<https://infojegyzet.hu/webszerkesztes/dokumentacio/>

<https://infojegyzet.hu/webszerkesztes/zarodolgozatmintak/recept24/recept24_vizsgaremek_dokumentacio.pdf>

tartalomjegyzék

* Bevezetés, a téma ismertetése, témaválasztás indoklása, szakmai célkitűzés
* Fejlesztői dokumentáció
* Felhasználói dokumentáció
* Összefoglalás, köszönetnyilvánítás
* Irodalomjegyzék

Fejlesztői dokumentáció

Fejlesztői környezet

A telock szoftverünk elkészítéséhez különböző fejlesztőeszközöket használunk, amelyek segítenek a fejlesztésben és az adatok kezelésében. A Visual Studio Code egy könnyen használható kódszerkesztő, amely segít a weboldal kialakításában. Az adatok kezelésére a Neon-t használjuk, ez egy serverless PostgreSQL adatbázis. A szoftvert Vercel segítségével futtatjuk. Ezeket a fejlesztőeszközöket választottuk, mert tanultunk velük, és ismerjük őket. A széles körű elérhetőségük és az, hogy ingyenesen elérhetők, tovább erősíti döntésünket. Segítségükkel hatékonyan fejleszthetünk és tesztelhetünk weboldalunkat, és könnyedén kezelhetjük az adatbázist.

Domain és tárhely

Az online elérhető szoftverünk megvalósithatósága miatt szükség volt olyan Szerver nélküli (serverless) hoszting oldalra amellyel a NextJS alapú szoftverünket futtatni tudtuk. Azért választottuk a Vercelt, mert ingyenes domain címet és futtatást biztosít

Github és Git környezet

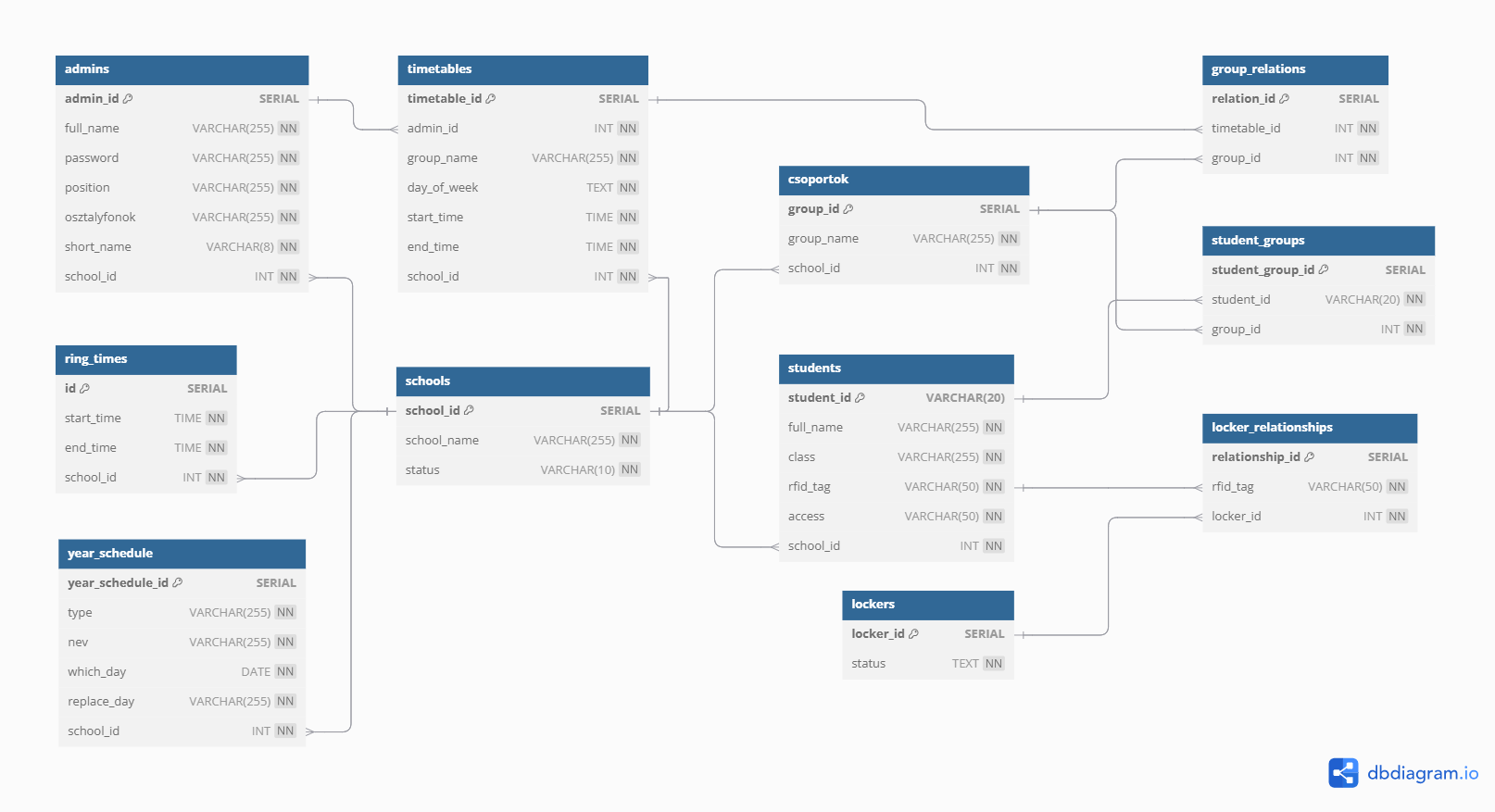
A kiírt igénynek eleget téve és közös munkánk zökkenőmentes végzése miatt, kialakítottunk egy GitHub környezetet a fejlesztésünknek. A környezetet az iskolánk által biztosított email címmel hoztuk létre, majd egy közös repository-t hoztunk létre „vizsgaremek” néven. Így a fejlesztés során végrehajtott változtatások könnyen követhetőkké váltak.

Egyéni környezeten, a https://git-scm.com/ helyről telepitett eszköz lehetőséget adott a saját gépünk és a felhős tárhely közötti Git szabályainak megfelelő kapcsolatot. Az elkészült munka, tartalmazza az adatbázist, a programkódot és a dokumentációt, a következő GitHub repositoryban érhető el: https://github.com/nagygabor123/vizsgaremek. Itt megtalálható az összes szükséges anyag, a projekt teljes körű megértéséhez.

Kialakított adatszerkezet és részletes bemutatása

A telock szoftverünk adatbázisa több táblát tartalmaz. Ezek a táblák strukturáltak és szervezettek, hogy hatékonyan kezeljék és tárolják az adatokat.

Adatbázis táblái



## 1. admins tábla

Ez a tábla az adminisztrátorok adatait tárolja.

Oszlopok:

• admin\_id: Egyedi azonosító (PRIMARY KEY, SERIAL)

• full\_name: Teljes név (VARCHAR(255), NOT NULL)

• password: Jelszó (VARCHAR(255), NOT NULL)

• position: Pozíció (VARCHAR(255), NOT NULL)

• osztalyfonok: Osztályfőnöki szerepkör (VARCHAR(255), NOT NULL)

• short\_name: Rövid név (VARCHAR(8), NOT NULL)

Az admins tábla az iskola alkalmazottainak adatait tárolja. Minden alkalmazottnak van egy egyedi azonosítója (admin\_id), teljes neve (full\_name), titkosított jelszava (password), beosztása (position), osztályfőnöki státusza (osztalyfonok) és rövid neve (short\_name). A tábla alapvető fontosságú a rendszer biztonságához, mivel itt tárolódnak a jogosultsági szintek. Az adminok kezelhetik a diákokat, csoportokat és az órarendeket.

## 2. students tábla

Ez a tábla a diákok adatait tartalmazza.

Oszlopok:

• student\_id: Egyedi diákazonosító (VARCHAR(20), PRIMARY KEY)

• full\_name: Teljes név (VARCHAR(255), NOT NULL)

• class: Osztály (VARCHAR(255), NOT NULL)

• rfid\_tag: RFID azonosító (VARCHAR(50), UNIQUE, NOT NULL)

• access: Hozzáférési jogosultság (VARCHAR(50), NOT NULL)

A students tábla az iskola diákjainak adatait tartalmazza. Minden diákot egyedi azonosító (student\_id) és RFID tag (rfid\_tag) azonosít. A tábla tárolja a diákok teljes nevét (full\_name), osztályát (class) és hozzáférési szintjét (access). Ez a tábla központi szerepet játszik a szekrények használatában és a jelenlét nyilvántartásában. Az RFID tag segítségével azonosítható a diák a szekrényeknél.

## 3. lockers tábla

Ez a tábla a szekrények állapotát tartalmazza.

Oszlopok:

• locker\_id: Egyedi azonosító (SERIAL, PRIMARY KEY)

• status: Szekrény állapota (TEXT, CHECK ("be", "ki"), NOT NULL)

A lockers tábla az iskola szekrényeinek állapotát tárolja. Minden szekrénynek van egy egyedi azonosítója (locker\_id) és állapota (status), ami lehet "be" (zárva) vagy "ki" (nyitva). A tábla nyomon követi, hogy melyik szekrények vannak jelenleg használatban. A szekrények állapota dinamikusan változik a diákok használata során.

## 4. locker\_relationships tábla

Ez a tábla a szekrények és RFID tagek kapcsolatát tárolja.

Oszlopok:

• relationship\_id: Egyedi azonosító (SERIAL, PRIMARY KEY)

• rfid\_tag: RFID azonosító (VARCHAR(50), FOREIGN KEY a students táblára)

• locker\_id: Szekrény azonosító (INT, FOREIGN KEY a lockers táblára)

Ez a tábla a diákok és a szekrények közötti kapcsolatot tárolja. Minden kapcsolatnak van egy egyedi azonosítója (relationship\_id), és tárolja a diák RFID tagját (rfid\_tag) és a szekrény azonosítóját (locker\_id). Ez a tábla biztosítja, hogy egy diák csak egy szekrényt használhasson, és egy szekrény csak egy diákhoz tartozzon. A kapcsolatok törlődnek, ha a diák vagy a szekrény törlődik.

## 5. csoportok tábla

A tanulói csoportokat tartalmazza.

Oszlopok:

• group\_id: SERIAL, PRIMARY KEY

• group\_name: VARCHAR(255), NOT NULL

A csoportok tábla az iskolai csoportokat (pl. osztályok, szakkörök) tárolja. Minden csoportnak van egy egyedi azonosítója (group\_id) és neve (group\_name). Ez a tábla szolgál alapul a tanórák és tevékenységek szervezéséhez. A csoportok segítségével logikai egységekbe lehet szervezni a diákokat különböző tevékenységekhez.

## 6. student\_groups tábla

A diákokat és csoportokat összekapcsoló tábla.

Oszlopok:

• student\_group\_id: SERIAL, PRIMARY KEY

• student\_id: VARCHAR(20), FOREIGN KEY a students táblára

• group\_id: INT, FOREIGN KEY a csoportok táblára

## Ez a tábla a diákok és a csoportok közötti kapcsolatot tárolja. Minden kapcsolatnak van egyedi azonosítója (student\_group\_id), és tárolja a diák azonosítóját (student\_id) és a csoport azonosítóját (group\_id). Egy diák több csoportba is tartozhat, és egy csoportban több diák is lehet. Ez a tábla lehetővé teszi a rugalmas csoportbeosztást.

## 7. timetables tábla

Az órarendi bejegyzéseket tartalmazza.

Oszlopok:

• timetable\_id: SERIAL, PRIMARY KEY

• admin\_id: INT, FOREIGN KEY az admins táblára

• group\_name: VARCHAR(255), NOT NULL

• day\_of\_week: TEXT, CHECK ('monday', 'tuesday', 'wednesday', 'thursday', 'friday'), NOT NULL

• start\_time: TIME, NOT NULL

• end\_time: TIME, NOT NULL

A timetables tábla az órarendeket tárolja. Minden órarendnek van egyedi azonosítója (timetable\_id), és tárolja a létrehozó admin azonosítóját (admin\_id), a csoport nevét (group\_name), a napot (day\_of\_week), valamint a kezdő és végidőt (start\_time, end\_time). Ez a tábla alapvető fontosságú az iskola napi működéséhez, mivel itt tárolódnak az összes óra időpontjai.

## 8. group\_relations tábla

Az órarendek és csoportok kapcsolatait tárolja.

Oszlopok:

• relation\_id: SERIAL, PRIMARY KEY

• timetable\_id: INT, FOREIGN KEY a timetables táblára

• group\_id: INT, FOREIGN KEY a csoportok táblára

Ez a tábla az órarendek és a csoportok közötti kapcsolatot tárolja. Minden kapcsolatnak van egyedi azonosítója (relation\_id), és tárolja az órarend azonosítóját (timetable\_id) és a csoport azonosítóját (group\_id). Ez a tábla biztosítja, hogy egy órarend több csoporthoz is tartozhasson, és egy csoportnak több órarendje is lehet.

## 9. system\_status tábla

A rendszer aktuális állapotát tartalmazza.

Oszlopok:

• id: SERIAL, PRIMARY KEY

• status: VARCHAR(10), NOT NULL

A system\_status tábla a rendszer globális állapotát tárolja. Jelenleg csak egy rekordot tartalmaz, ami a rendszer állapotát mutatja ("nyithato" vagy egyéb érték). Ez a tábla szolgál a rendszer nyitva/zárva állapotának nyilvántartására. Az érték változtatásával a rendszer adminisztrátorai befolyásolhatják a rendszer elérhetőségét.

## 10. year\_schedule tábla

A tanév eseményeit tartalmazza.

Oszlopok:

• year\_schedule\_id: SERIAL, PRIMARY KEY

• type: VARCHAR(255), NOT NULL

• nev: VARCHAR(255), NOT NULL

• which\_day: DATE, NOT NULL

• replace\_day: VARCHAR(255), NOT NULL

A year\_schedule tábla az iskola éves ütemtervét tárolja. Minden eseménynek van típusa (type), neve (nev), dátuma (which\_day) és helyettesítő napja (replace\_day). Ez a tábla tartalmazza a tanév fontos dátumait, mint a tanévkezdés és tanévzárás. Az információkat a rendszer a naptáriszinkronizációhoz használja.

## 11. ring\_times tábla

A csengetési időpontokat tárolja.

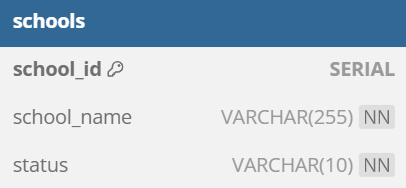
Oszlopok:

• id: SERIAL, PRIMARY KEY

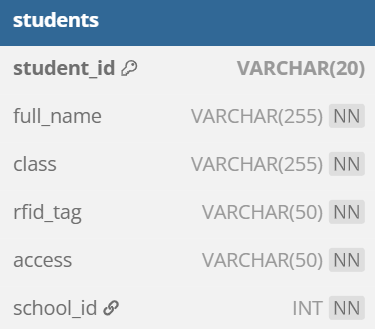
• start\_time: TIME, NOT NULL

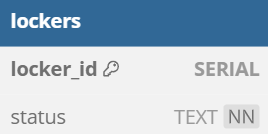
• end\_time: TIME, NOT NULL

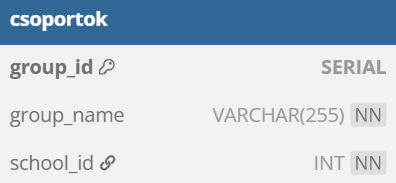
A ring\_times tábla az iskola harangszó időpontjait tárolja. Minden időpontnak van egyedi azonosítója (id), kezdő (start\_time) és végidőpontja (end\_time). Ez a tábla határozza meg a napirend szerinti harangszókat, amelyek az órák kezdetét és végét jelzik. Az időpontok pontos beállítása fontos az iskola napi ritmusának meghatározásához.

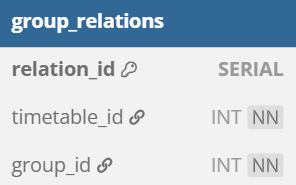




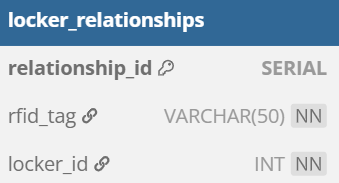




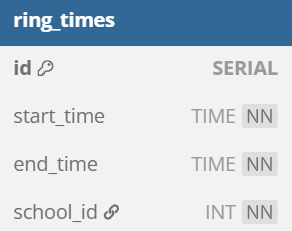


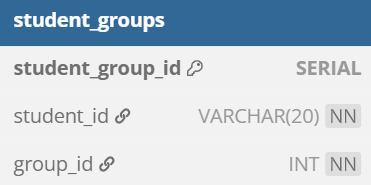












Api végpontok:

Státusz kódok:

200

201

400

404

405

500

**URL:** POST /api/setup/ascToDatabase?school\_id={az adott isokal azonosítója}

**Leírás:** Ez az API végpont az ASC órarend tervező szoftver XML fájljából nyeri ki az órarendi adatokat, majd azokat feltölti az adatbázisba. A feldolgozás során az alábbi információk kerülnek kinyerésre és tárolásra:

* **Csengetési rend**: Az egyes tanórák kezdési és befejezési időpontjai.
* **Tanárok**: A pedagógusok neve, rövidítése, valamint az osztályfőnöki szerepkör (ha elérhető).
* **Osztályok és csoportok**: A teljes osztályok és kisebb tanulócsoportok listája.
* **Órarend**: A tantárgyak, tanárok, időpontok és osztályok/csoportok kapcsolatai.

A végpont egy **multipart/form-data** kérést fogad el, amelyben a feltöltött XML fájlt dolgozza fel. A sikeres feldolgozás után az adatok továbbításra kerülnek az adatbázisba, és egy JSON választ küld vissza a mentett adatokkal.

Tartalom típusa: multipart/form-data

Query paraméter: school\_id (SZÜKSÉGES - az iskola azonosítója)

Törzs: A kérés egy XML fájlt kell, hogy tartalmazzon.

Működés:

A feltöltött XML fájlt először át ellenőrzi, hogy megfelel e a követelményeknek ha a formátum nem megfelelő vagy hiányos akkor 500-as hiba kóddal tér vissza. Ha az ellenőrzés sikeres akkor kezdi el feldolgozni az adatokat. Először a csengetésirendet szedi ki majd egy ringing nevű tömbbe tárolja el, utána a tanárokat és óraadókat gyűjti össze, egy employees tömbbe. Az órákat pedig a schedule tömbbe tárolja, utána már csak a csoportokat szedi ki egy groups nevű tömbbe. Miután az adatok rendezésével végzett ezeket a tömböket külön végpontoknak küldi amelyek feltöltik az adatbázisba ezeket az adatokat, olyan formában amely a szoftver zökkenő mentes futását lehetővé teszi. Ha az egész művelet sikeres volt 201-es üzenettel tér vissza.

Adatok feldolgozása:

A képen szöveg, képernyőkép látható

Előfordulhat, hogy a mesterséges intelligencia által létrehozott tartalom helytelen.

Csengetési rend adatok:

A képen szöveg, képernyőkép, Betűtípus, szoftver látható

Előfordulhat, hogy a mesterséges intelligencia által létrehozott tartalom helytelen.

Tanárok adatai:

A képen szöveg, képernyőkép, szoftver látható

Előfordulhat, hogy a mesterséges intelligencia által létrehozott tartalom helytelen.

Csoportok adatai

A képen szöveg, képernyőkép, szoftver, Betűtípus látható

Előfordulhat, hogy a mesterséges intelligencia által létrehozott tartalom helytelen.

Órarend adatok:

A képen szöveg, képernyőkép, szoftver látható

Előfordulhat, hogy a mesterséges intelligencia által létrehozott tartalom helytelen.

A képen szöveg, képernyőkép látható

Előfordulhat, hogy a mesterséges intelligencia által létrehozott tartalom helytelen.

**URL:** POST /api/setup/studentToDatabase?school\_id={az adott iskola azonosítója}

**Leírás:** Ez az API végpont lehetővé teszi diákok adatainak feltöltését egy CSV fájlból az adatbázisba. A fájl formátumának megfelelően kell tartalmaznia a diákok adatait. A feltöltött adatok alapján minden diáknak generálódik egy egyedi OM azonosító és egy RFID tag. Az adatok az adatbázisba kerülnek mentésre. Tartalom típusa: multipart/form-data

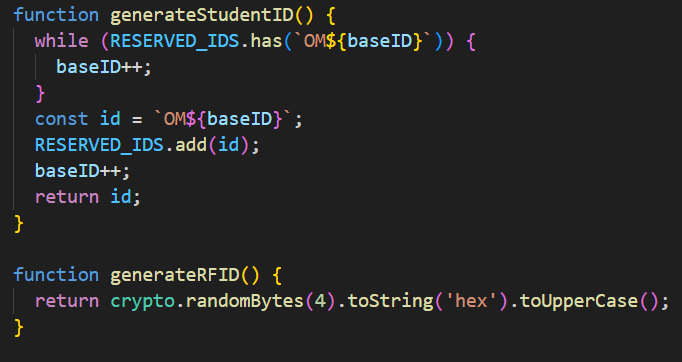
Query paraméter: school\_id (SZÜKSÉGES - az iskola azonosítója)

Törzs: A kérés egy CSV fájlt kell, hogy tartalmazzon.

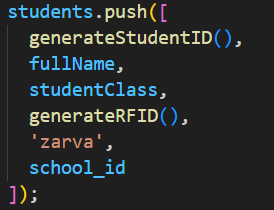
Működés:

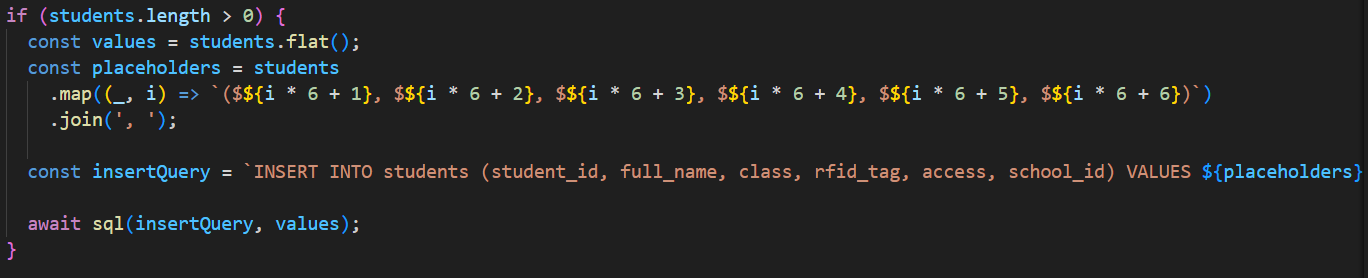
A fájl feltöltés után, ellenőrzi a CSV fájl adat szerkezetét, majd beolvassa az egészet. A beolvasás után minden diákhoz generál egy azonosítót (Például: OM11111), egy rfid azonosítót, az „access” értékét alapból „zarva”-ra állítja és még az iskola azonosítója kerül hozzáadásra. Ezeket egy tömbben tárolom el, majd ezt a tömböt egy sql paranccsal feltöltöm az adatbázisba. Sikeres feltöltés után meghívom a „uploadStudentGroups” ez a végpont az összes diáknak létrehozza a diák-csoport kapcsolatát, hogy egy diák mely csoportokban szerepel. Utána a „uploadStudLockRelations” végpont létrehozza a diák-szekrény kapcsolatot. Sikeres futás után 200-as kóddal tér vissza.

Diák azonosító és rfid azonosító generálása:

Jelenleg a RESERVED\_IDS azért kell, hogy a hardver működését is betudjuk mutatni, mivel ehhez a négy azonosítóhoz van fizikai szekrény hozzárendelve.

A students tömb:



Az adatbázis feltöltés: 

**URL:** POST /api/setup/yearChange?school\_id={az adott isokal azonosítója}

**Leírás:** Ez a végpont lehetővé teszi, hogy a tanév végén a rendszert újra indítsák és az új tanévet beállítsák.

Query paraméter: school\_id (SZÜKSÉGES - az iskola azonosítója)

Működés:

Az adatbázisban törli azokat az adatokat, amelyek az adott iskolához tartoznak, úgy, hogy a többi iskolának az adatai változnának. Ha a törlés sikertelen akkor 500-as hiba kóddal tér vissza. A törlés után az adott iskola tanév kezdésének és végének a dátumát frissíti egy évvel a „setYearStartEnd” végpont segítségével. Sikeres futás után 200-as kóddal tér vissza.

**URL:** POST /api/setup/uploadRinging?school\_id={az adott isokal azonosítója}

**URL:** POST /api/setup/uploadGroups?school\_id={az adott isokal azonosítója}

**Leírás:** Ezek a végpontok a kapott tömböt feltöltik az adatbázisba, a megfelelő iskola azonosítóval.

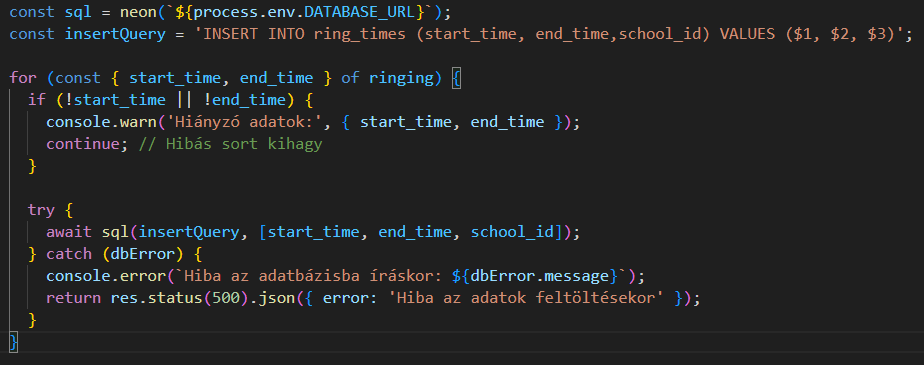
Query paraméter: school\_id (SZÜKSÉGES - az iskola azonosítója)

Törzs: ringing tömb vagy groups tömb

Működés:

A kapott tömb adatait ellenőrzik, majd feltöltik az adatbázisba. Fontos, hogy az iskola azonosítóval együtt, hogy követni tudjuk melyik csengetési rend és csoportok melyik iskolához tartozik.





**URL:** POST /api/setup/uploadEmployees?school\_id={az adott isokal azonosítója}

**Leírás:** Ez a végponto egy tömbből feltölti a tanárok adatit az adatbázisba.

Query paraméter: school\_id (SZÜKSÉGES - az iskola azonosítója)

Törzs: employees tömb

Működés:

A kapott tömb adatit ellenőri, majd az adatbázis feltöltés előtt egy hashellet alapjelszót ad minden tanárnak, amely áll: a tanár rövidített nevéből plusz „123”. Majd a tanárok bejelentkezés után megtudják változtatni az alapjelszót. És minden tanárhoz hozzárendeli az kepott iskola azonosítót. Utána egy sql paranccsal tölti fel az összes tanár adatát az adatbázisba. Sikeres futás után 200-as kóddal tér vissza. Ha az adatok feltöltése sikertelen 500-as, ha valamilyen parméter hiányzik vagy érvénytelen akkor 400-as vagy 404-es hiba kóddal tér vissza.

Az adatbázis feltöltéshez szükséges struktúra és maga a feltöltés:



**URL:** POST /api/setup/uploadTimetables?school\_id={az adott isokal azonosítója}

**Leírás:** Ez a végpont, egy tömbből feltölti a órarend adatit az adatbázisba.

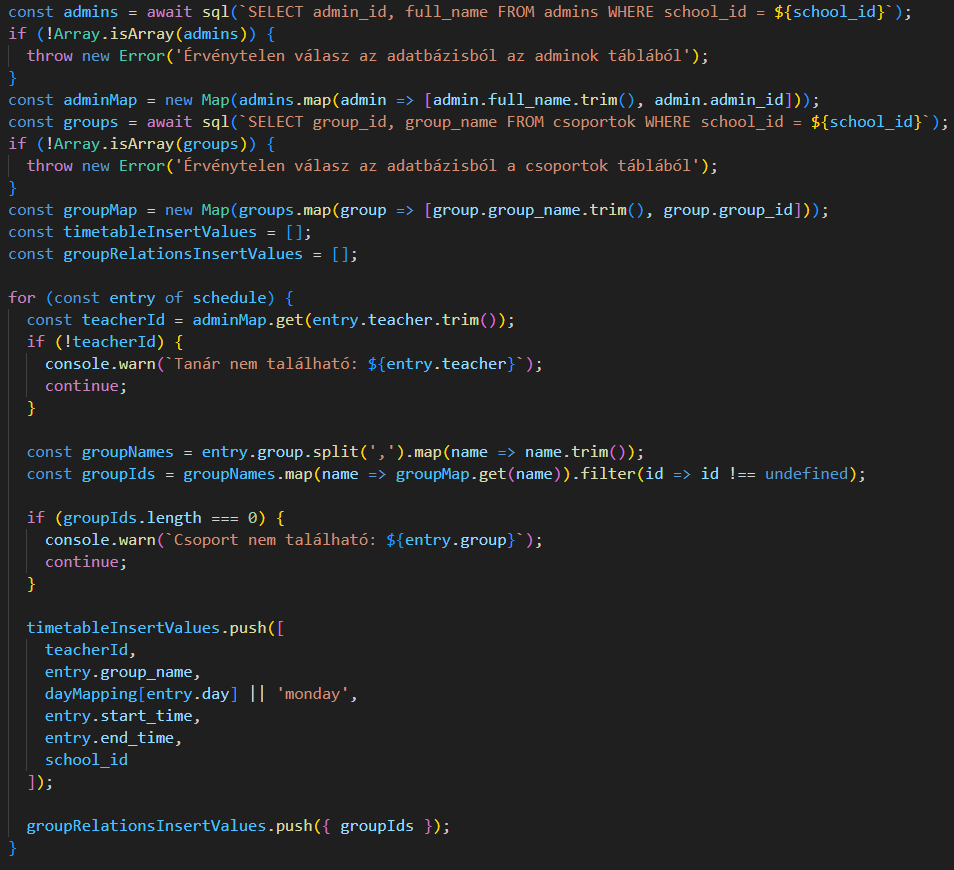
Query paraméter: school\_id (SZÜKSÉGES - az iskola azonosítója)

Törzs: schedule tömb

Működés:

Először ellenőrzi a tömb adatait, majd utána az admins táblából lekéri az adott órához tartozó tanár azonosítóját. Ha ez sikeres volt utána a schedules tömb „group” változóját vesszőnként szétválasztja majd minden elemet lekér a csoportok táblából, ha létezik az adott elem akkor a „groupRelationsInsertValues” tömbhöz adja a csoport azonosítóját, majd egy-egy sql paranccsal feltölti az órarendet a timetables táblába és az adott órához tatozó csoportokat a group\_relations táblába. Sikeres futás esetén 201-es kóddal tér vissza.

Órarend adatok rendezése és formázása:



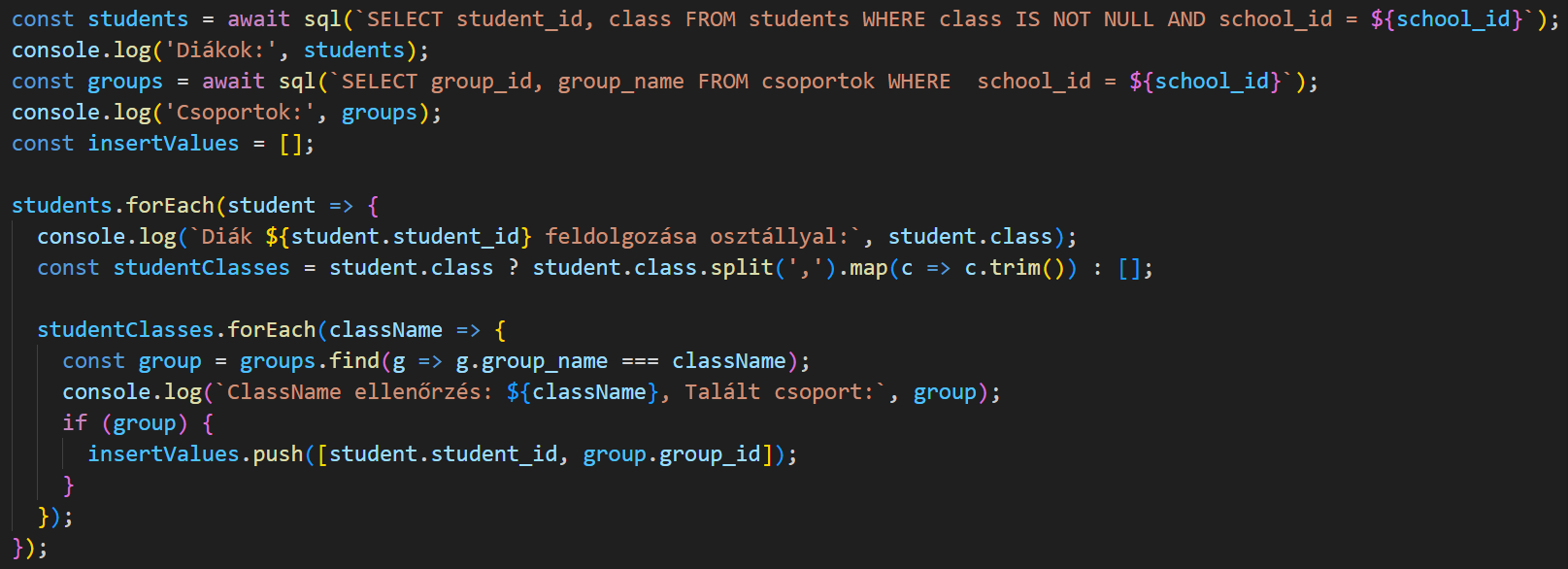
**URL:** POST /api/setup/uploadStudentGroups?school\_id={az adott isokal azonosítója}

**Leírás:** Ez a végpont, egy adott iskola összes diákjához létrehozza a diák-csoport kapcsolatokat.

Query paraméter: school\_id (SZÜKSÉGES - az iskola azonosítója)

Működés:

Lekéri azokat a diákokat akik az adott iskolához tartoznak és a „class” érékük nem üres. Majd lekéri az adott iskolához tartozó csoportokat. Utána az „insertValues” tömbbe rakja az adatokat.



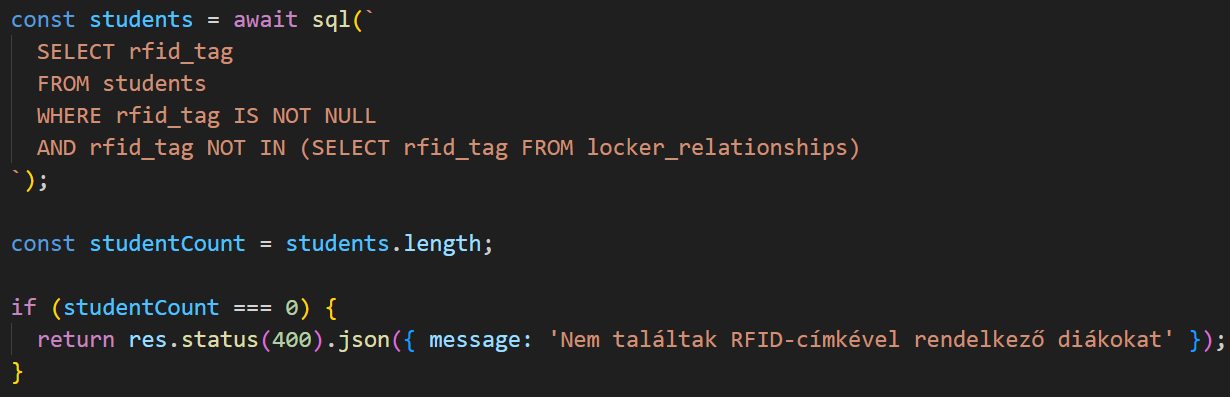
Ha a tömbbe rendezés sikeres volt akkor utána egy SQL paranccsal feltölti az adatbázisba.

**URL:** POST /api/setup/uploadStudLockRelations

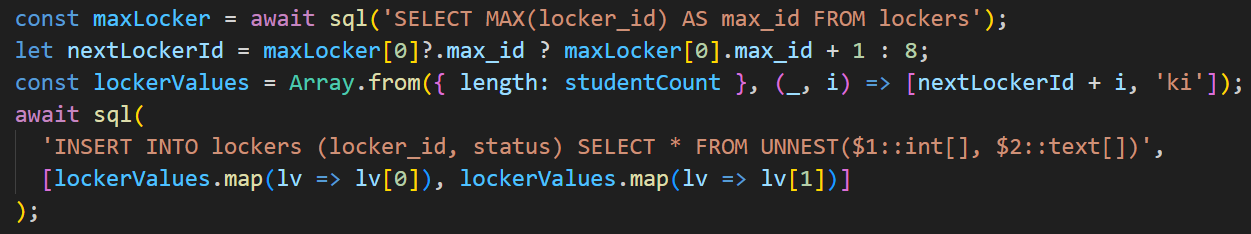
**Leírás:** Ez a végpont, egy adott iskola összes diákjához létrehozza a diák-szekrény kapcsolatokat.

**Működés:**

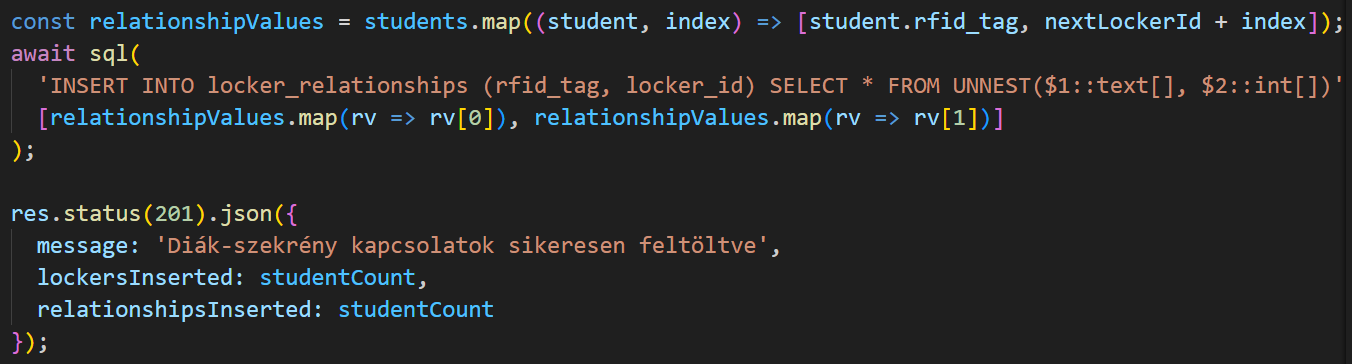
Lekéri azokat a diákokat akiknek van rfid azonosítójuk és még nincs szekrény kapcsolatuk



Majd lekéri az utolsó szekrény azonosítót és onnantól kezdve folytatja a kiosztást. Először feltölt annyi szekrényt a rendszerbe ahány diáknak kell és ezt feltölti az adatbázisba



Sikeres feltöltés után létrehozzuk a diák-szekrény kapcsolatokat, majd ezeket is feltöltjük az adatbázisba



**URL:** GET /api/timetable/getClassTimetable?className={Osztály neve}

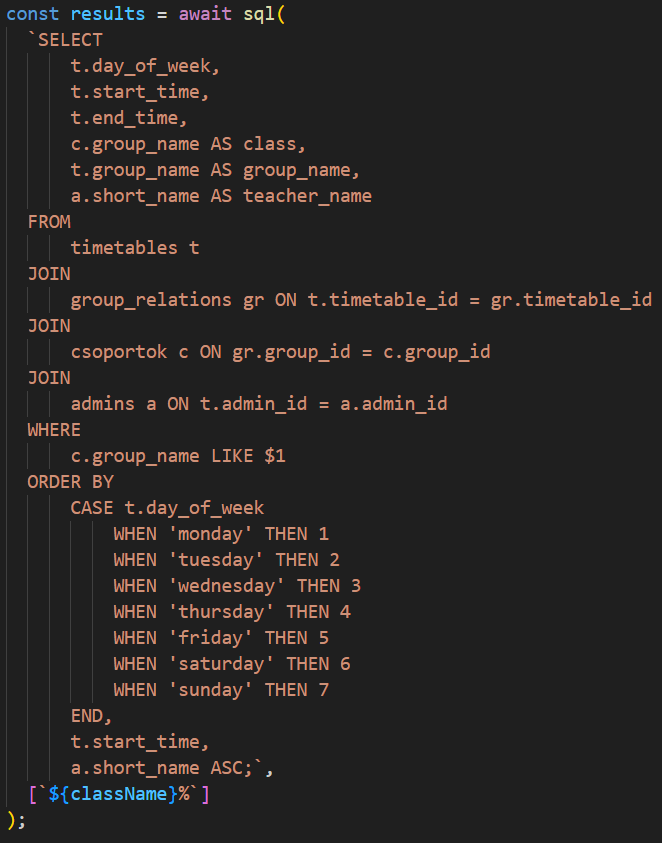
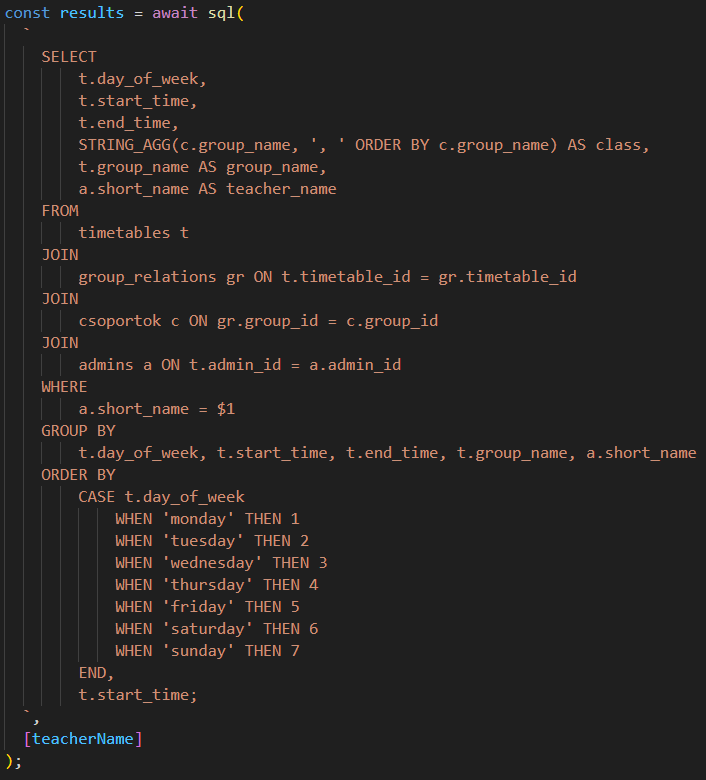
**URL:** GET /api/timetable/getTeacherTimetable?teacherName={Tanár rövidített neve}

**Leírás:** Ez a végpont lekéri egy adott osztály vagy tanár egész heti órarendjét.

Query paraméter: className (Az adott osztály neve) vagy shortName (Az adott rövidített neve)

Működés:

Osztály név vagy a tanár rövidített neve alapján lekéri az adatbázisból az egész ozstály/tanár heti órarendjét, ha csoportokra vannak bontva akkor azt is. majd sorba rendezi és visszatér egy tömbbel amelyben az órák szerepelnek.

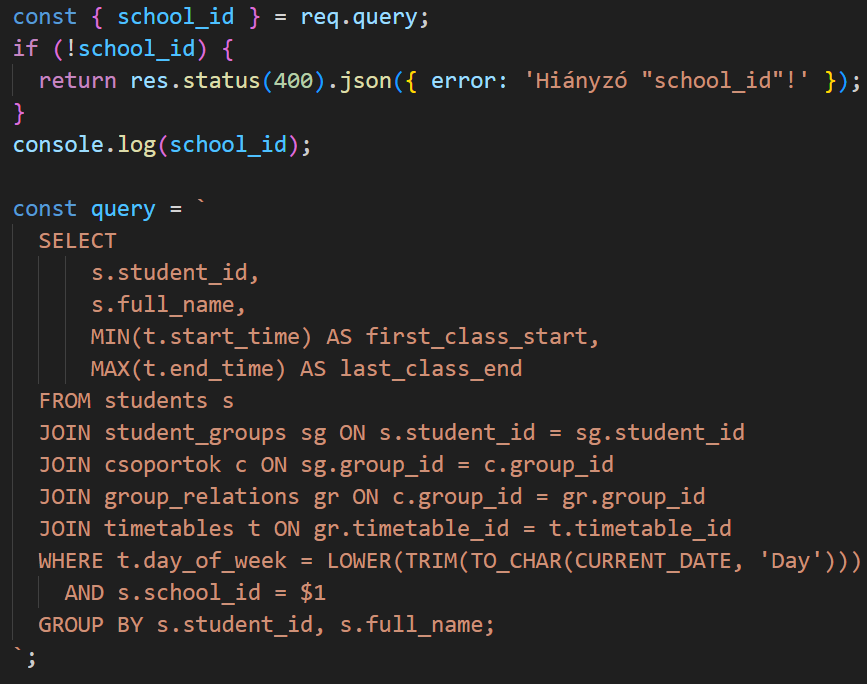
**URL:** GET /api/timetable/allScheduleStart?school\_id={ neve}

**Leírás:** Ez a végpont lekéri az adott iskolához tartozó összes diáknak az első óra kezdetét és az utolsó óra végét, egy tömbben adja vissza az adatokat.

Query paraméter: school\_id (SZÜKSÉGES - az iskola azonosítója)

Működés:

A kapott iskola azonosító alapján lekérdezi az iskolához tartozó összes diáknak, hogy mikor kezdődik az első órája és mikor fejezi be az utolsó óráját. A lekérdezésben mindig az adott napot figyeli. A lekért adatokat tömbösítve adja vissza.



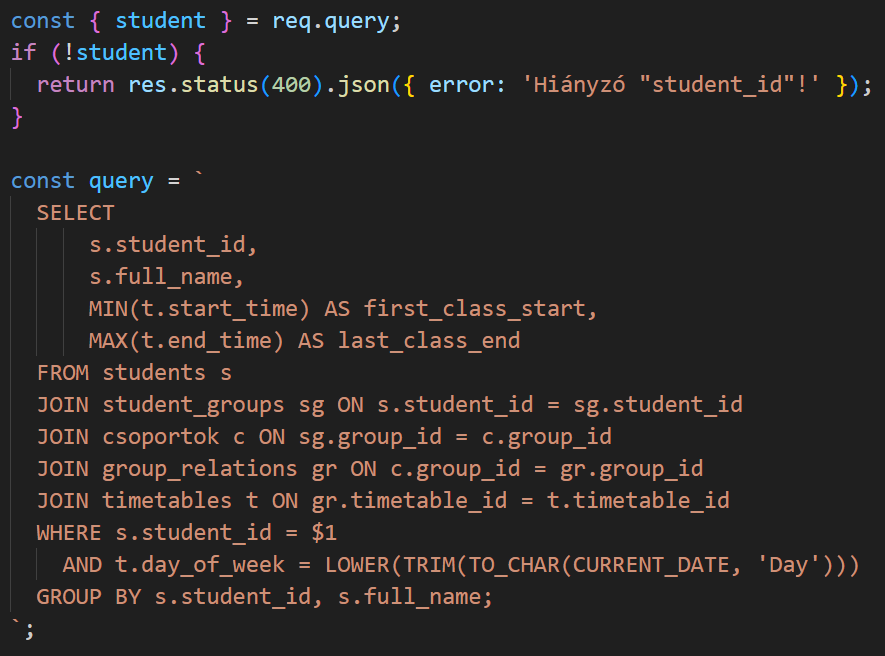
**URL:** GET /api/timetable/scheduleStart?student\_id={Diák azonosítója}

**Leírás:** Ez a végpont lekéri az adott diáknak az első óra kezdetét és az utolsó óra végét, egy tömbben adja vissza az adatokat.

Query paraméter: school\_id (SZÜKSÉGES - az iskola azonosítója)

Működés:

A kapott diák azonosító alapján lekérdezi az adott diáknak, hogy mikor kezdődik az első órája és mikor fejezi be az utolsó óráját. A lekérdezésben mindig az adott napot figyeli. A lekért adatokat json formátumban adja vissza.



**URL:** GET /api/locker/getLocker?rfid={rfid azonosító}

**Leírás:** Ez a végpont lekéri a kapott rfid azonosítóhoz tartozó szekrényt.

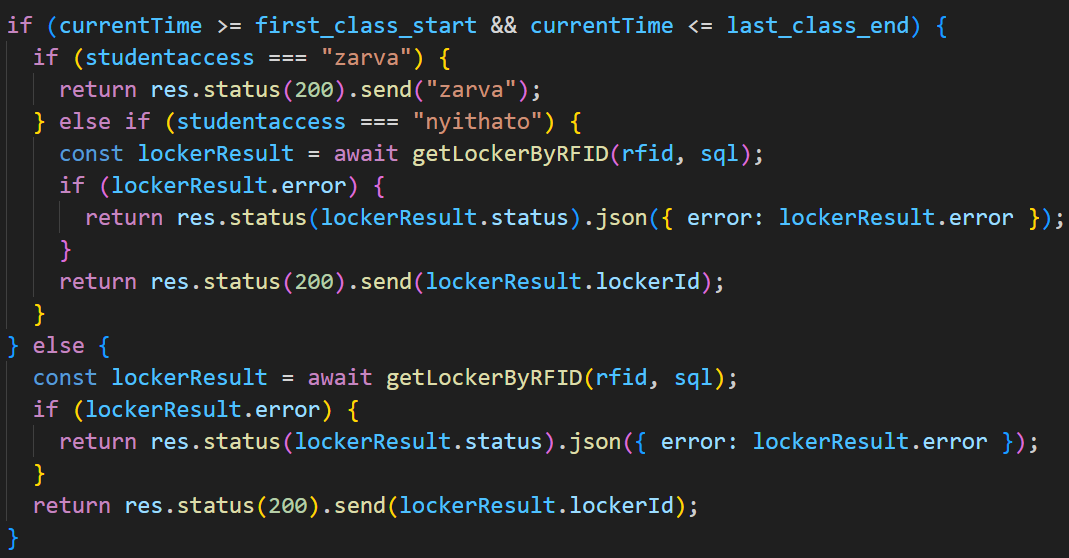
Query paraméter: rfid (Diákhoz tartozó rfid azonosító)

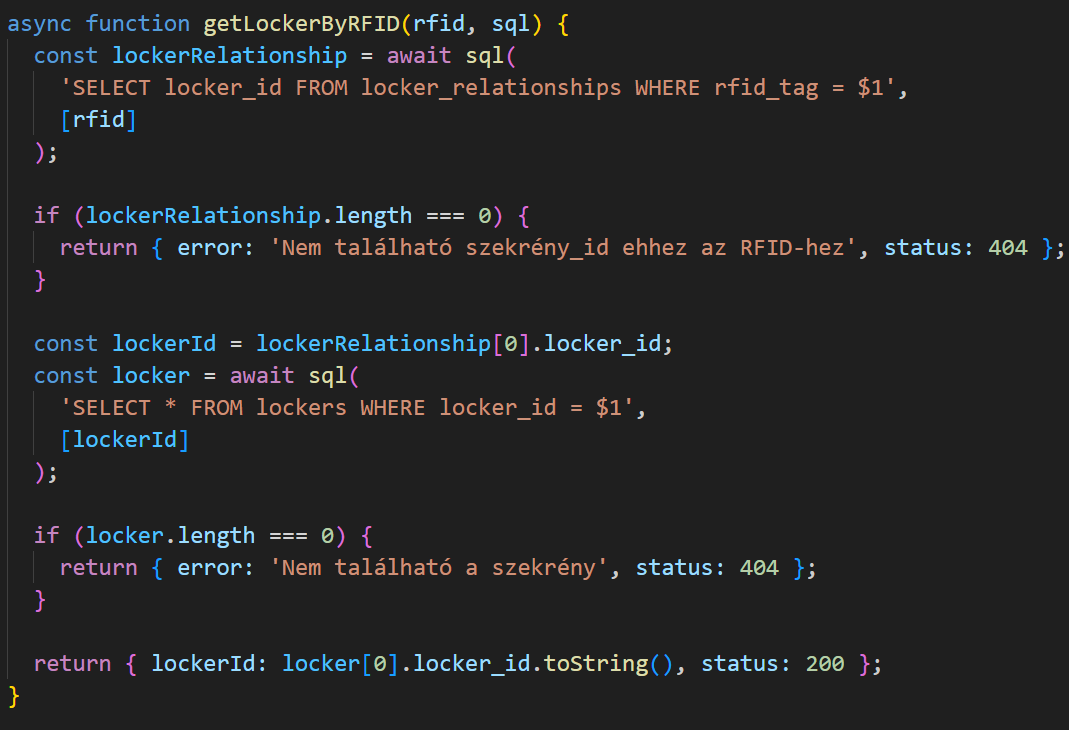
Működés:

Ez a végpont lekéri a kapott rfid azonosítóhoz tartozó diák azonosítót és a hozzáférhetőségét (nyithato/zarva) és a jelenlegi időt. Majd meghívja az „scheduleStart” végpontot.



Utána a végpont ellenőrzi, hogy a jelenlegi idő az a diák első órájának a kezdési ideje és az utolsó órájának a vége közé esik, akkor megnézi, ha a diáknak a hozzáférhetősége „nyithato”, akkor visszaadja a szekrény azonosítóját, ha „zarva” akkor csak egy „zarva” üzenetet küld vissza. ha a jelenlegi idő ezeken kívül esik akkor is visszaadja a szekrény azonosítóját.





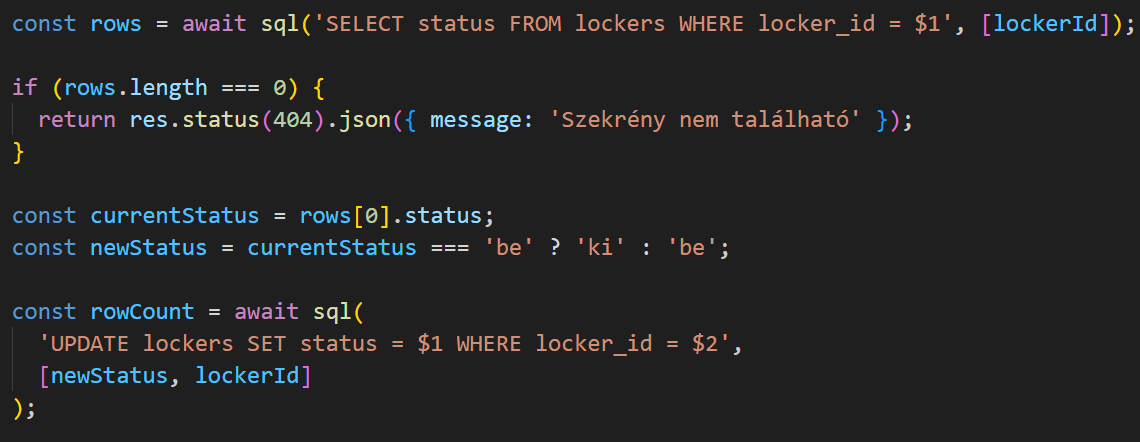
**URL:** PUT /api/locker/setLockerStatus?rfid={szekrény azonosító}

**Leírás:** Ez a végpont szerkeszti, hogy most az adott szekrényben van telefon vagy nincs.

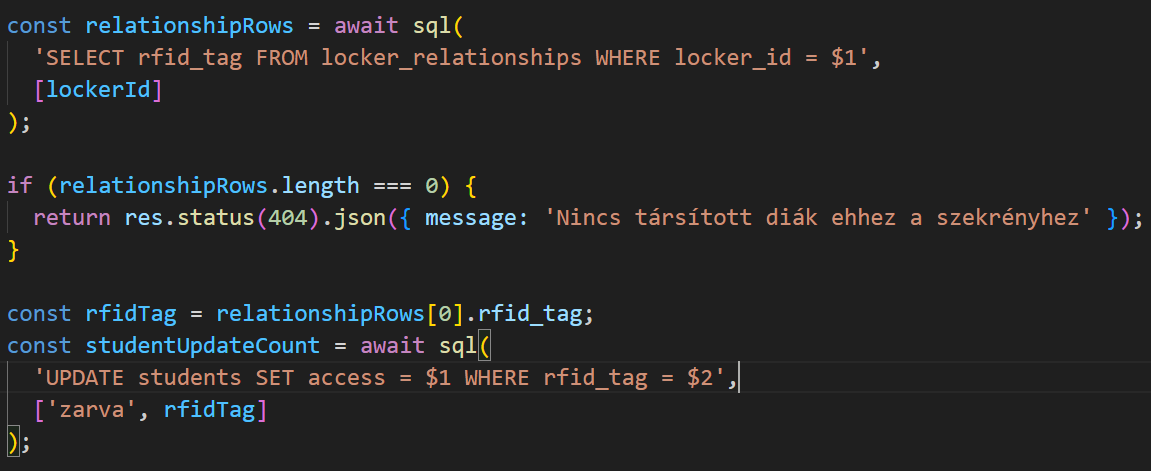
Query paraméter: id (Szekrény azonosító)

Működés:

A kapott szekrény azonosító alapján lekéri az eddigi státuszát, hogy volt e benne telefon („be”) vagy nem volt benne semmi („ki”). Majd frissíti a státuszt.



Utána a szekrényhez tartozó diáknak az „access”-ét „zarva”-ra állítja.



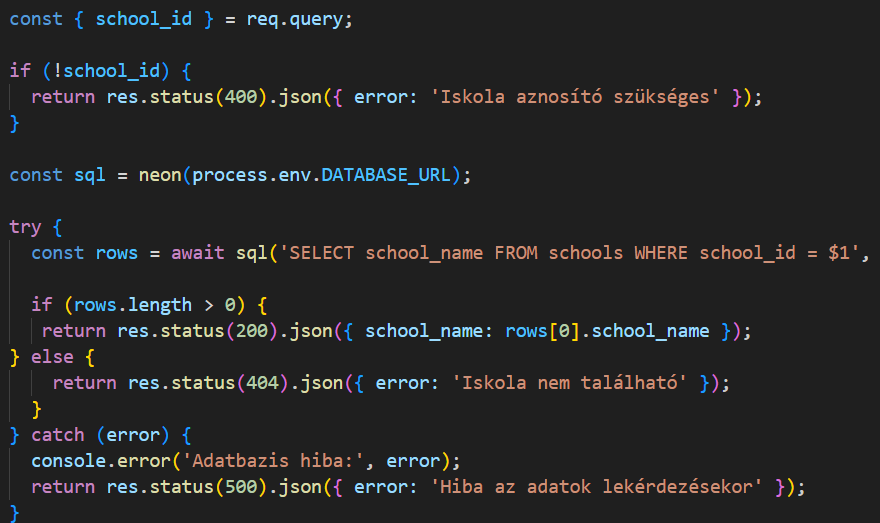
**URL:** GET /api/system/getSchool?school\_id={adott iskola azonosító}

**Leírás:** Ez a végpont lekéri az iskola teljes nevét.

Query paraméter: school \_id (Iskola azonosító)

Működés:

A kapott iskola azonosító alapján lekérdezi az iskola teljes nevét.



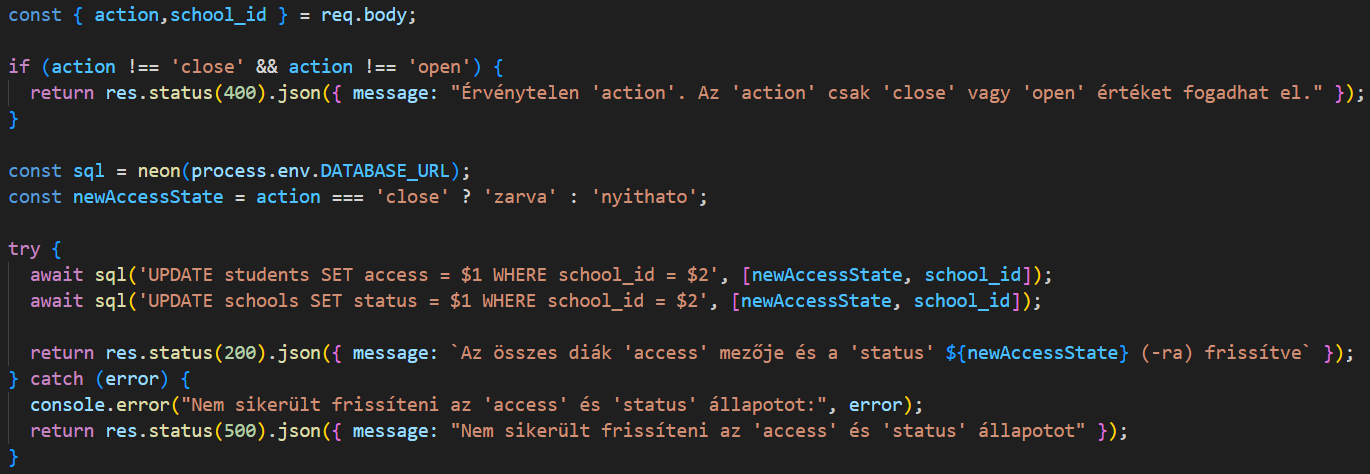
**URL:** POST /api/system/closeOpen

**Leírás:** Ez a végpont zárolja vagy feloldja az adott iskolában a szekrények nyithatóságát.

Törzs: action (close/open, attól függ, hogy zárolni szeretnéd a szekrényeket vagy feloldani), school\_id (Iskola azonosító)

Működés:

A kapott iskola azonosító alapján annak az iskolának a tárolóinak anyithatóságát frissíti, hogy a diákok iskola időben ne tudják kinyitni a szekrényeiket.



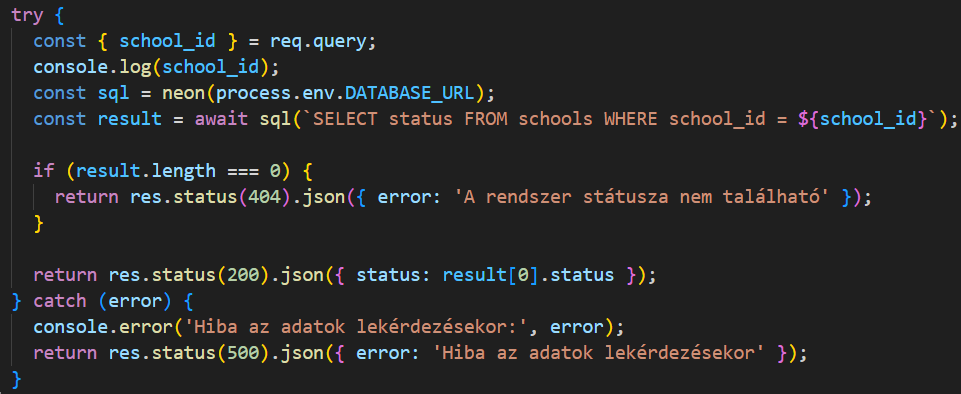
**URL:** GET /api/system/status?school\_id={adott iskola azonosító}

**Leírás:** Ez a végpont lekéri az adott iskola most éppen zárolva van vagy feloldva.

Query paraméter: school \_id (Iskola azonosító)

Működés:

A kapott iskola azonosító alapján lekérdezi az iskola státuszát („zarva” vagy „nyithato”).



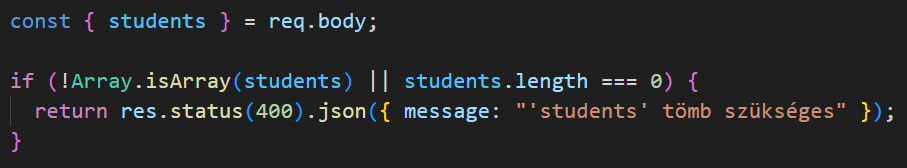
**URL:** POST /api/system/groupAccess

**Leírás:** Ez a végpont engedélyezi a szekrény nyitást egy csoportnak

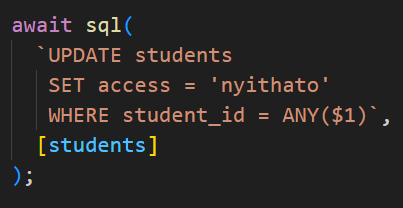
Törzs: „students” tömbb (Ebben a tömbben a diákok azonosítója van)

Működés:

Ellenőrzi a kapott tömböt, hogy szerepel e benne adat.



Majd ezeknek a diákoknak az „access” mezőjét „nyithato”-ra állítja.



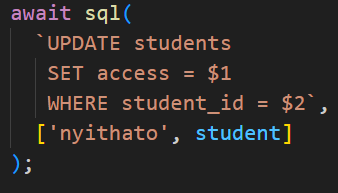
**URL:** POST /api/system/studentAccess?student\_id={Adott diák azonosító}

**Leírás:** Ez a végpont engedélyezi a szekrény nyitást egy diáknak

Query paraméter: student \_id (Diák azonosító)

Működés:

A kapott diák azonosító alapján frissíti az adott diák „access” mezőjét „nyithato”-ra.



**URL:** POST /api/students/create

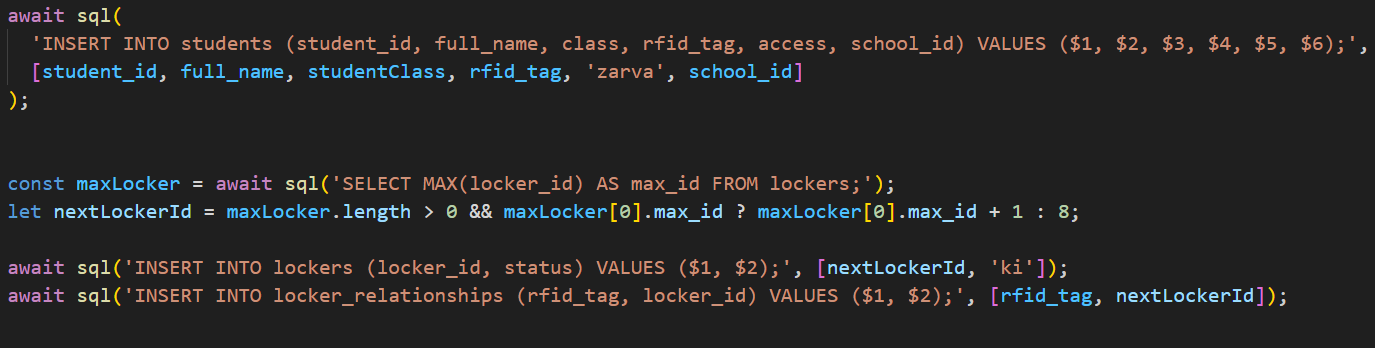
**Leírás:** Ez a végpont új egy diákot ad hozzá a rendszerhez.

Törzs:

* student\_id (Diák azonosító)
* full\_name (Teljes neve a diáknak)
* class (’,’ elválasztva a csoportok, a végpont az „studentClass”-ként hivatkozik rá)
* rfid\_tag (rfid azonosító)
* school\_id (Iskola azonosító)

Működés:

A kapott adatokat ellenőrzi. majd feltölti az adatbázisba, sikeres feltöltés után létrehozza a diák-szekrény kapcsolatot.



Utána meghívja a „setStudentGroups” végpontot amely legenerálja egy diák-csoport kapcsolatát. 

**URL:** GET /api/students/read?school\_id={Adott iskola azonosítója}

**Leírás:** Ez a végpont engedélyezi a szekrény nyitást egy diáknak

Query paraméter: school\_id (Iskola azonosító)

Működés:

Az adott iskolához tartozó összes diákot lekéri.



**URL:** POST /api/students/update

**Leírás:** Ez a végpont új egy már létező diák adatit frissíti.

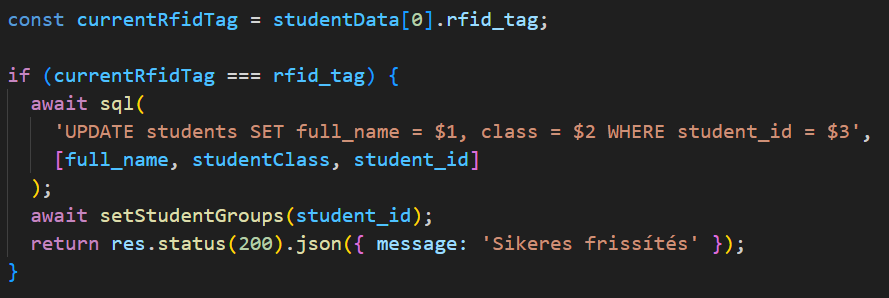
Törzs:

update (diákoknak):

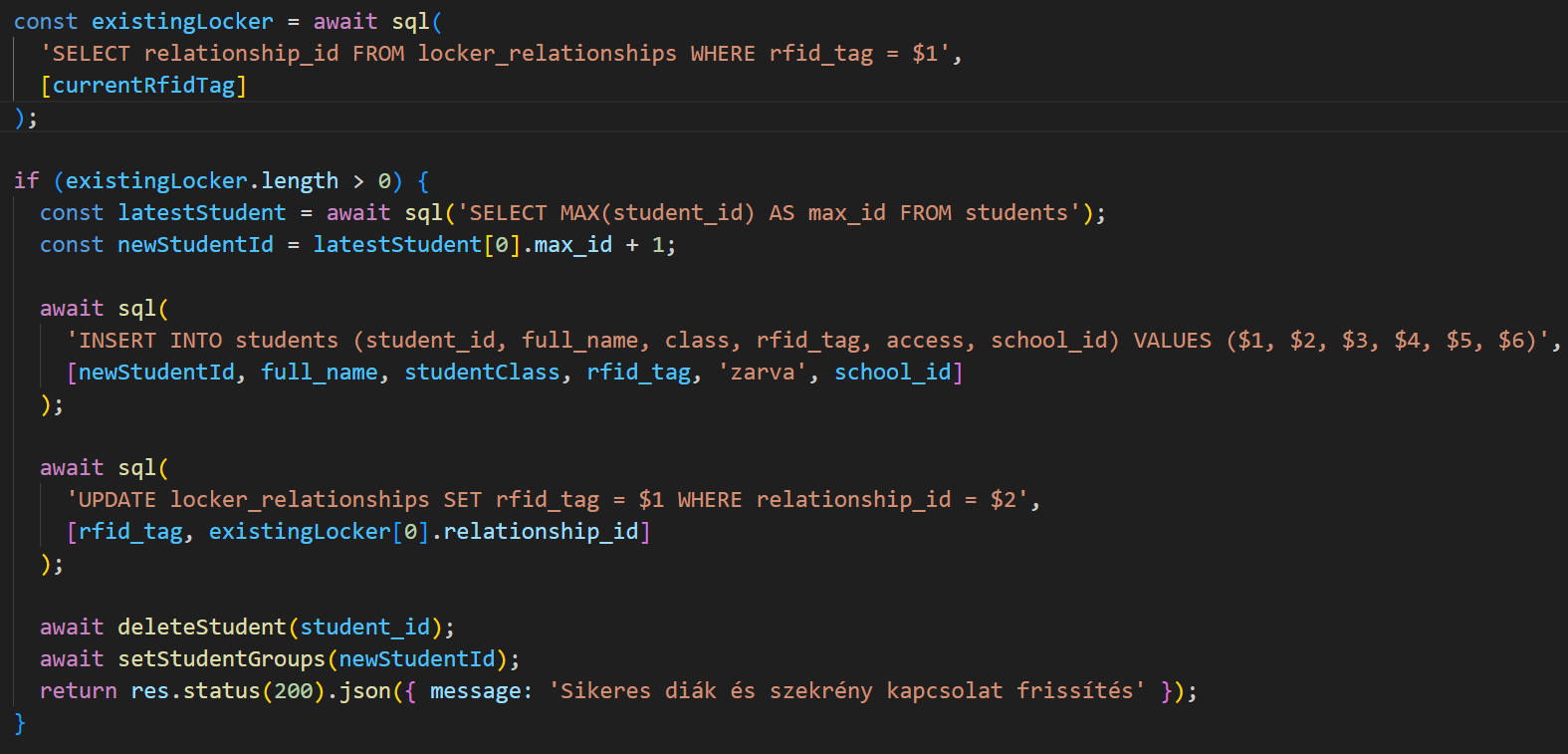
* student\_id (Diák azonosító)
* full\_name (Teljes neve a diáknak)
* class (’,’ elválasztva a csoportok, a végpont az „studentClass”-ként hivatkozik rá)
* rfid\_tag (rfid azonosító)
* school\_id (Iskola azonosító)

Működés:

Ellenőrzi, hogy a szükséges adatokat megkapta e a végpont, majd lekérdezi, a diák rfid azonosítóját, mert ha az rfid azonosító nem változott, akkor csak egyszerűen frissítjük az adatokat.



Ha viszont az rfid azonosító változott akkor elősször feltöltjök a szerkeszteni kívánt diákot új diákként majd a lekérjük a régi rfid azonosítóhoz tartozó szekrény kapcsolatot és frissíti az új rfid azonosítóra. Utána a „setStudentGroups” végponttal legenerálja a diák-csoport kapcsolatokat. És a régi diákot törli az adatbázisból.



**URL:** DELETE /api/students/delete?student\_id={Adott diák azonosítója}

**URL:** DELETE /api/config/deleteEmployee?admins\_id={Adott alkalmazott azonosítója}

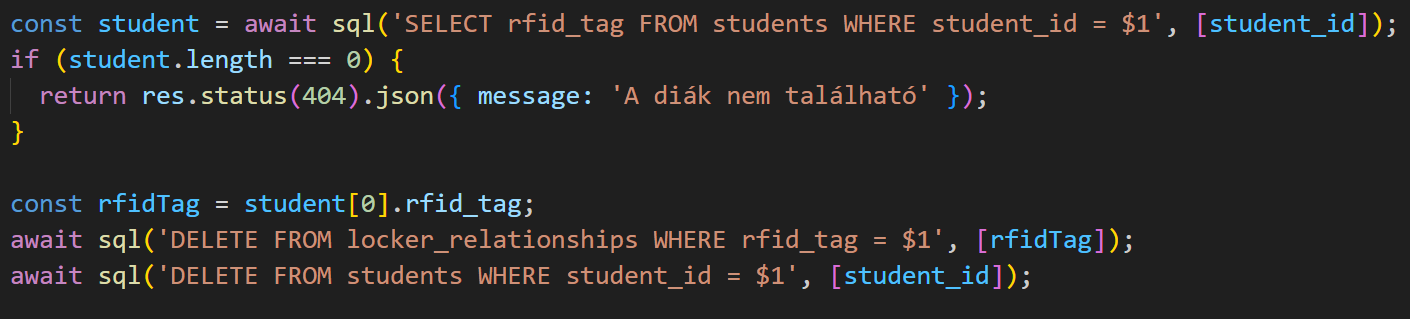
**URL:** DELETE /api/config/deletePlusBreak?year\_schedule\_id={Adott szunet azonosítója}

**Leírás:** Ezek a végpontok törlik az adott diákot/alkalmazottat/szunetet.

Query paraméter: student\_id/admins\_id/ year\_schedule\_id

Működés:

A végpont kap egy diák azonosítót, majd először törli a szekrény kapcsolatát, majd a magát a diákot. A többi végpont esetében csak egyszerű törlésről beszélhetünk. Alkalmazottak közül a rendszergazdát nem lehet kitörölni. Illetve a tanév kezdést és végét sem lehet.



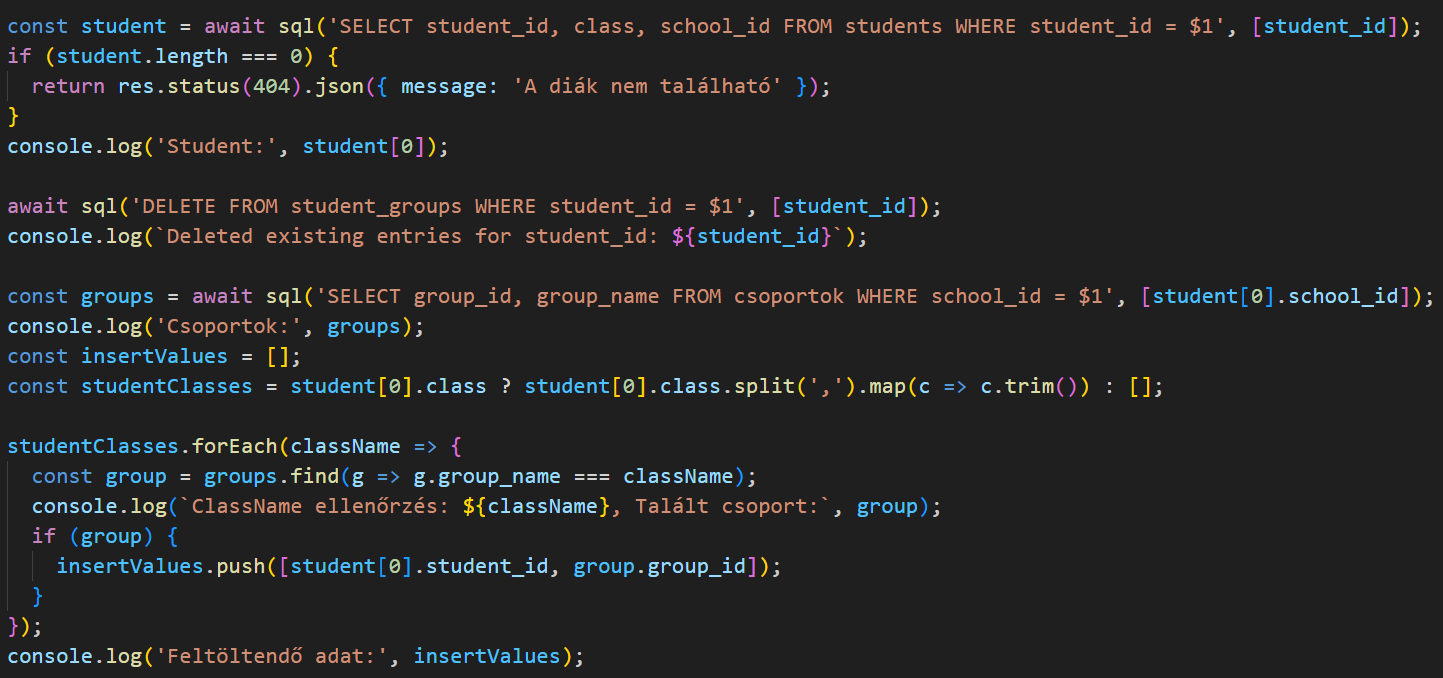
**URL:** POST /api/students/setStudentGroups?student\_id={Adott diák azonosító}

**Leírás:** Ez a végpontegy diknak létrehozza a diák-csoport kapcsolatoklat.

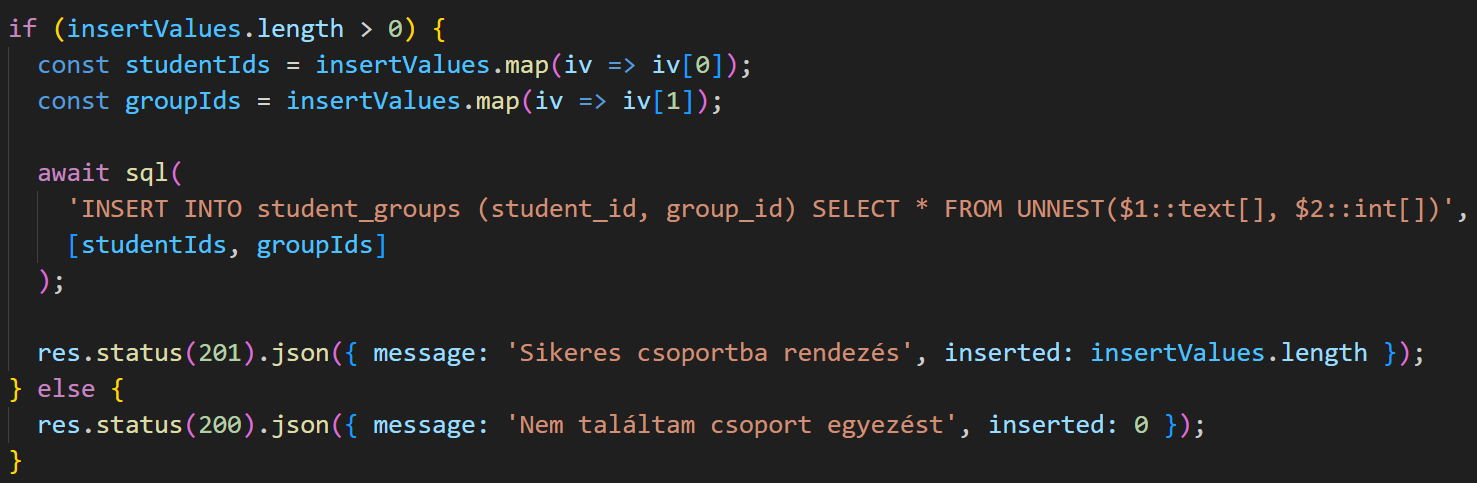
Query paraméter: student\_id (Diák azonosító)

Működés:

Először törli az eddigi csoport kapcsolatait a diáknak, majd a diák „class” mezőjének adatait sima vesszőnként (’,’) szétválasztja és a csoportok táblában szereplő csoportokkal ősszerendezi.



Majd ezeket a kapcsolatokat feltölti az adatbázisba



**URL:** POST /api/config/changePassword

**Leírás:** Ez a végpont modósítja az alapjelszót.

Működés:

A végpont ellenőrzi, hogy van e bejelentkezve felhasználó ha nincs akkor nem lehet meg változtani a jelszót. Ha van bejelentkezett felhasználó akkor megkell adni az alapjelszót majd újat amire változtatni szeretnéd. Majd a session-ben eltárolt hashelt jelszót lekéri a végpont és összehasonlítja a változtatni kívánt jelszóval.



Ha ezek megegyeznek akkor utána feltölti az adatbázisba az új jelszót. Ha nem egyeznek meg akkor hibát ad vissza.



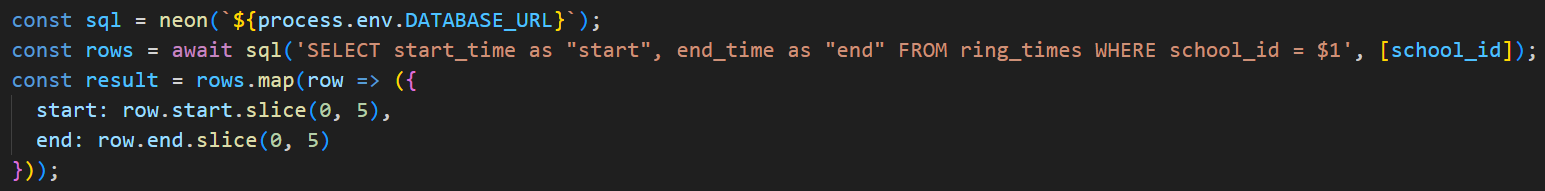
**URL:** GET /api/config/getRinging?school\_id={Adott iskola azonosító}

**URL:** GET /api/config/getREmployees?school\_id={Adott iskola azonosító}

**Leírás:** Ezek a végpont lekérdezik az adott iskolához tartozó csengetési rendet és alkalmazottakat.

Működés:

A kapott school\_id alapján lekérdezi ezeket az adatokat.



**URL:** GET /api/config/addEmployee

**URL:** GET /api/config/addPlusBreak

**Leírás:** Ezek a végpontok új alkalmazottat és új plusznapokat adnak hozzá a rendszerhez.

Törzs:

addEmployee:

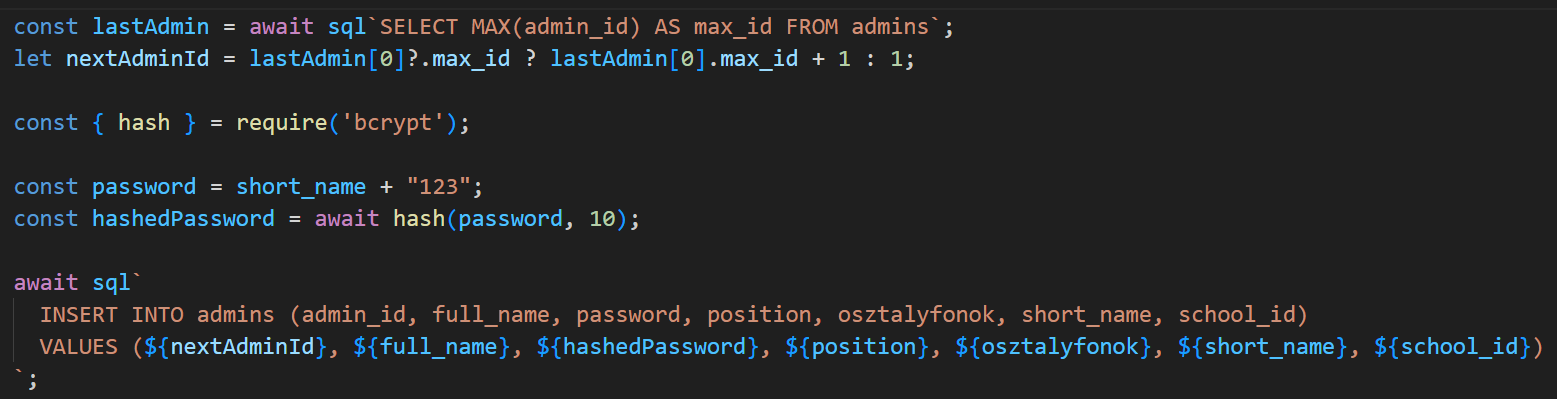
* full\_name (Alkalmazott teljes neve)
* position (Általa betöltött pozicíó)
* osztalyfonok (Melyik osztálynak az osztály főnöke, ha nincs neki akkor ez „nincs” érték)
* short\_name (Alkalmazott rövidített neve, 4 betűből áll)
* school\_id (Iskola azonosító)

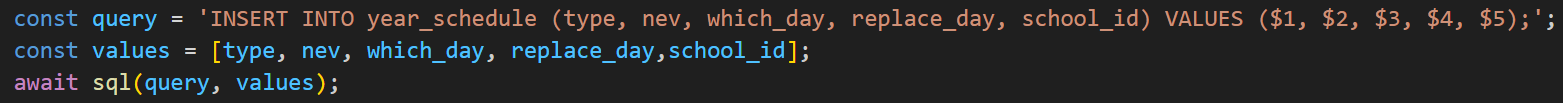
addPlusBreak:

* full\_type (Szünet típusa: szunet/plusznap/tanitasnelkul)
* nev (Neve a szünetnek, plusz napnak vagy a tanítás nélküli napnak)
* which\_day (Szünet esetében a szünet kezdete, Plusz nap esetében az a nap amikor a plusz nap lesz, tanítas nélküli nap esetében pedig az a nap amikor nem lesz tanítás)
* replace\_day (Szünet esetében a szünet utolsó napja, Plusz nap esetében itt kell megadni, hogy mely nap szerint lesz tanítás, tanítas nélküli nap esetében pedig a which\_day + 1 nap)
* school\_id (Iskola azonosító)

Működés:

A végpont ellenőrzi a kapott adatokat, majd ha minden szükséges adatot megkapott akkor feltölti az adatbázisba. Mindkét végpont esetében. Az „addEmployee” végpontban hasonló módón general alapjelszót mint az „uploadEmployees” végpontnál.





**URL:** POST /api/config/updateEmployee

**URL:** POST /api/config/setYearStartEnd

**Leírás: Ez a végpont frissíti az alkalmazott adatait vagy a tanév kezdést és befejezést.**

**Törzs:**

updateEmployee:

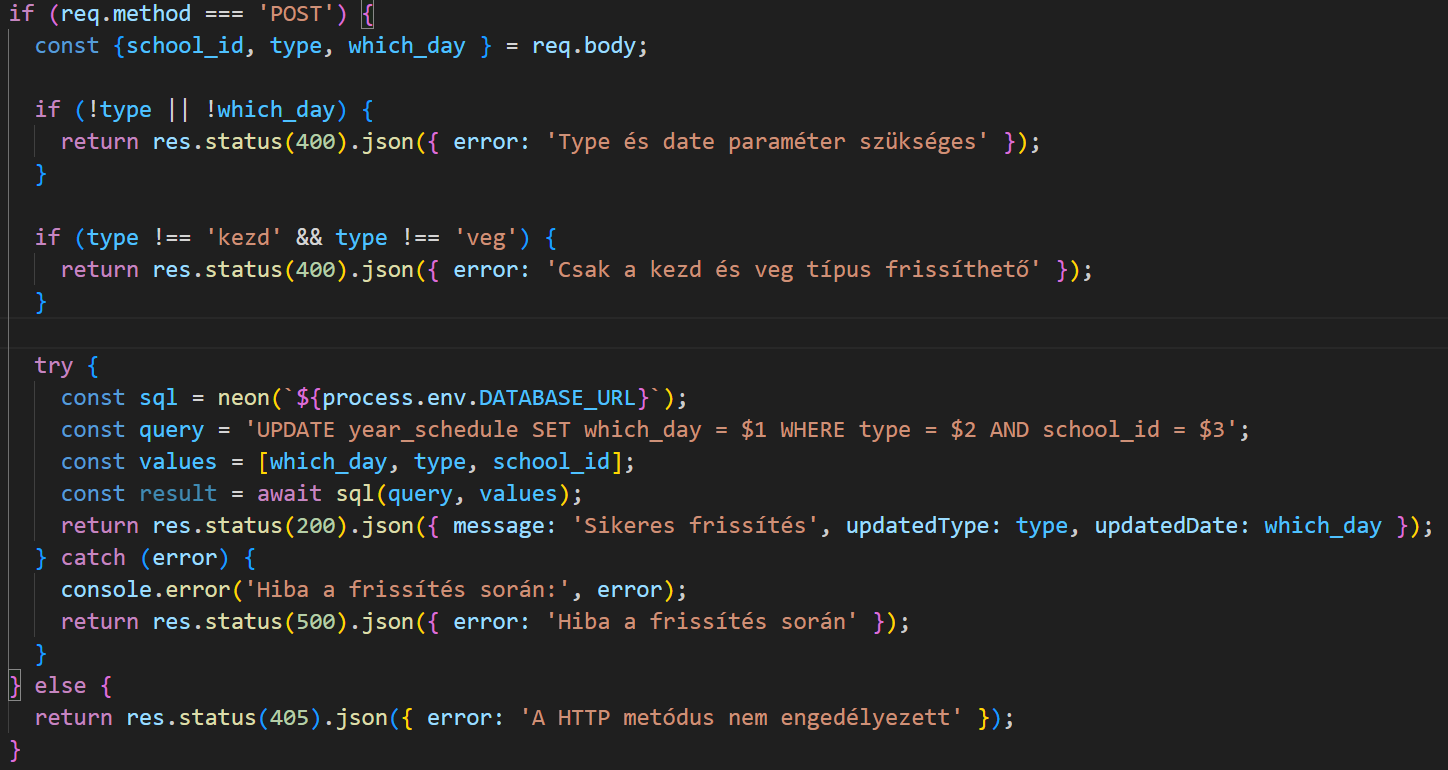
* full\_name (Alkalmazott teljes neve)
* position (Általa betöltött pozicíó)
* osztalyfonok (Melyik osztálynak az osztály főnöke, ha nincs neki akkor ez „nincs” érték)
* short\_name (Alkalmazott rövidített neve, 4 betűből áll)
* school\_id (Iskola azonosító)

setYearStartEnd:

* school\_id (Iskola azonosító)
* type (Csak „kezd” és „veg” lehet a típusa)
* which\_day (A dátum amire frissíteni szeretnéd)

Működik:

Minden iskolához egy darab tanév kezdés és egy darab tanév vége dátum van. Valamit ez egy default érték ezért csak szerkeszteni lehet (Kitörölni sem lehet). Mivel a kalendárt amire az órarendet rakja ezektől a dátumoktól-ig generálja le.



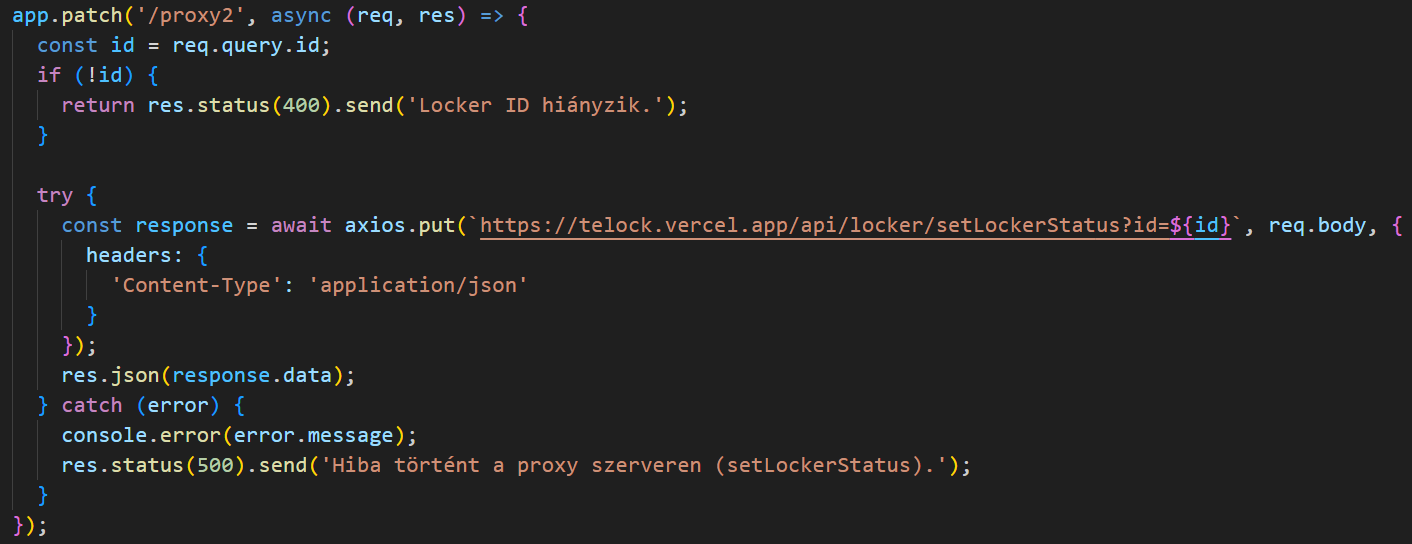
HTTPs kommunikáció megoldása….

A szekrényünk kezeléséhez arduino uno-t használunk. Ez az eszköz felel a szekrények nyitásáért és zárásáért. Viszont a vercel-en futó szoftverünk csak https kéréseket fogad el, az arduino uno pedig csak http kéréseket tud küldeni. Ezért írtunk egy áthídaló proxy szervert, amely fogadja az arduino http kéréseit és tovább küldi a vercelen futó szoftvernek https kérésként. Kód részletek az áthidaló szerverből:

Egy GET-es kérés, amely az rfid-hez tartozó szekrény azonosítót adja vissza.



Egy PATCH kérés, amely a szekrény azonosító alapján állítja, hogy most van e telefon a szekrényben vagy nincs.



Ezzel a megoldással, gyorsan és biztonságosan kommunikál a hardver és a szoftver.

Hardver:

A vizsgaremekünk hardveres része egy RFID alapú zárvezérlő rendszert valósít meg, amelyet **telefonok biztonságos tárolására** terveztünk. A rendszer egy Arduino Uno mikrokontrollerre épül, amely vezérli az **RFID olvasót**, a **reléket**, valamint két **szolenoid zárat**, amelyek a telefon tároló rekeszek fizikai nyitását és zárását végzik.

A fizikai kivitelezés részeként **három különálló tárolót** készítettünk **3D nyomtatással**:

* egyet a vezérlő elektronikának (Arduino, relék, áramellátás stb.),
* kettőt pedig a **telefonok számára** kialakított, szabványos méretű zárt rekeszeket.

A rendszer áramellátását egy külső **tápegység** biztosítja, amely elegendő energiát szolgáltat a vezérlőnek és a szolenoid zárak működtetéséhez.

A kommunikáció a rendszer és egy belső hálózaton elérhető szerver között Ethernet kapcsolaton keresztül történik. A felhasználó RFID kártyával azonosítja magát, a szerver visszajelzése alapján a megfelelő rekesz nyitható. Emellett a rendszer figyeli, hogy a rekeszek vissza lettek-e zárva, és erről státuszfrissítést küld a szerver felé.

## 🔩 Hardverösszetevők

### 🎛️ Elektronikai eszközök

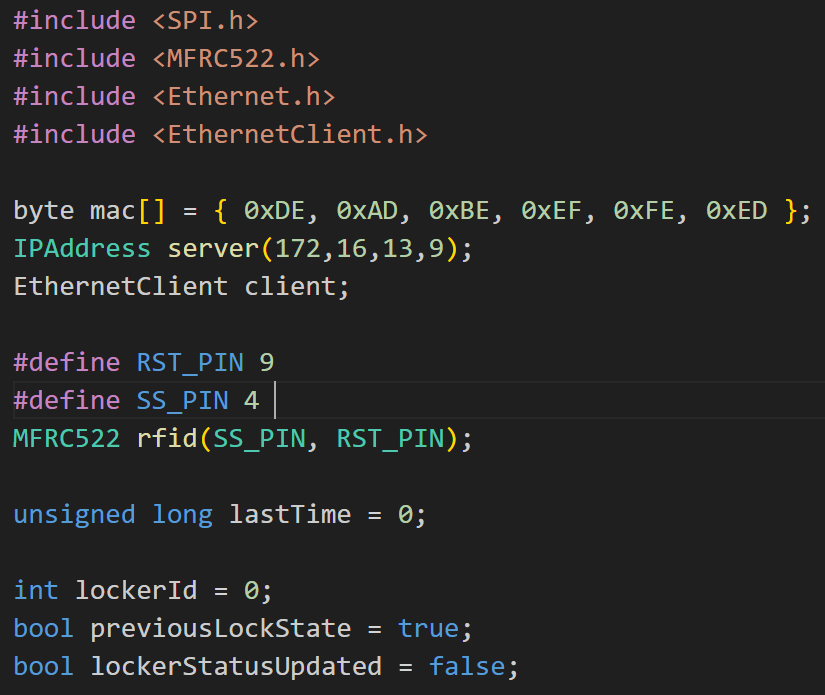
* **Arduino UNO** – 1 db
* **RFID olvasó (MFRC522)** – 1 db
* **Breadboard** – 1 db (összekötések, prototípus)
* **Relé modul** – 2 db (zárak kapcsolásához)
* **Szolenoid zár** – 2 db (fizikai zárás)
* **LCD kijelző** – 1 db (információk visszajelzéséhez – ha használva van)
* **Tápegység** – 1 db (külső áramforrás 5V/12V)

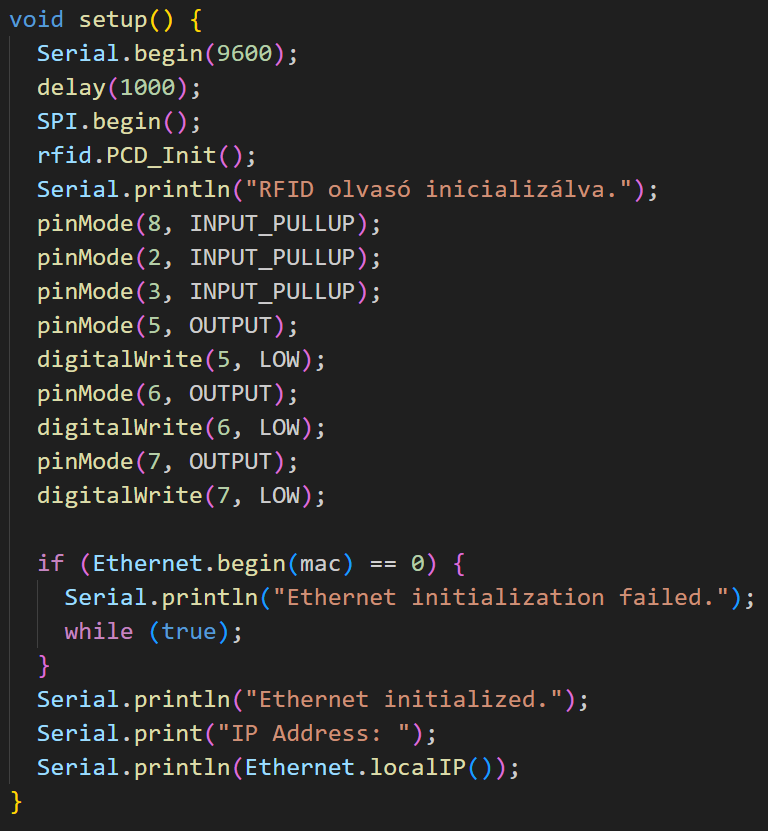
### 📦 Tároló egységek

* **Vezérlő egység doboza** – 1 db (elektronika elhelyezésére)
* **Telefon tároló rekesz** – 2 db (szabványos méretű, 3D-nyomtatott tárolók)

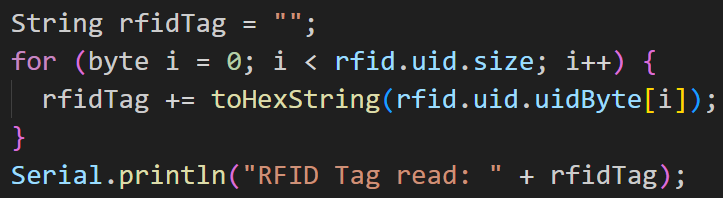
Arduino kod (arduino-programkod.ino):

Használt könyvtárak és default beállítások:

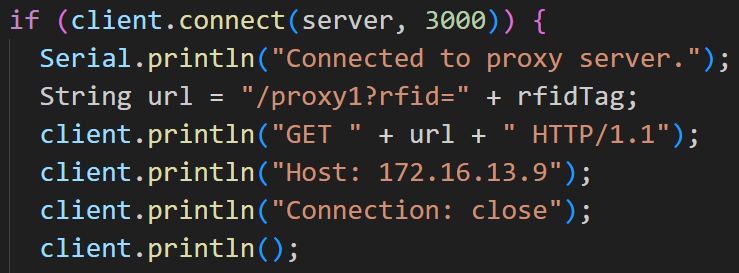




A rendszer RFID kártya leolvasásra vár. A kód elvégzi a leolvasást és az RFID címkét hexadecimális stringgé alakítja:



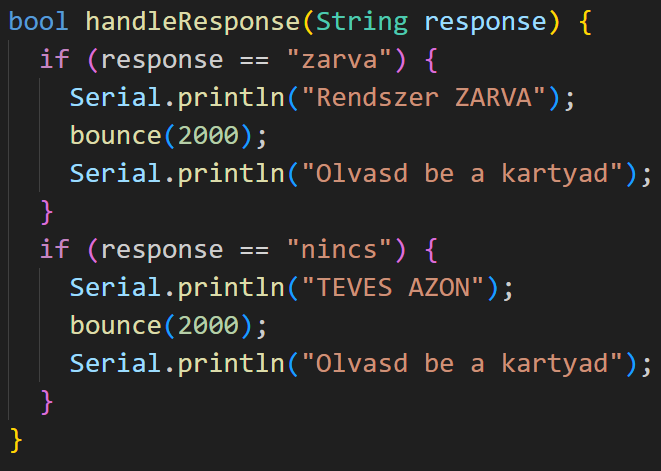
Ezután a rfid azonosítót egy HTTP GET kérésben elküldi az áthidaló szervernek:



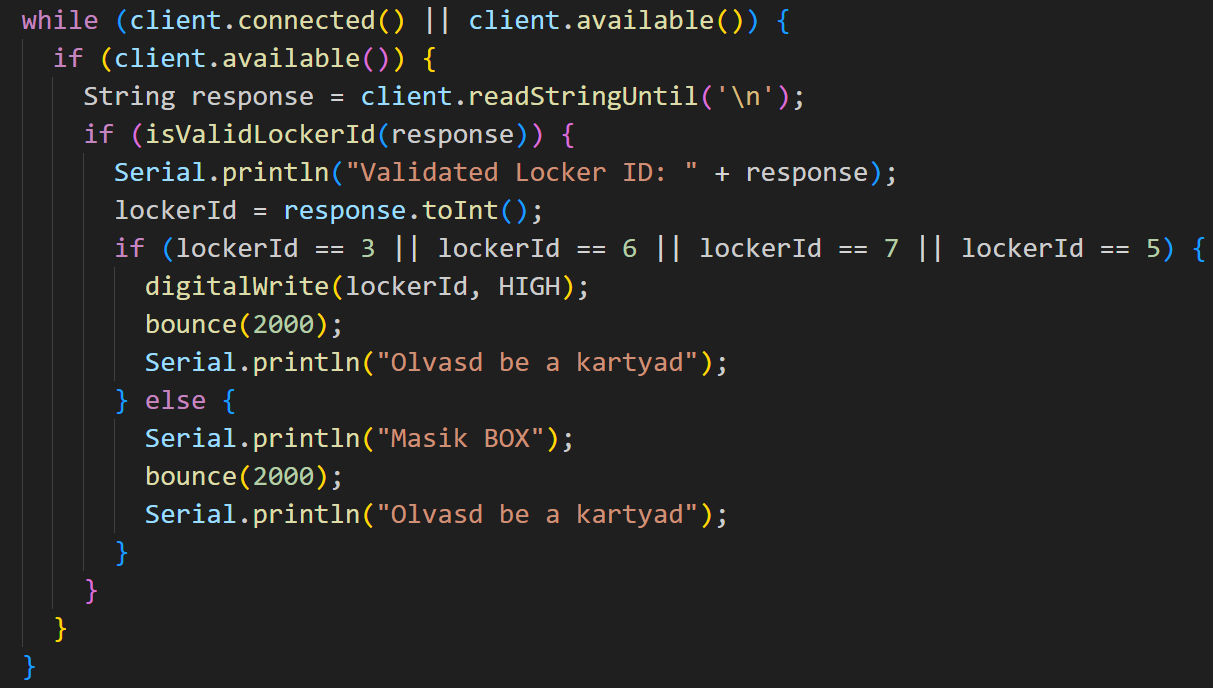
Az áthidaló szerver visszaküld egy locker\_id-t, vagy hibakódot (zarva, nincs).

A isValidLockerId() ellenőrzi, hogy számot kaptunk-e. Ha nem akkor a handleResponse() függvény kiírja a hibákat és várakozik.





Ha a lockerId 3, 5, 6 vagy 7, akkor a megfelelő relé HIGH-ra áll (Mivel a hardever csak demo jellegű ezért még csak ezken a pineken kezeli a zárakat):



Ez a HIGH jel aktiválhat egy relét, amely kinyitja a szekrényt.

## Szekrény visszazárás figyelése

A D8 lábhoz kapcsolt bemenet alapján érzékeli a zárás állapotát:



Ha a szekrény záródott (állapotváltás), akkor meghívódik a updateLockerStatus(lockerId).

A szekrény visszazárása után (szenzor alapján) a rendszer frissíti az állapotot a szerveren:

