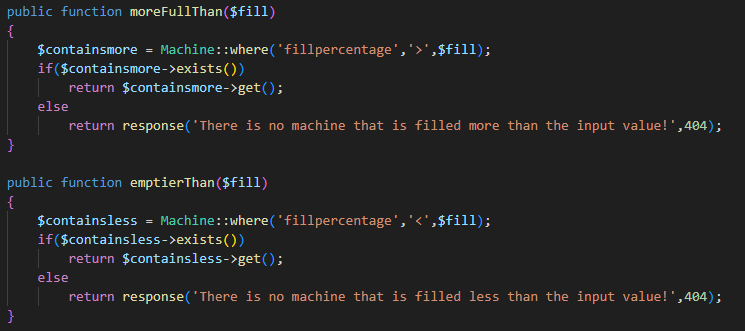
A store metódus létfontosságú eleme az API működésének, különösen az User Controllerben. Enélkül a teljes rendszer működésképtelenné válna, mivel kivitelezhetetlen lenne a felhasználók adatainak eltárolása az adatbázisban. Ez az eljárás paraméterben megkapja request formájában egy felhasználó adatait. Ezután ezen adatok validálása következik, ahol a kód ellenőrzi, hogy minden kötelezően megadandó elemet megkapott-e a metódus, ha igen akkor létrehoz egy új felhasználót, amit lement az adatbázisba, és küld egy választ JSON formátumban, 201-es státuszkóddal, amiben tudatja, hogy új felhasználó lett hozzáadva az adatbázishoz, és visszaküldi a felhasználónevet is. Hiba esetén 401-es státuszkódot továbbít, azzal az üzenettel, hogy valamelyik a kötelező adatok közül nem lett megadva. Ennek segítségével működik a frontendben a regisztráció és a bejelentkezés.



A destroy metódus feladata az adott elem törlése az adatbázisból. A jelenlegi projekt esetében ezt alkalmazni kizárólag a felhasználóknál volt értelme, máshol ritkán fordul elő, hogy az adatok törlése indokolt legyen, illetve a legtöbb esetben hibákat is idézne elő, ha máshonnan adatokat törölnénk. Alapvető felépítése leginkább a GetById metódusra hasonlít, paraméterben kap egy id-t, ami alapján rákeres a felhasználóra az adatbázisban, ha nem találja meg 404-es státuszkóddal tér vissza, és egy hibaüzenettel miszerint nem található ilyen azonosítójú felhasználó. Ha sikeresen megtalálja a felhasználót, akkor törli azt az adatbázisból, és visszatér egy 202-es státuszkóddal, üzenet nélkül. Ennek segítségével tudja a felhasználó törölni a fiókját, ha valamilyen oknál fogva már nem szeretne többet belépni.



Az update metódus felelős egy már meglévő adat megváltoztatásáért, ha például a felhasználó jelszót kíván változtatni, ez fog lefutni és kicserélni az adatbázisban az eltárolt jelszót. Működése hasonló a store metódushoz, de itt nem egy teljesen új elem kerül eltárolásra, hanem egy már meglévő módosul. Paraméterben kap egy requestet, amin keresztül az új adatok továbbítódnak, és egy id-t, ami a módosítandó elemet jelzi a kód számára. Először az kerül ellenőrzésre, hogy létezik-e felhasználó a megkapott azonosítóval, ha nem akkor 404-es státuszkódot ad vissza a metódus, azzal a státuszüzenettel, hogy nem található felhasználó ilyen id-vel. Ha viszont megtalálja a felhasználót, akkor a már meglévő adatok frissülnek és elmentődnek, majd egy státuszüzenettel visszatér a metódus, ami jelzi, hogy a felhasználó adatai módosultak, a felhasználónevet is visszaadja, illetve egy 200-es státuszkódot.

A moreFullThan és az emptierThan metódusok alapvető célja, hogy a kapott paramétert valamihez hasonlítsák, ez minden adattáblában más lehet, jelen esetben a REpont gépek töltöttségi szintjét vizsgálják, és ami oda tartozik azt visszaadják további felhasználásra. Paraméterben megkapnak egy töltöttségi szinten, ezután kiszűrik azokat, amik abba a tartományba esnek. Ha nincs ilyen akkor 404-es státuszkódot adnak, és egy üzenetet, ami leírja, hogy nem található olyan gép, aminek a töltöttségi szintje több/kevesebb volt, mint a megkapott érték. Ha találhatóak adatok, akkor visszaadja azokat státuszkód és státuszüzenet nélkül.

1. **Tesztelés**
2. **Továbbfejlesztési lehetőségek**
3. **Irodalomjegyzék**