

Kiskunfélegyházi Szent Benedek PG Két Tanítási Nyelvű Technikum és Kollégium

VIZSGAREMEK

TELOCK

Készítette:

Nagy Gábor
Szalkai-Szabó Ádám

Kiskunfélegyháza

2025

Tartalomjegyzék

1 Bevezetés	3
1.1 Témaválasztás	3
2 Fejlesztői dokumentáció	4
2.1 Hardver	4
Hardverösszetevők	4
Használt könyvtárak és alapbeállítások:	5
Szekrény visszazárás figyelése	7
2.2 Szoftver	9
2.2.1 Proxy szerver	9
2.2.2 Webes felület	10
2.2.2.1 Adatszerkezet bemutatása	10
schools tábla	11
admins tábla	11
students tábla	12
lockers tábla	13
locker_relationships tábla	13
csoportok tábla	13
student_groups tábla	14
timetables tábla	14
group_relations tábla	15
year_schedule tábla	15
ring_times tábla	16
2.2.2.2 API végpontok	17
2.3 Github és Git környezet	46
2.4 Hoszting platform	46
2.4.1 Bevezetés	46
2.4.2 Környezeti változók beállítása	48
2.4.3 Vercel és a Next.js projekt összekapcsolása	49
2.4.4 Adatbázis beállítása	51
3 Felhasználói dokumentáció	52
3.1 Bevezetés	52
3.2 Hardver és Szoftver igények	52
3.3 Webes felület elérése	53
3.4 Főoldal és tartalma	53
3.5 Belépés folyamata	54
3.6 Jelszó módosítása	55
3.7 Oldalsáv használata	56
3.8 Konfiguráció elvégzése	56
3.9 Tanév beállításaink elvégzése	57

3.9.1 Tanév váltása	57
3.9.2 Tanév első és utolsó napja	57
3.9.3 Tanév dátumai	58
3.10 Oldalak és pozíciók	59
3.10.1 Rendszergazda, igazgató és igazgatóhelyettes	59
3.10.2 Tanár	64
3.10.3 Osztályfőnök	65
3.10.4 Portás	66
3.11 Telefontároló működése	66
3.11.1 Telefon elhelyezése	66
3.11.2 Használat tanítási idő alatt	66
3.11.3 Telefon kivétele	66
3.11.4 Rendszer állapota	67

1 Bevezetés

1.1 Témaválasztás

A telock egy olyan rendszer, amelyet kifejezetten oktatási intézmények számára készítettünk, a tanulók mobiltelefon-használatának szabályozására. A megoldás egy webes vezérlőpultból és fizikai telefontároló szekrényekből áll, amely biztosítják minden tanuló számára a mobiltelefonjaik biztonságos elhelyezéséhez.

A vezérlőpult, lehetővé teszi az alkalmazottak számára a tárolók nyitásának engedélyezését. Ezáltal az iskola könnyedén kezelheti a telefonhasználatot, növelve az órai koncentrációt.

A telock rendszer bevezetése különösen időszerű a 2024. szeptember 1-jén életbe lépett kormányrendelet fényében, amely előírja, hogy a tanulóknak tanítási idő alatt le kell adniuk mobiltelefonjaikat, amelyeket az iskola által kijelölt, elzárt helyen kell tárolni. A telock megoldása segíti az iskolákat e szabályozás gyakorlati megvalósításában, biztosítva a mobiltelefonok biztonságos és szervezett kezelését.

2 Fejlesztői dokumentáció

2.1 Hardver

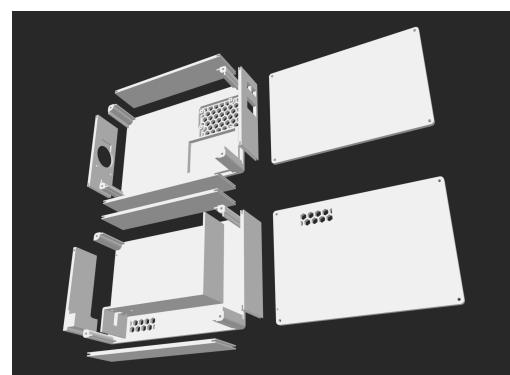
A vizsgaremekünk hardveres része egy RFID alapú zárvezérlő rendszert valósít meg, amelyet telefonok biztonságos tárolására terveztünk. A rendszer egy Arduino Uno mikrokontrollerre épül, amely vezérli az RFID olvasót, a reléket, valamint két szolenoid zárat, amelyek a telefon tároló rekeszek fizikai nyitását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

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

1. egyet a vezérlő elektronikának (Arduino, relék, áramellátás stb.),
2. 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 zákok működtetéséhez.

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



Hardverösszetevők

- Arduino Uno
- RFID olvasó (MFRC522)
- Relé modul
- Szolenoid zár
- LCD kijelző
- Tápegység
- 3D nyomtatott tárolók
- RFID kártyák és biléták

A hardverösszetevők hiánya esetén a webes felület továbbra is tesztelhető.

Használt könyvtárak és alapbeállítások:

```
#include <SPI.h>
#include <MFRC522.h>
#include <Ethernet.h>
#include <EthernetClient.h>

byte mac[] = { 0xDE, 0xAD, 0xBE, 0xEF, 0xFE, 0xED };
IPAddress server(172,16,13,9);
EthernetClient client;

#define RST_PIN 9
#define SS_PIN 4 |
MFRC522 rfid(SS_PIN, RST_PIN);

unsigned long lastTime = 0;

int lockerId = 0;
bool previousLockState = true;
bool lockerStatusUpdated = false;

void setup() {
  Serial.begin(9600);
  delay(1000);
  SPI.begin();
  rfid.PCD_Init();
  Serial.println("RFID olvasó inicializálva.");
  pinMode(8, INPUT_PULLUP);
  pinMode(2, INPUT_PULLUP);
  pinMode(3, INPUT_PULLUP);
  pinMode(5, OUTPUT);
  digitalWrite(5, LOW);
  pinMode(6, OUTPUT);
  digitalWrite(6, LOW);
  pinMode(7, OUTPUT);
  digitalWrite(7, LOW);

  if (Ethernet.begin(mac) == 0) {
    Serial.println("Ethernet initialization failed.");
    while (true);
  }
  Serial.println("Ethernet initialized.");
  Serial.print("IP Address: ");
  Serial.println(Ethernet.localIP());
}
```

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

```

String rfidTag = "";
for (byte i = 0; i < rfid.uid.size; i++) {
    rfidTag += toHexString(rfid.uid.uidByte[i]);
}
Serial.println("RFID Tag read: " + rfidTag);

```

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

```

if (client.connect(server, 3000)) {
    Serial.println("Connected to proxy server.");
    String url = "/proxy1?rfid=" + rfidTag;
    client.println("GET " + url + " HTTP/1.1");
    client.println("Host: 172.16.13.9");
    client.println("Connection: close");
    client.println();
}

```

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.

```

bool isValidLockerId(String response) {
    if (response.length() > 2)
    {
        handleResponse(response);
        return false;
    }
    for (int i = 0; i < response.length(); i++) {
        if (!isDigit(response[i]))
        {
            handleResponse(response);
            return false;
        }
    }
    return true;
}

bool handleResponse(String response) {
    if (response == "zarva") {
        Serial.println("Rendszer ZARVA");
        bounce(2000);
        Serial.println("Olvasd be a kartyad");
    }
    if (response == "nincs") {
        Serial.println("TEVES AZON");
        bounce(2000);
        Serial.println("Olvasd be a kartyad");
    }
}

```

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

```

while (client.connected() || client.available()) {
    if (client.available()) {
        String response = client.readStringUntil('\n');
        if (isValidLockerId(response)) {
            Serial.println("Validated Locker ID: " + response);
            lockerId = response.toInt();
            if (lockerId == 3 || lockerId == 6 || lockerId == 7 || lockerId == 5) {
                digitalWrite(lockerId, HIGH);
                bounce(2000);
                Serial.println("Olvasd be a kartyad");
            } else {
                Serial.println("Masik BOX");
                bounce(2000);
                Serial.println("Olvasd be a kartyad");
            }
        }
    }
}

```

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

Szekrény visszazárás figyelése

A D8 lábhoz kapcsolt bemenet alapján érzékeli a záras állapotát:

```

bool areAllLocksClosed(int lockerId) {
    bool currentLockState = (digitalRead(8) == LOW);

    if (currentLockState && !previousLockState) {
        updateLockerStatus(lockerId);
        lockerStatusUpdated = true;
    } else if (!currentLockState) {
        lockerStatusUpdated = false;
    }

    previousLockState = currentLockState;
    return currentLockState;
}

```

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

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

```
Serial.println("Connected to proxy server for status update.");
String url = "/proxy2?id=" + String(lockerId);
client.println("PUT " + url + " HTTP/1.1");
client.println("Host: 172.16.13.9");
client.println("Connection: close");
client.println();
```

2.2 Szoftver

2.2.1 Proxy szerver

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 áthidaló proxy szervert, amely fogadja az arduino http kéréseit és tovább küldi a vercelen futó szoftvernek https kérésként. Kód részletek az áthidaló szerverről:

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

```
app.get('/proxy1', async (req, res) => {
  const rfid = req.query.rfid;
  if (!rfid) {
    return res.status(400).send('RFID hiányzik.');
  }

  try {
    const response = await axios.get(`https://telock.vercel.app/api/locker/checkLocker?rfid=${rfid}`);
    res.json(response.data);
  } catch (error) {
    console.error(error.message);
    res.status(500).send('Hiba történt a proxy szerveren (getLocker).');
  }
});
```

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

```
app.patch('/proxy2', async (req, res) => {
  const id = req.query.id;
  if (!id) {
    return res.status(400).send('Locker ID hiányzik.');
  }

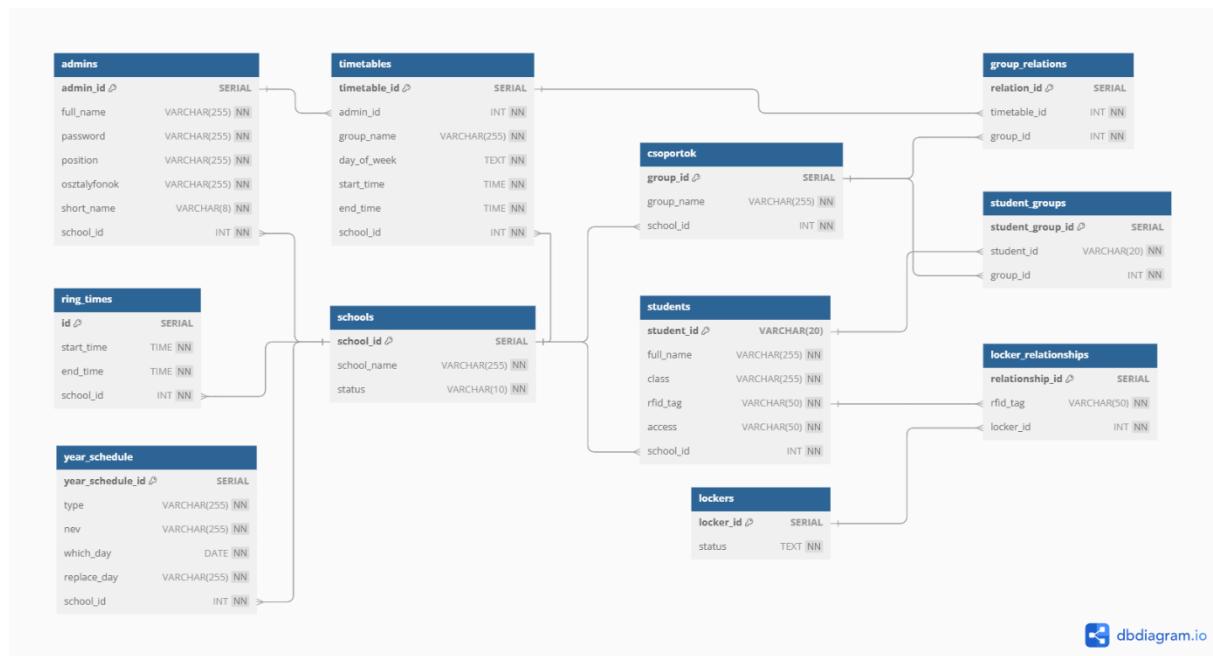
  try {
    const response = await axios.put(`https://telock.vercel.app/api/locker/setLockerStatus?id=${id}`, req.body, {
      headers: {
        'Content-Type': 'application/json'
      }
    });
    res.json(response.data);
  } catch (error) {
    console.error(error.message);
    res.status(500).send('Hiba történt a proxy szerveren (setLockerStatus).');
  }
});
```

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

2.2.2 Webes felület

2.2.2.1 Adatszerkezet bemutatása

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



Az adatbázis szerkezete egy iskolai adminisztrációs rendszer teljes körű működését támogatja. A központi elem a schools tábla, amely az intézményeket reprezentálja, ehhez kapcsolódik minden más entitás. Az admins és students táblák az iskolákhoz rendelt felhasználókat tartalmazzák, előbbiek az adminisztrátorokat, utóbbiak a tanulókat azonosítják, RFID-alapú hozzáféréssel. A lockers és locker_relationships táblák az automata szekrények kezelésére szolgálnak, a tanulók RFID-kártyájához rendelve. A csoportok (tantárgyi vagy osztály csoportok) és a student_groups kapcsolótábla biztosítja a csoporttagság nyilvántartását. Az órarendet a timetables, group_relations, valamint a ring_times táblák kezelik, utóbbi a csengetési rendet rögzíti. A year_schedule tábla speciális tanítási napokat és szüneteket tárol, lehetővé téve az év egyedi struktúrájának rögzítését. Az adatmodell szoros hivatkozási integritást biztosít, minden kulcsfontosságú kapcsolat ON DELETE CASCADE szabállyal vezérelve.

schools tábla

schools	
school_id	SERIAL
school_name	VARCHAR(255) NN
status	VARCHAR(10) NN

egy automatikusan növekvő elsődleges kulcs (SERIAL), amely egyértelműen azonosít minden iskolát. A school_name mező egyedi megkötést tartalmaz (UNIQUE), amely megakadályozza az azonos nevű iskolák többszöri rögzítését. A status mező az iskola állapotát reprezentálja, például „aktív”, „inaktív”, stb., amelyet szöveges típusban tárolunk és később akár státszkezelésre is használhatunk (például karbantartási mód vagy archiválás jelzésére).

admins tábla

admins	
admin_id	SERIAL
full_name	VARCHAR(255) NN
password	VARCHAR(255) NN
position	VARCHAR(255) NN
osztalyfonok	VARCHAR(255) NN
short_name	VARCHAR(8) NN
school_id	INT NN

A schools tábla az iskolák adatait tartalmazza, és központi szerepet tölt be a rendszer többi táblájának szervezeti szintű kapcsolódásaiban. minden rekord egyedi iskolát reprezentál. A school_id

Az admins tábla az adminisztratív felhasználók (tanárok, rendszergazdák stb.) adatait tárolja. Az admin_id egyedileg azonosítja az adminisztrátort, a full_name és position mezők a személyes és beosztási adatokat tartalmazzák. A password mező a jelszavakat tárolja – ennek védelmét ajánlott erős titkosítással (például bcrypt)

megoldani a rendszer implementációja során. Az osztalyfonok mező szövegesen rögzíti, hogy melyik osztály osztályfönökéről van szó (ha releváns), míg a short_name mező az adminisztrátor rövid nevét vagy becenevét rögzíti, például megjelenítés vagy keresés céljából. A school_id idegen kulcsként hivatkozik a schools táblára, biztosítva, hogy minden admin egy adott iskolához legyen hozzárendelve. A ON DELETE CASCADE opció biztosítja, hogy az iskola törlésével az összes hozzá tartozó admin is automatikusan törlődjön.

students tábla

students	
student_id ♂	VARCHAR(20)
full_name	VARCHAR(255) NN
class	VARCHAR(255) NN
rfid_tag	VARCHAR(50) NN
access	VARCHAR(50) NN
school_id ♂	INT NN

A students tábla a tanulók adatait tartalmazza. A student_id szöveges típusú elsődleges kulcs, amely például tanulói azonosítóként funkcionálhat, és nem feltétlenül növekvő számsorozat. A full_name a tanuló teljes neve, a class a tanuló osztálya (pl. "12.A"), míg az rfid_tag egyedi azonosítója az RFID-alapú beléptetéshez, tároláshoz, stb. A rfid_tag mezőre egyedi megkötés vonatkozik, így

nem lehet két tanulónak ugyanaz a címkeje. Az access mező különböző hozzáférési szinteket vagy jogosultsági típusokat definiálhat, például „tanuló”, „kitiltva”, „ideiglenes”. A school_id idegen kulcs a schools táblára mutat, a már említett CASCADE viselkedéssel. A tanulók adatait több kapcsolódó tábla is használja, például RFID alapú szekrényhozzárendelés során.

lockers tábla

lockers	
locker_id	SERIAL
status	TEXT NN

A lockers tábla az intelligens szekrények fizikai egységeit modellezi. A locker_id az elsődleges kulcs, minden szekrényt egyedileg azonosít. A status mező két lehetséges értéket vehet fel: 'be' (foglalt) vagy 'ki' (szabad). A CHECK megszorítás biztosítja, hogy csak ezen értékek

tárolhatók a mezőben. Ez a tábla lehetővé teszi az aktuális szekrényállapot lekérdezését, például amikor egy tanuló belép és szeretné használni a tárolót.

locker_relationships tábla

locker_relationships	
relationship_id	SERIAL
rfid_tag	VARCHAR(50) NN
locker_id	INT NN

A locker_relationships tábla az RFID címkek és a szekrények közötti kapcsolatot tárolja. minden rekord egy konkrét tanuló és szekrény összerendelését jelenti. Az rfid_tag mező idegen kulcs a students.rfid_tag mezőre, míg a locker_id a lockers tábla

megfelelő kulcsa. A ON DELETE CASCADE opció gondoskodik arról, hogy a tanuló vagy a szekrény törlése automatikusan érvénytelenítse a kapcsolatot. Ez a tábla lehetőséget ad az intelligens szekrények dinamikus kiosztására és felszabadítására.

csoportok tábla

csoportok	
group_id	SERIAL
group_name	VARCHAR(255) NN
school_id	INT NN

A csoportok (vagyis csoportok) tábla tanulói csoportokat reprezentál, például szaktárgyi vagy projektalapú csoportosításokat. A group_id az elsődleges kulcs, a group_name a csoport elnevezése, amely nem

feltétlenül egyedi. A school_id itt is meghatározza, hogy az adott csoport melyik iskolához

tartozik. A ON DELETE CASCADE biztosítja, hogy az iskola törlésével minden hozzá tartozó csoport automatikusan törlésre kerüljön.

student_groups tábla

student_groups		
student_group_id	SERIAL	
student_id	VARCHAR(20)	NN
group_id	INT	NN

és csoportok táblára. A ON DELETE CASCADE opció gondoskodik a referenciális integritás fenntartásáról törlés esetén. Ez a tábla lehetőséget nyújt rugalmas csoportos tanrend kialakítására és testreszabására.

timetables tábla

timetables		
timetable_id	SERIAL	
admin_id	INT	NN
group_name	VARCHAR(255)	NN
day_of_week	TEXT	NN
start_time	TIME	NN
end_time	TIME	NN
school_id	INT	NN

és end_time mezők pontos időintervallumot rögzítenek. A school_id itt is kötelező mező, és a

A student_groups egy sok-sok kapcsolatot modellező kapcsolótábla a tanulók és csoportok között. Egy tanuló több csoporthoz is tartozhat, és egy csoportban több tanuló is lehet. A student_id és group_id mezők idegen kulcsok a students

A timetables tábla a tanrendeket (órarendeket) tárolja. A timetable_id egyedi azonosító, az admin_id az órát tartó tanárra mutat. A group_name mező a tanóra által érintett csoportot nevezi meg, amely nem kapcsolódik közvetlenül a csoportok táblához (ezért a group_relations tábla biztosítja a strukturált kapcsolatot). A day_of_week mező korlátozott szöveges értékeket fogad el (hétfőtől péntekig), a start_time

CASCADE szabály továbbra is érvényben van. Ez a tábla központi szerepet játszik a tanulói és tanári órarendek leképezésében.

group_relations tábla

group_relations	
relation_id ↫	SERIAL
timetable_id ↫	INT NN
group_id ↫	INT NN

A group_relations tábla a tanórák (timetables) és a csoportok (csoportok) közötti kapcsolatot írja le. Ez lehetővé teszi, hogy egy órarendi bejegyzés több csoporthoz is kapcsolódjon – például összevont órák esetén. A timetable_id és group_id mezők biztosítják az egyértelmű kapcsolatot, az idegen kulcsok pedig gondoskodnak a referenciaik épségéről. Ez a tábla elengedhetetlen az órarendek rugalmas kezeléséhez.

year_schedule tábla

year_schedule	
year_schedule_id ↫	SERIAL
type	VARCHAR(255) NN
nev	VARCHAR(255) NN
which_day	DATE NN
replace_day	VARCHAR(255) NN
school_id ↫	INT NN

A year_schedule tábla az iskolai naptár eseményeit tartalmazza. A type mező az esemény típusát határozza meg (pl. „ünnep”, „rövidített nap”, „plusz nap”), a nev az esemény neve (pl. „Nemzeti ünnep”), míg a which_day konkrét dátumra vonatkozik. A replace_day mező abban az esetben releváns, ha az adott nap egy másik hétköznapot helyettesít (pl. szombat munkanap, ami hétfőként viselkedik). A school_id idegen kulcs itt is meghatározza az esemény intézményi érvényességét. Ez a tábla fontos szerepet játszik az automatikus tanrend-korrekciónak.

ring_times tábla

ring_times	
id ↗	SERIAL
start_time	TIME NN
end_time	TIME NN
school_id ↗	INT NN

szekrényidőzítés, beléptetési ablakok).

A ring_times tábla az iskolai csengetési rendet rögzíti. minden rekord egy óra kezdési és befejezési idejét tartalmazza, az adott iskolához kötve (school_id). A mezők start_time és end_time típusúak, és pontos időzítést tesznek lehetővé. Ez a tábla elengedhetetlen az órarendek és szünetek pontos kezeléséhez, valamint automatizált rendszerekhez (pl. csengetésvezérlés,

2.2.2.2 API végpontok

Státusz kódok:

200, 201, 400, 404, 405, 500

URL: POST /api/setup/ascToDatabase?school_id={az adott iskolai 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 szedik ki majd egy ringing nevű tömbbe tárolja el, utána a tanárokat és óraadókat gyűjtö össze, egy employees tömbbe. Az órákat pedig a schedule tömbbe tárolja, utána már csak a csoportokat szedik 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:

```
const ringing = extractRingingSchedule(parsedXml);
const employees = extractTeachers(parsedXml);
const schedule = extractSchedule(parsedXml);
const groups = extractGroups(parsedXml);

console.log('Kinyert órarendi adatok:', schedule);
console.log('Kinyert tanárok:', employees);
console.log('Kinyert csoportok:', groups);

await Promise.all([
  sendRingingData(ringing, school_id),
  sendEmployeesData(employees, school_id),
  sendGroupsData(groups, school_id),
]);

await waitForDatabaseToBeReady(sql, employees.length, school_id);
await sendScheduleData(schedule, school_id);

return res.status(201).json({
  message: 'XML adatok sikeresen feldolgozva és továbbítva!',
  ringing,
  employees,
});
```

Csengetési rend adatok:

```
function extractRingingSchedule(parsedXml) {
  if (!parsedXml.timetable || !parsedXml.timetable.periods || !parsedXml.timetable.periods[0].period) {
    return [];
  }

  return parsedXml.timetable.periods[0].period.map(p => ({
    start_time: p.$.starttime,
    end_time: p.$.endtime,
  }));
}
```

Tanárok adatai:

```
function extractTeachers(parsedXml) {
  if (!parsedXml.timetable || !parsedXml.timetable.teachers || !parsedXml.timetable.teachers[0].teacher) {
    return [];
  }

  const teachers = parsedXml.timetable.teachers[0].teacher;
  const classes = parsedXml.timetable.classes ? parsedXml.timetable.classes[0].class : [];
  const lessons = parsedXml.timetable.lessons ? parsedXml.timetable.lessons[0].lesson : [];
  const classTeacherMap = {};

  lessons.forEach(lesson => {
    if (lesson.$._subjectid === "DAA739606BB71EE6" && lesson.$._teacherids && lesson.$._classids) {
      classTeacherMap[lesson.$._teacherids] = lesson.$._classids;
    }
  });

  return teachers.map(t => ({
    full_name: t.$._name,
    short_name: t.$._short,
    position: "Tanár",
    osztalyfonok: classTeacherMap[t.$._id] ? classes.find(c => c.$._id === classTeacherMap[t.$._id])?.name || null : null,
  }));
}
```

Csoportok adatai

```
function extractGroups(parsedXml) {
  if (!parsedXml.timetable ||
    !parsedXml.timetable.classes ||
    !parsedXml.timetable.classes[0].class ||
    !parsedXml.timetable.groups ||
    !parsedXml.timetable.groups[0].group) {
    return [];
  }

  const classes = parsedXml.timetable.classes[0].class;
  const csoportok = parsedXml.timetable.groups[0].group;

  return [
    ...classes.map(c => ({
      name: c.$._name,
    })),
    ...csoportok
      .filter(g => g.$._name !== "Egész osztály")
      .map(g => ({
        name: g.$._name,
      }))
  ];
}
```

Órarend adatok:

```
function extractSchedule(parsedXml) {
  if (!parsedXml.timetable?.lessons?.[0]?.lesson || !parsedXml.timetable?.cards?.[0]?.card) return [];

  // 1. Időszakok feldolgozása (óra kezdési és befejezési időpontjai)
  const periods = Object.fromEntries(
    parsedXml.timetable.periods[0].period.map(p => [
      p.$.period.trim(),
      { start: p.$.starttime, end: p$.endtime }
    ])
  );

  // 2. Napok dekódolása
  const dayCodes = ["Hétfő", "Kedd", "Szerda", "Csütörtök", "Péntek"];
  const daysMap = Object.fromEntries(
    parsedXml.timetable.daysdefs[0].daysdef.map(d => {
      const daysBinary = d.$.days.padStart(5, '0');
      const activeDays = daysBinary
        .split("")
        .map((bit, index) => (bit === "1" ? dayCodes[index] : null))
        .filter(Boolean);

      return [d.$.id.trim(), activeDays];
    })
  );

  // 3. Tantárgyak és tanárok lekérése
  const subjects = Object.fromEntries(
    parsedXml.timetable.subjects[0].subject.map(s => [s.$.id.trim(), s.$.name])
  );

  const teachers = Object.fromEntries(
    parsedXml.timetable.teachers[0].teacher.map(t => [t.$.id.trim(), t.$.name])
  );

  // 4. Osztályok betöltése
  const classes = Object.fromEntries(
    parsedXml.timetable.classes[0].class.map(cls => [cls.$.id.trim(), cls.$.name])
  );

  // 5. Csoportok (`groups`) betöltése és kapcsolás osztályokhoz
  const groups = Object.fromEntries(
    parsedXml.timetable.groups[0].group.map(g => [
      g.$.id.trim(),
      g.$.entireclass === "1" ? classes[g$.classid.trim()] || "Ismeretlen osztály" : g.$.name
    ])
  );
}
```

```

// 6. Az órák (`lessons`) lekérése `lessonid` alapján
const lessons = Object.fromEntries(
  parsedXml.timetable.lessons[0].lesson.map(lesson => [lesson.$.id.trim(), lesson.$])
);

// 7. Órarend feldolgozása a `cards` szekció alapján
return parsedXml.timetable.cards[0].card.flatMap(card => {
  const lesson = lessons[card.$.lessonid.trim()];
  if (!lesson) return [];

  // Period hozzárendelése
  const period = periods[card.$.period?.trim()] || { start: "N/A", end: "N/A" };

  // Napok dekódolása a `days` mezőből
  const daysBinary = card.$.days.padStart(5, '0');
  const activeDays = daysBinary
    .split("")
    .map((bit, index) => (bit === "1" ? dayCodes[index] : null))
    .filter(Boolean);

  // Tanárok keresése
  const teacherIds = lesson.teacherids || "";
  const teacherNames = teacherIds.split(",")
    .map(id => teachers[id.trim()] || "Ismeretlen tanár")
    .join(", ");

  // Tantárgy keresése
  const subjectName = subjects[lesson.subjectid?.trim()] || "Ismeretlen tantárgy";

  // Csoport nevének megkeresése (`groups` táblából)
  const groupIds = lesson.groupids?.split(",")
    .map(id => groups[id.trim()] || id.trim())
    .join(",") || "Nincs csoport";

  return activeDays.map(day => ({
    day,
    start_time: period.start,
    end_time: period.end,
    group_name: subjectName,
    teacher: teacherNames,
    group: groupIds
  }));
});
}

```

URL: POST /api/setup/studentToDatabase?school_id={az adott iskola azonosítója}

Leírás: Ez az API végpont lehetővé teszi tanulók adatainak feltöltését egy CSV fájlból az adatbázisba. A fájl formátumának megfelelően kell tartalmaznia a tanulók adatait. A feltöltött adatok alapján minden tanulónak 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 tanulóhoz 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 parancssal feltöltöm az adatbázisba. Sikeres feltöltés után meghívom a „uploadStudentGroups” ez a végpont az összes tanulónak létrehozza a tanuló-csoport kapcsolatát, hogy egy tanuló mely csoportokban szerepel. Utána a „uploadStudLockRelations” végpont létrehozza a tanuló-szekrény kapcsolatot. Sikeres futás után 200-as kóddal tér vissza.

tanuló azonosító és rfid azonosító generálása:

```
const RESERVED_IDS = new Set(["OM11111", "OM22222", "OM33333", "OM44444"]);
let baseID = 7000;

function generateStudentID() {
    while (RESERVED_IDS.has(`OM${baseID}`)) {
        baseID++;
    }
    const id = `OM${baseID}`;
    RESERVED_IDS.add(id);
    baseID++;
    return id;
}

function generateRFID() {
    return crypto.randomBytes(4).toString('hex').toUpperCase();
}
```

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:

```
students.push([
    generateStudentID(),
    fullName,
    studentClass,
    generateRFID(),
    'zarva',
    school_id
]);
```

Az adatbázis feltöltés:

```
if (students.length > 0) {
  const values = students.flat();
  const placeholders = students
    .map((_, i) => `(${i * 6 + 1}, ${i * 6 + 2}, ${i * 6 + 3}, ${i * 6 + 4}, ${i * 6 + 5}, ${i * 6 + 6})`)
    .join(', ');
  const insertQuery = `INSERT INTO students (student_id, full_name, class, rfid_tag, access, school_id) VALUES ${placeholders}`;
  await sql(insertQuery, values);
}
```

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 ével 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.

```
const { school_id } = req.query;
console.log(school_id);
const { ringing } = req.body;
console.log(ringing);

if (!school_id || !Array.isArray(ringing) || ringing.length === 0) {
  return res.status(400).json({ error: 'A ringing tömb üres vagy hibás' });
}
```

```

const sql = neon(` ${process.env.DATABASE_URL}`);
const insertQuery = 'INSERT INTO ring_times (start_time, end_time, school_id) VALUES ($1, $2, $3)';

for (const { start_time, end_time } of ringing) {
  if (!start_time || !end_time) {
    console.warn('Hiányzó adatok:', { start_time, end_time });
    continue; // Hibás sort kihagy
  }

  try {
    await sql(insertQuery, [start_time, end_time, school_id]);
  } catch (dbError) {
    console.error(`Hiba az adatbázisba íráskor: ${dbError.message}`);
    return res.status(500).json({ error: 'Hiba az adatok feltöltésekor' });
  }
}

```

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 nevéről plusz „123”. Majd a tanárok bejelentkezés után megtudják változtatni az alapjelszót. És minden tanárhoz hozzárendeli az képott iskola azonosítót. Utána egy sql parancsal 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 paraméter hiányzik vagy érvénytelen akkor 400-as vagy 404-es hiba kóddal tér vissza.

Az adatbázis feltöltéshez szükséges struktúra és maga a feltöltés:

```
const insertValues = await Promise.all(employees.map(async (employee) => {
  const password = employee.short_name + "123";
  const hashedPassword = await hash(password, 10);

  return [
    nextAdminId++,
    employee.full_name,
    hashedPassword,
    employee.position,
    employee.osztalyfonok || 'nincs',
    employee.short_name || null,
    school_id
  ];
}));

const query = `
INSERT INTO admins
  (admin_id, full_name, password, position, osztalyfonok, short_name, school_id)
VALUES ${insertValues.map((_, i) =>
  `(${${i*7+1}}, ${${i*7+2}}, ${${i*7+3}}, ${${i*7+4}}, ${${i*7+5}}, ${${i*7+6}}, ${${i*7+7}}`)
).join(', ')}

const params = insertValues.flat();

await sql(query, params);
```

URL: POST /api/setup/uploadTimetables?school_id={az adott isokal azonosítója}

Leírás: Ez a végpont, egy tömbből feltölти 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 parancssal feltölti az órarendet a timetables táblába és az adott órához tatozó csoportokat a group_relations táblába. Sikeres futás esetén 201-es kódossal tér vissza.

Órarend adatok rendezése és formázása:

```
const admins = await sql(`SELECT admin_id, full_name FROM admins WHERE school_id = ${school_id}`);
if (!Array.isArray(admins)) {
  throw new Error('Érvénytelen válasz az adatbázisból az adminok táblából');
}
const adminMap = new Map(admins.map(admin => [admin.full_name.trim(), admin.admin_id]));
const groups = await sql(`SELECT group_id, group_name FROM csoportok WHERE school_id = ${school_id}`);
if (!Array.isArray(groups)) {
  throw new Error('Érvénytelen válasz az adatbázisból a csoportok táblából');
}
const groupMap = new Map(groups.map(group => [group.group_name.trim(), group.group_id]));
const timetableInsertValues = [];
const groupRelationsInsertValues = [];

for (const entry of schedule) {
  const teacherId = adminMap.get(entry.teacher.trim());
  if (!teacherId) {
    console.warn(`Tanár nem található: ${entry.teacher}`);
    continue;
  }

  const groupNames = entry.group.split(',').map(name => name.trim());
  const groupIds = groupNames.map(name => groupMap.get(name)).filter(id => id !== undefined);

  if (groupIds.length === 0) {
    console.warn(`Csoport nem található: ${entry.group}`);
    continue;
  }

  timetableInsertValues.push([
    teacherId,
    entry.group_name,
    dayMapping[entry.day] || 'monday',
    entry.start_time,
    entry.end_time,
    school_id
  ]);

  groupRelationsInsertValues.push({ groupIds });
}
```

URL: POST /api/setup/uploadStudentGroups?school_id={az adott isokal azonosítója}

Leírás: Ez a végpont, egy adott iskola összes tanulójához létrehozza a tanuló-csoport kapcsolatokat.

Query paraméter: school_id (SZÜKSÉGES - az iskola azonosítója)

Működés:

Lekéri azokat a tanulókat 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.

```

const students = await sql(`SELECT student_id, class FROM students WHERE class IS NOT NULL AND school_id = ${school_id}`);
console.log('Diákok:', students);
const groups = await sql(`SELECT group_id, group_name FROM csoportok WHERE school_id = ${school_id}`);
console.log('Csoportok:', groups);
const insertValues = [];

students.forEach(student => {
  console.log(`Diák ${student.student_id} feldolgozása osztállyal:`, student.class);
  const studentClasses = student.class ? student.class.split(',').map(c => c.trim()) : [];
  studentClasses.forEach(className => {
    const group = groups.find(g => g.group_name === className);
    console.log(`ClassName ellenőrzés: ${className}, Talált csoport:`, group);
    if (group) {
      insertValues.push([student.student_id, group.group_id]);
    }
  });
});

```

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

URL: POST /api/setup/uploadStudLockRelations

Leírás: Ez a végpont, egy adott iskola összes tanulójához létrehozza a tanuló-szekrény kapcsolatokat.

Működés:

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

```

const students = await sql(`SELECT rfid_tag
  FROM students
  WHERE rfid_tag IS NOT NULL
  AND rfid_tag NOT IN (SELECT rfid_tag FROM locker_relationships)
`);

const studentCount = students.length;

if (studentCount === 0) {
  return res.status(400).json({ message: 'Nem találtak RFID-címkelvel rendelkező diákokat' });
}

```

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

```

const maxLocker = await sql('SELECT MAX(locker_id) AS max_id FROM lockers');
let nextLockerId = maxLocker[0].max_id ? maxLocker[0].max_id + 1 : 8;
const lockerValues = Array.from({ length: studentCount }, (_, i) => [nextLockerId + i, 'ki']);
await sql(
  `INSERT INTO lockers (locker_id, status) SELECT * FROM UNNEST($1::int[], $2::text[])
  [lockerValues.map(lv => lv[0]), lockerValues.map(lv => lv[1])]
`);

```

Sikeress feltöltés után létrehozzuk a tanuló-szekrény kapcsolatokat, majd ezeket is feltöljük az adatbázisba.

```
const relationshipValues = students.map((student, index) => [student.rfid_tag, nextLockerId + index]);
await sql(
  'INSERT INTO locker_relationships (rfid_tag, locker_id) SELECT * FROM UNNEST($1::text[], $2::int[])
  [relationshipValues.map(rv => rv[0]), relationshipValues.map(rv => rv[1])]
);

res.status(201).json({
  message: 'Diák-szekrény kapcsolatok sikeresen feltöltve',
  lockersInserted: studentCount,
  relationshipsInserted: studentCount
});
```

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 osztály/tanár heti órarendjét, ha csoportokra vannak bontva akkor azt is. majd sorba rendezi és visszatér egy tömbbel amelyben az órák szerepelnek.

```
const results = await sql(
  `SELECT
    t.day_of_week,
    t.start_time,
    t.end_time,
    c.group_name AS class,
    t.group_name AS group_name,
    a.short_name AS teacher_name
  FROM
    timetables t
  JOIN
    group_relations gr ON t.timetable_id = gr.timetable_id
  JOIN
    csoportok c ON gr.group_id = c.group_id
  JOIN
    admins a ON t.admin_id = a.admin_id
  WHERE
    c.group_name LIKE ${className}
  ORDER BY
    CASE t.day_of_week
      WHEN 'monday' THEN 1
      WHEN 'tuesday' THEN 2
      WHEN 'wednesday' THEN 3
      WHEN 'thursday' THEN 4
      WHEN 'friday' THEN 5
      WHEN 'saturday' THEN 6
      WHEN 'sunday' THEN 7
    END,
    t.start_time,
    a.short_name ASC;``,
  [`${className}`]
);

const results = await sql(
  `SELECT
    t.day_of_week,
    t.start_time,
    t.end_time,
    STRING_AGG(c.group_name, ', ' ORDER BY c.group_name) AS class,
    t.group_name AS group_name,
    a.short_name AS teacher_name
  FROM
    timetables t
  JOIN
    group_relations gr ON t.timetable_id = gr.timetable_id
  JOIN
    csoportok c ON gr.group_id = c.group_id
  JOIN
    admins a ON t.admin_id = a.admin_id
  WHERE
    a.short_name = ${shortName}
  GROUP BY
    t.day_of_week, t.start_time, t.end_time, t.group_name, a.short_name
  ORDER BY
    CASE t.day_of_week
      WHEN 'monday' THEN 1
      WHEN 'tuesday' THEN 2
      WHEN 'wednesday' THEN 3
      WHEN 'thursday' THEN 4
      WHEN 'friday' THEN 5
      WHEN 'saturday' THEN 6
      WHEN 'sunday' THEN 7
    END,
    t.start_time;
  `,
  [`${shortName}`]
);
```

URL: GET /api/timetable/allScheduleStart?school_id={ neve}

Leírás: Ez a végpont lekéri az adott iskolához tartozó összes tanulónak 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 tanulónak, hogy mikor kezdődik az első órája és mikor fejezi be az utolsó óráját. A lekérdezésben mindenig az adott napot figyeli. A lekért adatokat tömbösítve adja vissza.

```
const { school_id } = req.query;
if (!school_id) {
    return res.status(400).json({ error: 'Hiányzó "school_id"!' });
}
console.log(school_id);

const query = `
    SELECT
        s.student_id,
        s.full_name,
        MIN(t.start_time) AS first_class_start,
        MAX(t.end_time) AS last_class_end
    FROM students s
    JOIN student_groups sg ON s.student_id = sg.student_id
    JOIN csoportok c ON sg.group_id = c.group_id
    JOIN group_relations gr ON c.group_id = gr.group_id
    JOIN timetables t ON gr.timetable_id = t.timetable_id
    WHERE t.day_of_week = LOWER(TRIM(TO_CHAR(CURRENT_DATE, 'Day')))
        AND s.school_id = $1
    GROUP BY s.student_id, s.full_name;
`;
```

URL: GET /api/timetable/scheduleStart?student_id={tanuló azonosítója}

Leírás: Ez a végpont lekéri az adott tanulónak 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 tanuló azonosító alapján lekérdezi az adott tanulónak, hogy mikor kezdődik az első órája és mikor fejezi be az utolsó óráját. A lekérdezésben minden az adott napot figyeli. A lekért adatokat json formátumban adja vissza.

```
const { student } = req.query;
if (!student) {
  return res.status(400).json({ error: 'Hiányzó "student_id"!' });
}

const query = `
SELECT
  s.student_id,
  s.full_name,
  MIN(t.start_time) AS first_class_start,
  MAX(t.end_time) AS last_class_end
FROM students s
JOIN student_groups sg ON s.student_id = sg.student_id
JOIN csoportok c ON sg.group_id = c.group_id
JOIN group_relations gr ON c.group_id = gr.group_id
JOIN timetables t ON gr.timetable_id = t.timetable_id
WHERE s.student_id = $1
  AND t.day_of_week = LOWER(TRIM(TO_CHAR(CURRENT_DATE, 'Day')))
GROUP BY s.student_id, s.full_name;
`;
```

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

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

Query paraméter: rfid (tanulóhoz tartozó rfid azonosító)

Működés:

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

```
const studentid = student[0].student_id;
const studentaccess = student[0].access;
console.log(`Aktuális id: ${studentid}`);
console.log(`Aktuális access: ${studentaccess}`);

const scheduleResponse = await fetch(`https://vizsgaremek-mocha.vercel.app/api/timetable/scheduleStart?student=${studentid}`);
if (!scheduleResponse.ok) {
    return res.status(500).json({ error: 'Nem sikerült lekérni a diákok órarendjét.' });
}
const schedule = await scheduleResponse.json();
const { first_class_start, last_class_end } = schedule[0] || {};

const currentTime = new Date().toLocaleTimeString('hu-HU', {
    timeZone: 'Europe/Budapest',
    hour12: false,
    hour: '2-digit',
    minute: '2-digit',
    second: '2-digit'
});

console.log(schedule);
console.log(`Aktuális idő: ${currentTime}`);
console.log(`Első óra kezdete: ${first_class_start}`);
console.log(`Utolsó óra vége: ${last_class_end}`);
```

Utána a végpont ellenőrzi, hogy a jelenlegi idő az a tanuló első órájának a kezdési ideje és az utolsó órájának a vége közé esik, akkor megnézi, ha a tanulónak a hozzáférhetősége „nyitható”, 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.

```

if (currentTime >= first_class_start && currentTime <= last_class_end) {
  if (studentaccess === "zarva") {
    return res.status(200).send("zarva");
  } else if (studentaccess === "nyithato") {
    const lockerResult = await getLockerByRFID(rfid, sql);
    if (lockerResult.error) {
      return res.status(lockerResult.status).json({ error: lockerResult.error });
    }
    return res.status(200).send(lockerResult.lockerId);
  }
} else {
  const lockerResult = await getLockerByRFID(rfid, sql);
  if (lockerResult.error) {
    return res.status(lockerResult.status).json({ error: lockerResult.error });
  }
  return res.status(200).send(lockerResult.lockerId);
}

```

```

async function getLockerByRFID(rfid, sql) {
  const lockerRelationship = await sql(
    'SELECT locker_id FROM locker_relationships WHERE rfid_tag = $1',
    [rfid]
  );

  if (lockerRelationship.length === 0) {
    return { error: 'Nem található szekrény_id ehhez az RFID-hez', status: 404 };
  }

  const lockerId = lockerRelationship[0].locker_id;
  const locker = await sql(
    'SELECT * FROM lockers WHERE locker_id = $1',
    [lockerId]
  );

  if (locker.length === 0) {
    return { error: 'Nem található a szekrény', status: 404 };
  }

  return { lockerId: locker[0].locker_id.toString(), status: 200 };
}

```

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 (Székreny 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.

```

const rows = await sql('SELECT status FROM lockers WHERE locker_id = $1', [lockerId]);

if (rows.length === 0) {
  return res.status(404).json({ message: 'Szekrény nem található' });
}

const currentStatus = rows[0].status;
const newStatus = currentStatus === 'be' ? 'ki' : 'be';

const rowCount = await sql(
  'UPDATE lockers SET status = $1 WHERE locker_id = $2',
  [newStatus, lockerId]
);

```

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

```

const relationshipRows = await sql(
  'SELECT rfid_tag FROM locker_relationships WHERE locker_id = $1',
  [lockerId]
);

if (relationshipRows.length === 0) {
  return res.status(404).json({ message: 'Nincs társított diákok ehhez a szekrényhez' });
}

const rfidTag = relationshipRows[0].rfid_tag;
const studentUpdateCount = await sql(
  'UPDATE students SET access = $1 WHERE rfid_tag = $2',
  ['zarva', rfidTag]
);

```

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.

```

const { school_id } = req.query;

if (!school_id) {
  return res.status(400).json({ error: 'Iskola aznosító szükséges' });
}

const sql = neon(process.env.DATABASE_URL);

try {
  const rows = await sql('SELECT school_name FROM schools WHERE school_id = $1',

    if (rows.length > 0) {
      return res.status(200).json({ school_name: rows[0].school_name });
    } else {
      return res.status(404).json({ error: 'Iskola nem található' });
    }
} catch (error) {
  console.error('Adatbazis hiba:', error);
  return res.status(500).json({ error: 'Hiba az adatok lekérdezésekor' });
}

```

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 tárolónak anyithatóságát frissíti, hogy a tanulók iskola időben ne tudják kinyitni a szekrényeket.

```

const { action,school_id } = req.body;

if (action !== 'close' && action !== 'open') {
  return res.status(400).json({ message: "Érvénytelen 'action'. Az 'action' csak 'close' vagy 'open' értéket fogadhat el." });
}

const sql = neon(process.env.DATABASE_URL);
const newAccessState = action === 'close' ? 'zarva' : 'nyithato';

try {
  await sql('UPDATE students SET access = $1 WHERE school_id = $2', [newAccessState, school_id]);
  await sql('UPDATE schools SET status = $1 WHERE school_id = $2', [newAccessState, school_id]);

  return res.status(200).json({ message: `Az összes diák 'access' mezője és a 'status' ${newAccessState} (-ra) frissítve` });
} catch (error) {
  console.error("Nem sikerült frissíteni az 'access' és 'status' állapotot:", error);
  return res.status(500).json({ message: "Nem sikerült frissíteni az 'access' és 'status' állapotot" });
}

```

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”).

```
try {
  const { school_id } = req.query;
  console.log(school_id);
  const sql = neon(process.env.DATABASE_URL);
  const result = await sql(`SELECT status FROM schools WHERE school_id = ${school_id}`);

  if (result.length === 0) {
    return res.status(404).json({ error: 'A rendszer státusza nem található' });
  }

  return res.status(200).json({ status: result[0].status });
} catch (error) {
  console.error('Hiba az adatok lekérdezésekor:', error);
  return res.status(500).json({ error: 'Hiba az adatok lekérdezésekor' });
}
```

URL: POST /api/system/groupAccess

Leírás: Ez a végpont engedélyezi a szekrény nyitást egy csoportnak

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

Működés:

Ellenőrzi a kapott tömböt, hogy szerepel-e benne adat.

```
const { students } = req.body;

if (!Array.isArray(students) || students.length === 0) {
  return res.status(400).json({ message: "'students' tömb szükséges" });
}
```

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

```
await sql(
  `UPDATE students
   SET access = 'nyithato'
  WHERE student_id = ANY($1)`,
  [students]
);
```

URL: POST /api/system/studentAccess?student_id={Adott tanuló azonosító}

Leírás: Ez a végpont engedélyezi a szekrény nyitást egy tanulónak

Query paraméter: student_id (tanuló azonosító)

Működés:

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

```
await sql(`  
  UPDATE students  
    SET access = $1  
  WHERE student_id = $2`,  
  ['nyithato', student]  
);
```

URL: POST /api/students/create

Leírás: Ez a végpont új egy tanulót ad hozzá a rendszerhez.

Törzs:

- student_id (tanuló azonosító)
- full_name (Teljes neve a tanulónak)
- class (, elválasztva a csoportok, a végpont az „studentClass”-ként hivatkozik rá)
- rfid_tag (rfid azonosító)
- school_id (Iskola azonosító)

Működés:

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

```

await sql(
  'INSERT INTO students (student_id, full_name, class, rfid_tag, access, school_id) VALUES ($1, $2, $3, $4, $5, $6)', 
  [student_id, full_name, studentClass, rfid_tag, 'zarva', school_id]
);

const maxLocker = await sql('SELECT MAX(locker_id) AS max_id FROM lockers;');
let nextLockerId = maxLocker.length > 0 && maxLocker[0].max_id ? maxLocker[0].max_id + 1 : 8;

await sql('INSERT INTO lockers (locker_id, status) VALUES ($1, $2)', [nextLockerId, 'ki']);
await sql('INSERT INTO locker_relationships (rfid_tag, locker_id) VALUES ($1, $2)', [rfid_tag, nextLockerId]);

```

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

```

async function setStudentGroups(student_id) {
  const url = `https://vizsgaremek-mocha.vercel.app/api/students/setStudentGroups?student_id=${student_id}`;

  try {
    const response = await fetch(url, {
      method: 'POST',
      headers: {
        'Content-Type': 'application/json',
      },
    });

    if (!response.ok) {
      throw new Error(`HTTP error! status: ${response.status}`);
    }

    const data = await response.json();
    console.log('Válasz:', data);
    return data;
  } catch (error) {
    console.error('Hiba az "api/students/setStudentGroups" végpont meghívása során', error);
    throw error;
  }
}

```

URL: GET /api/students/read?school_id={Adott iskola azonosítója}

Leírás: Ez a végpont engedélyezi a szekrény nyitást egy tanulónak

Query paraméter: school_id (Iskola azonosító)

Működés:

Az adott iskolához tartozó összes tanulót lekéri.

```

const query = `

SELECT
    students.*,
    lockers.status AS status
FROM
    students
LEFT JOIN
    locker_relationships ON students.rfid_tag = locker_relationships.rfid_tag
LEFT JOIN
    lockers ON locker_relationships.locker_id = lockers.locker_id
WHERE
    students.school_id = $1
`;

const students = await sql(query, [school_id]);

```

URL: POST /api/students/update

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

Törzs:

update (tanulóknak):

- student_id (tanuló azonosító)
- full_name (Teljes neve a tanulónak)
- 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 tanuló rfid azonosítóját, mert ha az rfid azonosító nem változott, akkor csak egyszerűen frissítjük az adatokat.

```

const currentRfidTag = studentData[0].rfid_tag;

if (currentRfidTag === rfid_tag) {
    await sql(
        'UPDATE students SET full_name = $1, class = $2 WHERE student_id = $3',
        [full_name, studentClass, student_id]
    );
    await setStudentGroups(student_id);
    return res.status(200).json({ message: 'Sikeres frissítés' });
}

```

Ha viszont az rfid azonosító változott akkor először feltöljök a szerkeszteni kívánt tanulót új tanulóként majd a lekérjük a régi rfid azonosítóhoz tartozó szekrény kapcsolatot és frissíti az új rfid azonosítóra. Utána a „setStudentGroups” végponttal legenerálja a tanuló-csoport kapcsolatokat. És a régi tanulót törli az adatbázisból.

```
const existingLocker = await sql(
  'SELECT relationship_id FROM locker_relationships WHERE rfid_tag = $1',
  [currentRfidTag]
);

if (existingLocker.length > 0) {
  const latestStudent = await sql('SELECT MAX(student_id) AS max_id FROM students');
  const newStudentId = latestStudent[0].max_id + 1;

  await sql(
    'INSERT INTO students (student_id, full_name, class, rfid_tag, access, school_id) VALUES ($1, $2, $3, $4, $5, $6)',
    [newStudentId, full_name, studentClass, rfid_tag, 'zarva', school_id]
  );

  await sql(
    'UPDATE locker_relationships SET rfid_tag = $1 WHERE relationship_id = $2',
    [rfid_tag, existingLocker[0].relationship_id]
  );
}

await deleteStudent(student_id);
await setStudentGroups(newStudentId);
return res.status(200).json({ message: 'Sikeres diák és szekrény kapcsolat frissítés' });
}
```

```
async function deleteStudent(student_id) {
  const deleteResponse = await fetch(`https://telock.vercel.app/api/students/delete`, {
    method: 'DELETE',
    headers: { 'Content-Type': 'application/json' },
    body: JSON.stringify({ student_id })
  });
  if (!deleteResponse.ok) {
    throw new Error('Sikertelen törlés');
  }
}
```

```

async function setStudentGroups(student_id) {
  const url = `https://telock.vercel.app/api/students/setStudentGroups?student_id=${student_id}`;

  try {
    const response = await fetch(url, {
      method: 'POST',
      headers: {
        'Content-Type': 'application/json',
      },
    });

    if (!response.ok) {
      throw new Error(`HTTP error! status: ${response.status}`);
    }

    const data = await response.json();
    console.log('Válasz:', data);
    return data;
  } catch (error) {
    console.error('Hiba az "api/students/setStudentGroups" végpont meghívása során:', error);
    throw error;
  }
}

```

URL: DELETE /api/students/delete?student_id={Adott tanuló 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 tanulóot/alkalmazottat/szunetet.

Query paraméter: student_id/admins_id/ year_schedule_id

Működés:

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

```

const student = await sql('SELECT rfid_tag FROM students WHERE student_id = $1', [student_id]);
if (student.length === 0) {
  return res.status(404).json({ message: 'A diákok nem található' });
}

const rfidTag = student[0].rfid_tag;
await sql('DELETE FROM locker_relationships WHERE rfid_tag = $1', [rfidTag]);
await sql('DELETE FROM students WHERE student_id = $1', [student_id]);

```

URL: POST /api/students/setStudentGroups?student_id={Adott tanuló azonosító}

Leírás: Ez a végpontegy diknak létrehozza a tanuló-csoport kapcsolatokat.

Query paraméter: student_id (tanuló azonosító)

Működés:

Először törli az eddigى csoport kapcsolatait a tanulónak, majd a tanuló „class” mezőjének adatait sima vesszőnként (’,’) szétválasztja és a csoportok táblában szereplő csoportokkal összerendezi.

```
const student = await sql('SELECT student_id, class, school_id FROM students WHERE student_id = $1', [student_id]);
if (student.length === 0) {
  return res.status(404).json({ message: 'A diákok nem találhatók' });
}
console.log('Student:', student[0]);

await sql('DELETE FROM student_groups WHERE student_id = $1', [student_id]);
console.log(`Deleted existing entries for student_id: ${student_id}`);

const groups = await sql('SELECT group_id, group_name FROM csoportok WHERE school_id = $1', [student[0].school_id]);
console.log('Csoportok:', groups);
const insertValues = [];
const studentClasses = student[0].class ? student[0].class.split(',') .map(c => c.trim()) : [];

studentClasses.forEach(className => {
  const group = groups.find(g => g.group_name === className);
  console.log(`ClassName ellenőrzés: ${className}, Talált csoport:`, group);
  if (group) {
    insertValues.push([student[0].student_id, group.group_id]);
  }
});
console.log('Feltöltendő adat:', insertValues);
```

Majd ezeket a kapcsolatokat feltölti az adatbázisba.

```
if (insertValues.length > 0) {
  const studentIds = insertValues.map(iv => iv[0]);
  const groupIds = insertValues.map(iv => iv[1]);

  await sql(
    'INSERT INTO student_groups (student_id, group_id) SELECT * FROM UNNEST($1::text[], $2::int[])',
    [studentIds, groupIds]
  );

  res.status(201).json({ message: 'Sikeres csoportba rendezés', inserted: insertValues.length });
} else {
  res.status(200).json({ message: 'Nem találtam csoport egyezést', inserted: 0 });
}
```

URL: POST /api/config/changePassword

Leírás: Ez a végpont modósítja az alapjelszót.

Működés:

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

```

const sql = neon(` ${process.env.DATABASE_URL}`);
const session = await getServerSession(req, res, authOptions);
if (!session) {
    console.log("Hiba: Nincs bejelentkezve!");
    return res.status(401).json({ message: "Nincs bejelentkezve." });
}

console.log(" Bejelentkezett felhasználó:", session.user.short_name);
const { oldPassword, newPassword } = req.body;
if (!oldPassword || !newPassword) {
    console.log(" Hiányzó adatok:", { oldPassword, newPassword });
    return res.status(400).json({ message: "Minden mezőt ki kell tölteni." });
}

const isMatch = await compare(oldPassword, session.user.password);
if (!isMatch) {
    console.log("Hibás régi jelszó!");
    return res.status(400).json({ message: "Hibás régi jelszó." });
}

```

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

```

const newHashedPassword = await hash(newPassword, 10);

try {
    await sql(
        "UPDATE admins SET password = $1 WHERE short_name = $2",
        [newHashedPassword, session.user.short_name]
    );
    console.log(" Jelszó sikeresen módosítva!");
    return res.status(200).json({ message: "Jelszó sikeresen módosítva!" });
} catch (error) {
    console.log("Hiba az adatbázis frissítésekor:", error.message);
    return res.status(500).json({ message: "Hiba a jelszó módosítása során.", error: error.message });
}

```

URL: GET /api/config/getRinging?school_id={Adott iskola azonosító}

URL: GET /api/config/getREmployees?school_id={Adott iskola azonosító}

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

Működés:

A kapott school_id alapján lekérdezi ezeket az adatokat.

```

const sql = neon(` ${process.env.DATABASE_URL}`);
const rows = await sql(`SELECT start_time as "start", end_time as "end" FROM ring_times WHERE school_id = $1`, [school_id]);
const result = rows.map(row => ({
  start: row.start.slice(0, 5),
  end: row.end.slice(0, 5)
}));

```

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 pozíció)
- 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, tanítás 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, tanítás nélküli nap esetében pedig a which_day + 1 nap)
- school_id (Iskola azonosító)

Működés:

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

```

const lastAdmin = await sql`SELECT MAX(admin_id) AS max_id FROM admins`;
let nextAdminId = lastAdmin[0].max_id ? lastAdmin[0].max_id + 1 : 1;

const { hash } = require('bcrypt');

const password = short_name + "123";
const hashedPassword = await hash(password, 10);

await sql`  

  INSERT INTO admins (admin_id, full_name, password, position, osztalyfonok, short_name, school_id)  

  VALUES (${nextAdminId}, ${full_name}, ${hashedPassword}, ${position}, ${osztalyfonok}, ${short_name}, ${school_id})  

`;

```

```

const query = 'INSERT INTO year_schedule (type, nev, which_day, replace_day, school_id) VALUES ($1, $2, $3, $4, $5)';
const values = [type, nev, which_day, replace_day, school_id];
await sql(query, values);

```

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 pozíció)
- 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)

Működés:

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 dátumuktól-ig generálja le.

```
if (req.method === 'POST') {
  const {school_id, type, which_day} = req.body;

  if (!type || !which_day) {
    return res.status(400).json({ error: 'Type és date paraméter szükséges' });
  }

  if (type !== 'kezd' && type !== 'veg') {
    return res.status(400).json({ error: 'Csak a kezd és veg típus frissíthető' });
  }

  try {
    const sql = neon(`#${process.env.DATABASE_URL}`);
    const query = 'UPDATE year_schedule SET which_day = $1 WHERE type = $2 AND school_id = $3';
    const values = [which_day, type, school_id];
    const result = await sql(query, values);
    return res.status(200).json({ message: 'Sikerességtelen frissítés', updatedType: type, updatedDate: which_day });
  } catch (error) {
    console.error('Hiba a frissítés során:', error);
    return res.status(500).json({ error: 'Hiba a frissítés során' });
  }
} else {
  return res.status(405).json({ error: 'A HTTP metódus nem engedélyezett' });
}
```

2.3 Github és Git környezet

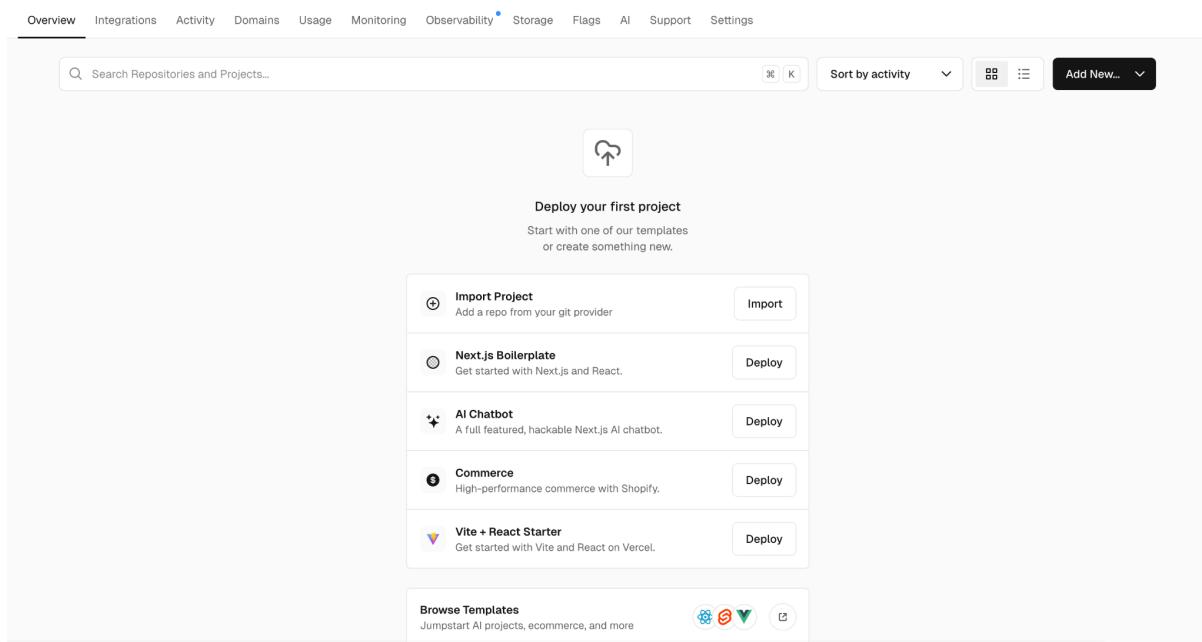
A kiírt igénynek eleget téve és közös munkánk zökkenőmentes végzése miatt, kialakítottunk egy GitHub környezetet a fejlesztésünknek. A környezetet az iskolánk által biztosított email címmel hoztuk létre, majd egy közös repository-t hoztunk létre „vizsgaremek” néven. Így a fejlesztés során végrehajtott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

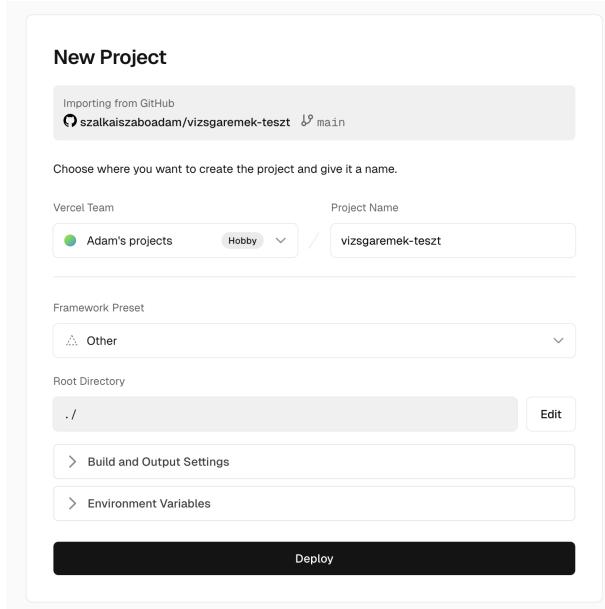
Egyéni környezeten, a <https://git-scm.com/> helyről telepített eszköz lehetőséget adott a saját gépünk és a felhős tárhely közötti Git szabályainak megfelelő kapcsolatot. Az elkészült munka, tartalmazza az adatbázist, a programkódot és a dokumentációt, a következő GitHub repositoryban érhető el: <https://github.com/nagygabor123/vizsgaremek>.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

2.4 Hoszting platform

2.4.1 Bevezetés

A webes felület hosztingolására a Vercel nevű platformot választottuk, mivel ingyenes tárhelyet és domain biztosít, így teljes mértékben megfelel a rendszerünk igényeinek. Belépve a megfelelő GitHub-fiókkal, az ”Import Project” szekcióban az ”Import” gombra kattintva kiválasztható a korábban létrehozott GitHub repót, amely tartalmazza a Next.js projekt mappáit és fájljait (1. ábra).

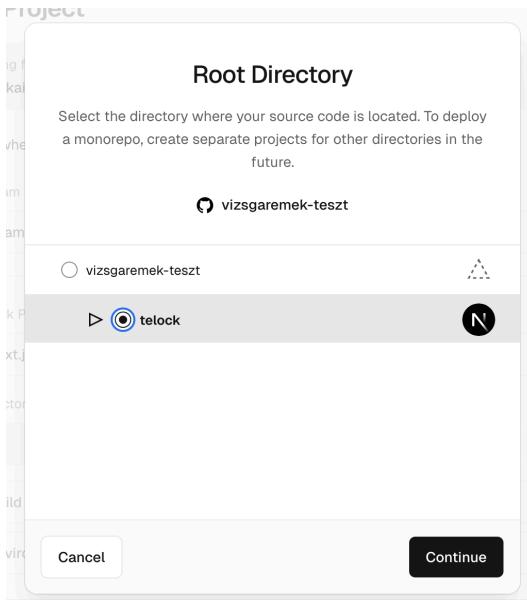




A következő lépéseket a "vizsgaremek-teszt" nevű minta projekten keresztül szemléltetjük a dokumentációban (1. ábra). Első lépésként ki kell választani a "Framework Preset" legördülő menüből azt a keretrendszert, ebben az esetben a Next.js-t (2. ábra).

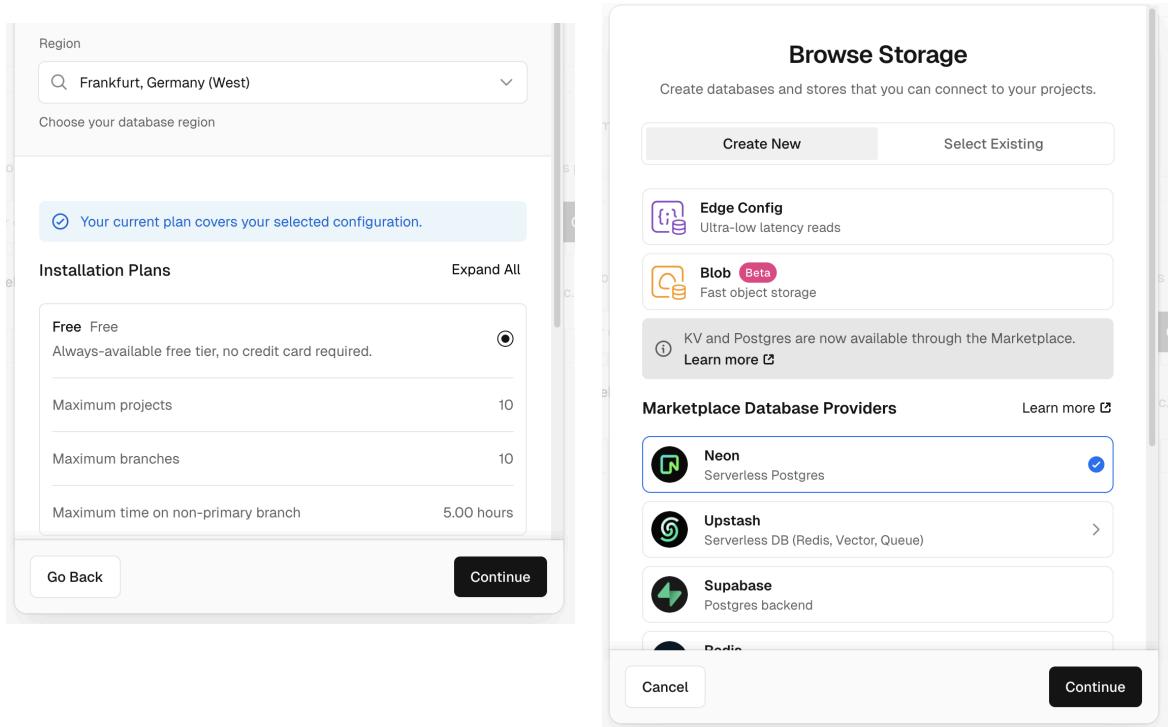


Végül meg kell adni a Next.js projekt elérési útját, a "Root Directory" gombra kattintva (3. ábra).



A következő lépésben létrehozható vagy kiválasztható az adatbázis. A Neon nevű adatbázis mellett döntöttünk, mivel felhőalapú tárolást biztosít, PostgreSQL aláú (4. ábra).

Új adatbázis létrehozásánál szükséges kiválasztani egy régiót, valamint a nekünk megfelelő díjcsomagot. A mi szükséleteinkhez az ingyenes csomag elegendő (5. ábra).



2.4.2 Környezeti változók beállítása

Az adatbázis létrehozása után a fejlécben a "Settings" oldalra átlépve meg kell adni a NextAuth.js autentikációhoz szükséges környezeti változókat az oldalsávban található "Environment Variables" szekcióban (6. ábra). Itt a "Key" és a "Value" mezők kitöltésével tudunk új változókat hozzáadni (7. és 8. ábra).

The screenshot shows the "Project Settings" page with the "Environment Variables" section highlighted. A new environment variable is being created with the key "CLIENT_KEY" and the value "SECRET". The "Sensitive" checkbox is checked. Other options like "All Environments" and "Select a custom Preview branch" are also visible.

- Key: NEXTAUTH_SECRET
- Value: h0h+3ZtI7yA4Qax0bkYE1MrW3JViF7XeqQs1XB1De1o=

A változó értékét az OpenSSL eszközzel egy 32 bájtos (256 bites) véletlenszerű bájtsorozattal kell generáltatni (Base64 formátumban).

Key	Value
NEXTAUTH_SECRET	h0h+3ZtI7yA4Qax0bkYE1MrW3JViF7XeqQs1XB1De1o=

[Add Another](#)

[Import .env](#) or paste the .env contents above [Save](#)

- Key: NEXTAUTH_URL
- Value: <https://vizsgaremek-teszt-pink.vercel.app>

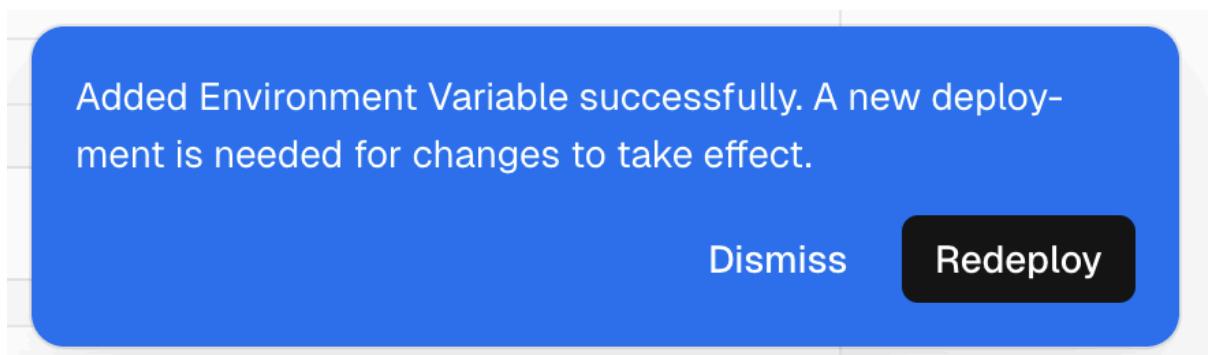
A változó értékeként a Vercel projekt URL-címét kell megadni.

Key	Value
NEXTAUTH_URL	https://vizsgaremek-teszt-pink.vercel.app/

[Add Another](#)

[Import .env](#) or paste the .env contents above [Save](#)

A két új változó hozzáadása után újra kell telepíteni a projektet a ”Redeploy” gombbal.



2.4.3 Vercel és a Next.js projekt összekapcsolása

Visszalépve a Vercel projekt vezérlőpultjára, a fejlécben a ”Storage” oldalon található az adatbázis elérési útja és a Vercel projekttel való összekapcsolás menete. Kód szerkesztő

program megnyitása után, a terminálba kell beírni az alábbi parancsokat. Először be kell jelentkezni a Vercel-be a megfelelő GitHub fiókkal: vercel login (6. ábra).

```
● szalkaiadaam@MacBookAir telock % vercel login
Vercel CLI 41.4.1
? Log in to Vercel Continue with GitHub
> Success! GitHub authentication complete for szalkai-szabo.adam@diak.szbi-pg.hu
Congratulations! You are now logged in. In order to deploy something, run `vercel`.
💡 Connect your Git Repositories to deploy every branch push automatically (https://vercel.link/git).
○ szalkaiadaam@MacBookAir telock %
```

Ezután a már meglévő Vercel projektet kell összekapcsolni a Next.js projekttel: vercel link (7. ábra).

```
● szalkaiadaam@MacBookAir telock % vercel link
Vercel CLI 41.4.1
? Set up “~/Documents/GitHub/vizsgaremek-teszt/telock”? yes
? Which scope should contain your project? Adam's projects
? Link to existing project? yes
? What's the name of your existing project? vizsgaremek-teszt
✓ Linked to adams-projects-ee9c7a47/vizsgaremek-teszt (created .vercel)
```

A vercel env pull .env.development.local parancssal lehet lekérni a környezeti változókat (8. ábra).

```
● szalkaiadaam@MacBookAir telock % vercel env pull .env.development.local
Vercel CLI 41.4.1
> Downloading `development` Environment Variables for adams-projects-ee9c7a47/vizsgaremek-teszt
✓ Created .env.development.local file [231ms]
○ szalkaiadaam@MacBookAir telock %
```

Végül telepíteni kell a Neon adatbázis könyvtárát: npm install @neondatabase/serverless.

2.4.4 Adatbázis beállítása

Az adatbázis a <https://console.neon.tech/> URL-címen érhető el. A megfelelő projekt, a "vizsgaremek-teszt" kiválasztása után (9. ábra), az oldalsávban található "SQL Editor" oldalon lehet feltölteni a táblákat (10. ábra).

The screenshot shows the Neon Tech console interface. At the top, there are buttons for "New Project" and "Import database". Below that is a section titled "Account Usage" with five metrics: Storage (0.07 / 0.5 GB), Compute (0.12 / 191.9 h), Branch Compute (0 / 5 h), Data Transfer (0 / 5 GB), and Projects (2 / 10). A note says "Metrics may be delayed up to one hour. Learn more here." Below this is a table of projects:

Name	Region	Created at	Storage	Postgres version	Integrations	⋮
vizsgaremek-teszt	AWS Europe Central 1 (Frankfurt)	Apr 6, 2025 9:10 am	34.71 MB	17	Add	⋮

A táblákat létrehozó kód a GitHub repóban található.

The screenshot shows the SQL Editor in the Neon Tech console. On the left is a sidebar with project and branch navigation. The main area has tabs for "Untitled" and "Saved", with "Untitled" selected. The code editor contains a series of SQL commands, likely for creating tables and inserting data. The status bar at the bottom says "Ready to connect" and has a "Run" button.

Tables

The screenshot shows the "Tables" page in the Neon Tech console. It lists various tables grouped by schema: neondb (neondb, public, admins), csoportok, group_relations, locker_relationships, lockers, ring_times, schools, student_groups, students, timetables, and year_schedule. The "admins" table is currently selected.

A sikeres feltöltést követően a "Tables" oldalon az összes tábla megjelenik. Most, hogy a telepítési folyamat minden lépése befejeződött, a rendszer készen áll a tesztelésre.

3 Felhasználói dokumentáció

3.1 Bevezetés

Ez a dokumentáció a telock, iskolai telefontároló rendszer felhasználói számára készült, és részletes útmutatást nyújt a rendszer funkcióinak megfelelő használatához.

A telock egy webes vezérlőpult, amely 3D nyomtatott szekrények és Solenoid zárak, valamint az elektronikai vezérlést végző Arduino segítségével tárolja el egy-egy tárolóban a tanuló mobiltelefonját a tanítási idő alatt.

A webes felület konfigurációja egyszerű, a rendszer végigvezet a minden fontos lépésen. A tanév fontos dátumai, úgymint a tanév kezdése és befejezése, tanítási szünetek, szombati tanítási napok és tanítás nélküli munkanapok mind beállíthatóak. A különböző pozíciókkal rendelkező alkalmazottak engedélyezhetik a teljes osztály vagy csoport tanulóinak a szekrények nyitását tanítási idő allatt, valamint megtekinthető a összes osztálynak és tanárnak az órarendje. Új tanulók és alkalmazottak is hozzáadhatók a rendszerhez, illetve szerkeszthetők és törölhetők is.

3.2 Hardver és Szoftver igények

A webes felület eléréséhez szükséges az alábbiak közül legalább egy eszköz amelynek stabil internetkapcsolattal rendelkezik:

- Számítógép (asztali számítógép vagy laptop)
- Mobiltelefon
- Tablet

A webes felület eléréséhez szükséges egy webböngésző használata, amely lehetővé teszi a rendszer teljes funkcionalitását. Mi az alábbiakat ajánljunk:

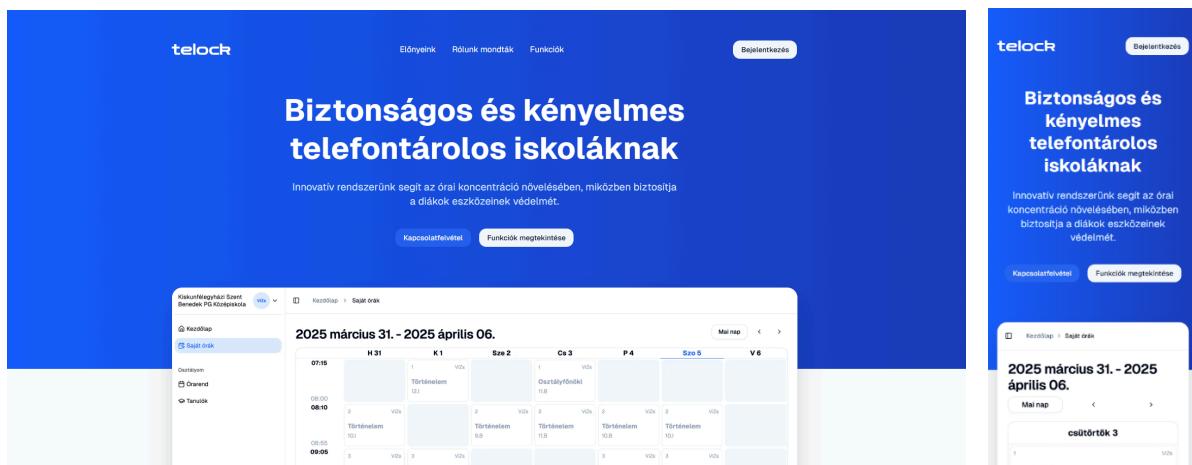
- Google Chrome (A fejlesztés során ezt a webböngészőt használtuk)
- Safari
- Firefox
- Opera

3.3 Webes felület elérése

A kiválasztott webböngésző keresőmezőjébe az alábbi URL címet kell beírni:

<https://telock.vercel.app/>

3.4 Főoldal és tartalma

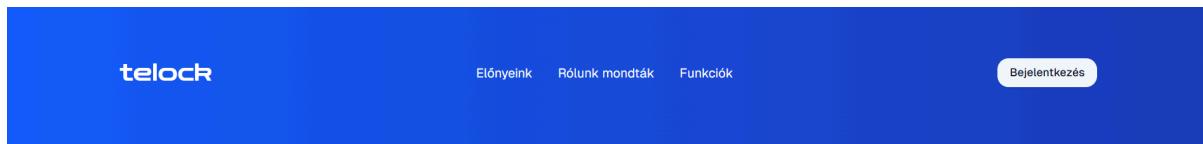


A fenti képen a telock főoldala látható, ezen az oldalon olvashatják el

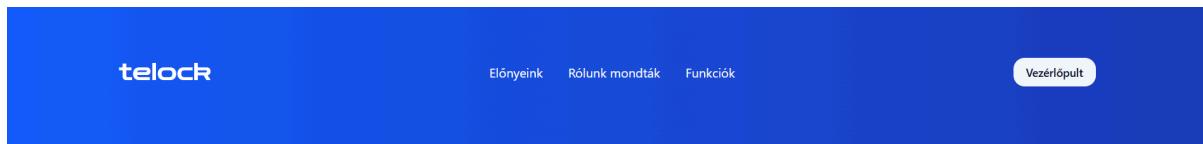
az “Előnyeink”, a “Rólunk mondták” és a “Funkciók” szekciókat, melyek a fejlécben található gombok segítségével érhetők el a legkönnyebben.

A jobb felső sarokban található a Bejelentkezés gomb, amely dinamikusan változik, attól függően, hogy be vagyunk-e jelentkezve vagy éppen nem.

Bejelentkezés előtt



Bejelentkezés után



3.5 Belépés folyamata

A rendszer legelső indulásakor, amíg nem végzik el a konfigurációt, csak egy rendszergazda felhasználója lesz az adott iskolának.

A rendszergazda belépési adatai így állnak össze:

A felhasználónév minden “Ad” kezdetű és utána jön az iskola két betűs rövidítése a “Pg” (Szent Benedek PG Technikum).

A jelszó a felhasználónévből és “123” számsorból áll (fontos, hogy az első belépést megelőzően minden felhasználónak egy ideiglenes jelszót generál a rendszer, amelyet bejelentkezést követően meg lehet változtatni).

- Felhasználónév: AdPg
- Ideiglenes jelszó: AdPg123

A belépési adatakat sikeresen beírva a rendszer visszajelzést ad és pár másodpercen belül átirányítja a vezérlőpultra, ez látható a első képen.

Viszont ha valamilyen okból rossz jelszót vagy felhasználónevet adnak meg, a második képen látható hibaüzenettel jelez a rendszer.

Bejelentkezés

Üdvözöljük! Kérjük, adja meg bejelentkezási adatait a folytatáshoz.

Felhasználónév	<input type="text" value="AdPg"/>
Jelszó	<input type="password" value="....."/>

Bejelentkezés

(✓) Sikeres bejelentkezés
 Nem sokára átirányítunk a vezérlőpultra.

Bejelentkezés

Üdvözöljük! Kérjük, adja meg bejelentkezási adatait a folytatáshoz.

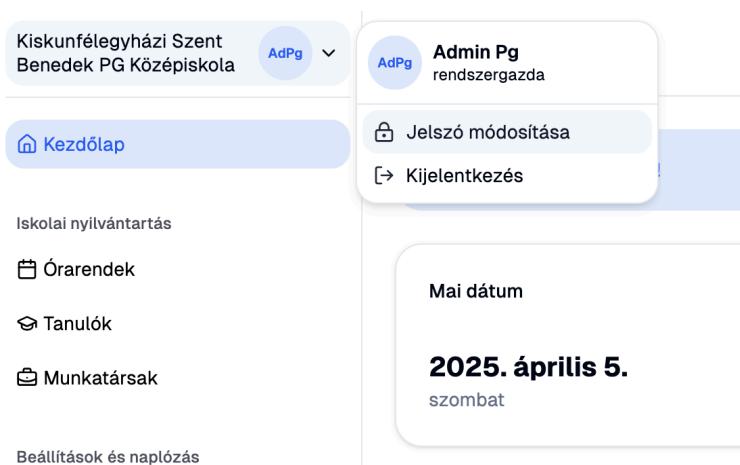
Felhasználónév	<input type="text" value="AdPg"/>
Jelszó	<input type="password" value="....."/>

Bejelentkezés

(!) Sikeretlen bejelentkezés
 Hibás felhasználónév vagy jelszól!

3.6 Jelszó módosítása

Sikeres bejelentkezést követően a rendszer átirányítja a felhasználót a vezérlőpultra. A bal felső sarokban olvasható az iskolának a neve és, hogy ki van bejelentkezve, a lefelé mutató nyílra kattintva megnyíló menüben, olvasható a felhasználó teljes neve és pozíciója valamint lehetőség van jelszót változtatni, vagy kijelentkezni.



Kiskunfélegyházi Szent Benedek PG Középiskola

AdPg ▾

- [Kezdőlap](#)
- Iskolai nyilvántartás
- Órarendek
- Tanulók
- Munkatársak
- Beállítások és naplázás

Admin Pg
rendszerigazda

Jelszó módosítása

Kijelentkezés

Mai dátum
2025. április 5.
szombat

Jelszó módosítása

Kérjük, adja meg jelenlegi jelszavát, majd állítson be egy újat.

Régi jelszó	<input type="password"/>
Új jelszó	<input type="password"/>

(✓) Sikeres jelszováltoztatás
 Jelszó sikeresen módosítva!

Feldolgozás...

Jelszó változtatáshoz, szükség lesz az ideiglenes jelszó és az új tetszőleges jelszó megadására, sikeres jelszómódosítást követően ismételten be kell jelentkeznie a felhasználónak.

3.7 Oldalsáv használata

A oldalsáv segítségével lehet navigálni a különböző oldalak között, a rendszergazdának az alábbi oldalak érhetők el:

Kezdőlap

Iskolai nyilvántartás

Órarendek

Tanulók

Munkatársak

Beállítások és naplázás

Tanév beállításai

- Kezdőlap
- Órarendek
- Tanulók
- Munkatársak
- Tanév beállításai

Az oldalak elérhetősége pozíciók szerint változik, a dokumentációban később részletesen ismertetésre kerülnek ezek. Az “Órarendek” oldalon, a rendszer jelzi, ha még nem végezték el a konfigurációt, ez látható az alábbi képen.



A rendszer nincs beállítva. Kérjük, végezze el a szükséges konfigurációt!

Konfigurálás most

3.8 Konfiguráció elvégzése

A “Konfigurálás most” gombra kattintva nyílik a konfigurációs felület, ezen a felületen szükséges feltölteni az alábbi fájlokat:

- ASC órarend (.xml formátumban)
- Tanulók listája (.csv formátumban)

The screenshot shows a configuration wizard titled "Beállítási folyamat". It displays the first step: "1. ASC órarend feltöltése" (Upload ASC timetables). A dashed blue box highlights the file upload area, which contains a circular arrow icon and the text "Válassza ki a feltölteni kívánt XML-fájlt" (Select the XML file to be uploaded). At the bottom right of the wizard is a "Mentés & Tovább" (Save & Continue) button.

Ha valamilyen okból megszakad a konfiguráció, a rendszer elmenti az adott állapotot, így nem kell újból kezdeni.

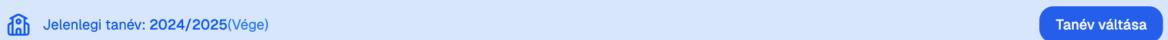
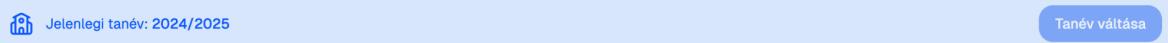
Ha sikeres volt a konfiguráció akkor az adatok sikeresen feltöltődtek az adatbázisba, innentől kezdve már használható a rendszer.

3.9 Tanév beállításaink elvégzése

A további beállításokat az oldalsávon lévő “Tanév beállításai” gombra kattintva érünk el. Az oldalt három szekcióra bontható:

3.9.1 Tanév váltása

Jelenlegi tanév kijelzése, illetve egy “Tanév váltása” gomb található itt, ami csak az adott tanév vége után érhető el, feladata, hogy minden adatot töröl majd ismételten a konfiguráció elvégzése szükséges.



3.9.2 Tanév első és utolsó napja

A tanév első és utolsó napjának beállítást lehet itt elvégezni.

Tanítási év első napja



Tanítási év utolsó napja



3.9.3 Tanév dátumai

Itt lehet új tanítási nélküli munkanapokat, szombati tanítási napok és tanítási szüneteket hozzáadni a rendszerhez.

Tanítás nélküli munkanapok

Itt láthatóak azok a napok, amikor nincs tanítás.

Új nap hozzáadás

Dátum	Művelet
2025. 02. 02.	

Szombati tanítási napok

Itt láthatóak a szombati tanítási napok.

Új nap hozzáadás

Dátum	Órarendi nap	Művelet
2025. 03. 22.	Hétfő	

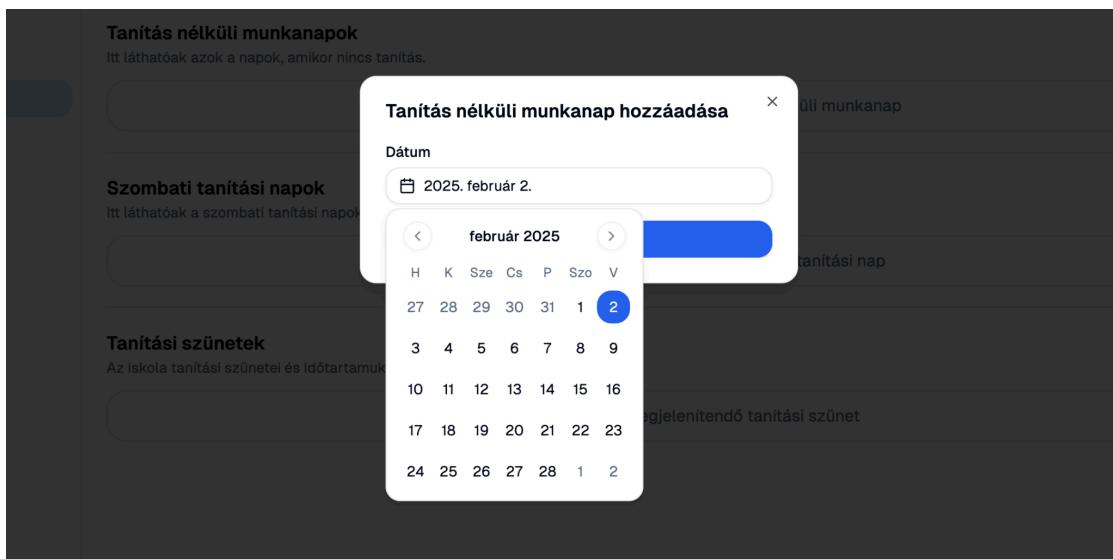
Tanítási szünetek

Az iskola tanítási szünetei és időtartamuk.

Új szünet hozzáadás

Név	Időtartam	Művelet
Tavaszi szünet	2025. 04. 14. - 2025. 05. 20.	

”Új nap hozzáadás” gombot használva adjunk hozzá egy tanítási nélküli munkanapot a rendszerhez. A dátum kiválasztása után a “Mentés” gombbal hozzá is adódik és egyből megjelenik a táblázatban az újonnan hozzáadott nap. Törlésre is lehetőség van, ha például rossz dátum lett kiválasztva. A tanítási szünetek és szombati tanítási napok hozzáadása is hasonló elven történik.



3.10 Oldalak és pozíciók

A rendszergazdára vonatkozó jogosultságokat a dokumentáció korábbi része már részben ismertette, azonban az alábbiakban részletesebben bemutatásra kerülnek a rendszergazda, igazgató, igazgatóhelyettes valamint tanár, osztályfőnök és portás jogosultságai is.

Az oldalsávon, minden felhasználó számára elérhető a “Kezdőlap” gomb, az oldalon az adott nap dátuma kerül mindig kijelzésre, illetve a rendszer státusza, valamint a rendszerbe feltöltött tanulók és a szekrényekben jelenleg eltárolt telefonk.

The screenshot shows the 'Kezdőlap' (Home Page) of a school management system. At the top, it displays the date as '2025. április 5.' (Sunday). Below this, there are four main sections: 'Mai dátum' (Today's date), 'Rendszer állapota' (System status), 'Tanulók száma' (Number of students), and 'Tárolt eszközök' (Stored devices). Under 'Segítség és támogatás' (Help and support), there are links for 'Használati útmutató' (User manual), 'GYIK' (FAQ), 'Kapcsolat' (Contact), and 'Oktatóvideók' (Educational videos).

3.10.1 Rendszergazda, igazgató és igazgatóhelyettes

Kiskunfélegyházi Szent
Benedek PG Középiskola

AdPg

A jogosultságai megegyeznek a rendszergazdának, igazgatónak és igazgatóhelyettesnek is, nekik teljes körű hozzáférésük van minden adathoz.

[Kezdőlap](#)

Iskolai nyilvántartás

[Órarendek](#)

[Tanulók](#)

[Munkatársak](#)

Beállítások és naplázás

[Tanév beállításai](#)

A “Tanév beállításai” oldalon a beállthatók a tanévhez kapcsolódó legfontosabb dátumok, például a tanév első és az utolsó napja, valamint a szünetek, szombati tanítási napok és munkaszüneti napok. Ezen funkcióról a dokumentáció korábbi részében már esett szó így az most itt nem kerül részletezésre.

Az “Órarendek” oldalon megtekinthető bármely osztály és bármely tanár órarendje, valamint engedélyezhető egy adott osztály/csoport szekrényeinek nyitása az adott tanórán.

A bal felső sarokban olvasható el a jelenlegi hét első és utolsó napjának dátuma. A jobb oldalon a legördülő menüből lehet választani az órarendek között, a nyilak segítségével lehet a hetek között váltani. Az “13.I” osztály órarendjének kiválasztása:

Kezdőlap > Iskolai nyilvántartás > Órarendek

2025 április 07. - 2025 április 13.

	H 7	K 8	Sze 9	Cs 10	P 11	Sz	V 13
07:15							
08:00 08:10							
08:55 09:05							
09:50 10:00							
10:45 10:55							
11:40 11:55							
12:40 12:55							
13:40							

Válasszon... Mai nap < >

Osztályok

- nincs
- 13.I
- 10.B
- 9.C
- 13.D
- 12.D
- 10.D
- 9.I
- 9/Knyc
- 10.I

2025 március 31. - 2025 április 06.

13.I Mai nap < >

	H 31	K 1	Sze 2	Cs 3	P 4	Szo 5	V 6
07:15	1 HaAt Háló... 13.I-I1	1 KiGl Front... 13.I-S	1 KuTi Munk... 13.I-A1	1 PaJu Munk... 13.I-A2	1 KuTi Ango... 13.I-A1	1 PaJu Ango... 13.I-A2	
08:00	2 HaAt Háló... 13.I-I1	2 KiGl Front... 13.I-S	2 KuTi Munk... 13.I-A1	2 PaJu Munk... 13.I-A2	2 KuTi Ango... 13.I-A1	2 PaJu Ango... 13.I-A2	1 HaAt Háló... 13.I-I1
08:10							1 KiGl Back... 13.I-S
08:55	3 HaAt Háló... 13.I-I1	3 KiGl Front... 13.I-S	3 HaAt Háló... 13.I-I1	3 PaZo Adat... 13.I-S	3 BeMo Osztályfőnöki 13.I	3 KuTi Ango... 13.I-A1	3 PaJu Ango... 13.I-A2
09:05							3 HaAt Háló... 13.I-I1
09:50	4 HaAt Háló... 13.I-I1	4 KiGl Front... 13.I-S	4 HaAt Háló... 13.I-I1	4 PaZo Adat... 13.I-S	4 BeMo Testnevelés 13.I	4 GrEr Hittan 13.I	4 SiTa Back... 13.I-S
10:00							4 KiGl Háló... 13.I-I1
10:45	5 KoDe Fehlő... 13.I-I1	5 PaZo Aszt... 13.I-S	5 HaAt Háló... 13.I-I1	5 PaZo Aszt... 13.I-S	5 KuTi Ango... 13.I-A1	5 PaJu Ango... 13.I-A2	5 BeMo Testnevelés 13.I
10:55							5 KiGl Back... 13.I-S
11:40	6 KoDe Fehlő... 13.I-I1	6 PaZo Aszt... 13.I-S	6 HaAt Háló... 13.I-I1	6 PaZo Aszt... 13.I-S		6 BeMo Testnevelés 13.I	6 SiTa Back... 13.I-S
11:55							6 KiGl Háló... 13.I-I1
12:40	7 KoDe Fehlő... 13.I-I1	7 PaZo Aszt... 13.I-S	7 KiGl Front... 13.I-S	7 KoDe Szer... 13.I-S			7 HaAt IKT projektmun...
12:55							

Ha éppen tart egy tanóra akkor az órarendben kiemelten sötétkék színnel jelenik meg, miután becsöngették egy adott órára. Mindig csak a éppen tartó órákat lehet kezelni. Egy megnyíló ablakban lehet kezelni az adott csoport/osztály tanulóiit.

A "Nyitás engedélyezése" gombbal lehet az egész csoport/osztály tanulóinak engedélyezni a szekrényük nyitását a tanóra alatt.

💡 Nyitás engedélyezése

A tanulók nevei mellett láthat kör alakú színes ikonok jelzik a szekrények státuszát:

- ✓ Telefon bent van a szekrényben
- ⊖ Nincs semmi a szekrényben
- ❗ Nincs a tanulóhoz szekrény beállítva

Illetve a 💡 ikonnal lehet csak az annak az egy tanulónak engedélyezni a szekrényének nyitását.

A “Tanulók” oldalon megtekinthető az összes tanuló. A nevük, az osztályuk és csoportjaik alapján lehet keresni a táblázatban. A “Feloldás/Korlátozás” gombbal lehet feloldani a zárolást az összes szekrényről illetve, korlátozni, hogy csak órarendjük alapján bírják a tanulók kivenni és betenni a telefonjaikat. A státusz ikonok elve megegyező az “Órarendek” oldalon lévővel.

Keresés név szerint...
Keresés osztály szerint...
 Feloldás
 Új tanuló hozzáadás

Teljes név ↑	Osztály és csoportok↑	Sztárusz	Műveletek
Bíró Boglárka	10.I,10.I-A1	⊖	  
László Ádám Zétény	10.I,10.I-A1	⊖	  
Karcagi Levente	9.I,9.I-N,9.I-A1	⊖	  
Katona Csaba	13.A,13.A-BN	⊖	  
Krepál Krisztián	10.D,10.D-A1	⊖	  
Csanád Miklós	11.D,11.D-emat,11.D-A1,11.D-nemat	⊖	  
Benke István	2/12.CNC,2/12.CNC-M	⊖	  
Pál Nikolett	9/Knya,9/Knya-N2	⊖	  
Beregi Milán	9.C,9.C-A1	⊖	  
Tamás Ákos Sámuel	12.A,12.A-N2	⊖	  
Szotyi Szandra	12.B,12.B-N	⊖	  
Dániel Attila	9/Knyc,9/Knyc-A1	⊖	  
Lippai Dániel	9.C,9.C-A1,9.C-GY2	⊖	  
Dávid László	10.I,10.I-A1	⊖	  

< Előző
1 / 72
Következő >

Az alábbi műveletek végezhetőek el egy adott tanulóval:

 Szekrény nyitás engedélyezés

 Tanuló szerkesztése

 Tanuló törlése

Tanuló hozzáadása

Azonosító szám
OM1234567

Teljes név
Teszt Elek

Osztály és csoportok
9.I,9.I-A2

RFID azonosító
R6HF6K86

Mentés

A tanulókat szerkeszteni a gombra kattintva megnyíló ablakban lehet, az adatok módosítása után a “Mentés” gombbal lehet elmenteni a változtatásokat.

Új tanulót létrehozni pedig az Új tanuló hozzáadás gombbal lehet, a megnyíló ablakban, ki kell tölteni minden adatot majd a “Mentés” gombbal véglegesíteni kell, az új tanuló egyből megjelenik a táblázatban.

A “Munkatársak” oldalon megtekinthető az összes alkalmazott. A nevük, a pozíciójuk és osztályaik alapján lehet keresni a táblázatban.

Keresés név szerint...	Keresés pozíció szerint...	Keresés osztály szerint... ▾	Új alkalmazott hozzáadás
Teljes név ↑	Pozíció ↑↓	Osztály ↑↓	Műveletek
Admin Pg (AdPg)	rendszerelő	nincs	
Alács Gyula László (AlGy)	Tanár	nincs	
Balázs László (BaLa)	Tanár	nincs	
Preiszné Bene Mónika Melinda (BeMo)	Tanár	13.I	
Erdélyi Szabolcs (ErSz)	Tanár	10.B	
Áron-Görög Szilvia (AGSz)	Tanár	9.C	
Bogácsi Csaba Mihály (BoCs)	Tanár	nincs	
Bozó Beáta (BoBe)	Tanár	13.D	
Buri Gábor Sándor (BuGa)	Tanár	nincs	
Czakóné Kállai Ildikó (CzKa)	Tanár	nincs	
Csenkiné Bihal Mária Magdolna (CsBM)	Tanár	nincs	
Csósz Beáta (CsBe)	Tanár	12.D	
Csuka Tibor (CsTi)	Tanár	10.D	
Dobos László (DoLa)	Tanár	nincs	

< Előző 1 / 5 Következő >

Az alábbi műveletek végezhetőek el egy adott alkalmazottal:

- ∅ Alkalmazott szerkesztése
- ⊖ Alkalmazott törlése

Alkalmazott szerkesztése

Teljes név

Pozíció

Mentés

A rendszergazdát, és órát tartó tanárokot törölni nem engedélyezett, valamint az igazgatót és igazgatóhelyettesét sem.

A tanulókat szerkeszteni ∅ gombra kattintva megnyíló ablakban lehet, az adatok módosítása után a “Mentés” gombbal lehet elmenteni a változtatásokat.

Alkalmazott hozzáadása

Teljes név

Rövidített név (felhasználónév)

Ideiglenes jelszó

Pozíció

Mentés

Új alkalmazottat létrehozni pedig az **⊕ Új alkalmazott hozzáadás** gombbal lehet, a megnyíló ablakban, ki kell tölteni minden adatot majd a “Mentés” gombbal véglegesíteni kell, az új alkalmazott egyből megjelenik a táblázatban.

3.10.2 Tanár

Kiskunfélegyházi Szent
Benedek PG Középiskola

BaLa

Kezdőlap

Saját órák

A tanár csak a saját óráit látja, amit az oldalsávon a “Saját órák” gombbal érhet el. Ezen az oldalon megtekintheti az órarendjét, valamint engedélyezheti egy adott osztály/csoport szekrényeinek nyitását az adott tanórán. A működés megegyezik a “Órarendek” oldalon lévővel.

3.10.3 Osztályfőnök

Kiskunfélegyházi Szent
Benedek PG Középiskola

BeMo

Kezdőlap

Saját órák

Osztályom

Órarend

Tanulók

Az a tanár akinek van saját osztálya, az a saját óráin túl az osztályának órarendjét és az összes tanulját is láthatja, amit az oldalsávon az "Órarend" és "Tanulók" gombbal érhet el (1. ábra). Ezeknek az oldalak a működésük és felületük nagyban megegyeznek azokkal az oldalakkal amiket a rendszerelő, igazgató és igazgatónövöttes lát, viszont kevesebb jogosultsággal.

Az "Órarend" oldalon megtekintheti az osztályának órarendjét, valamint engedélyezheti a osztály/csoport szekrényeinek nyitását az adott tanórán.

A "Tanulók" oldalon megtekinthető a osztály összes tanulója. A nevük alapján lehet keresni a táblázatban. Az osztályfőnök nem törölhet és nem szerkeszthet tanulót, csak a szekrény nyitását engedélyezheti (3. ábra).

Kezdőlap > Osztályom > Tanulók

Keresés név szerint...

Teljes név ↑↓	Osztály és csoportok↑↓	Státusz	Műveletek
Fábi Balázs	13.I,13.I-A2,13.I-S		
Kristóf István Gyula	13.I,13.I-II		
Margit Rozália Lili	13.I,13.I-S		
Márk Zétény Levente	13.I,13.I-S,13.I-II,13.I-A2		
Elemér Róbert	13.I,13.I-S		
Kőszegi Mirkó	13.I,13.I-S		
Vajda Kamilla	13.I,13.I-S,13.I-A1		
Nagybányai Kevin	13.I,13.I-A2,13.I-S		
Enikő Hella Dorina	13.I,13.I-A1		
Fóti László	13.I,13.I-S,13.I-II,13.I-A2		
Dániel Károly Márton	13.I,13.I-A1,13.I-II,13.I-S		
Fráter Arnold	13.I,13.I-A1		
Bence Tamás	13.I,13.I-A1		
Magdolna Diána Anasztázia	13.I,13.I-A2,13.I-II		

< Előző

1 / 4

Következő >

3.10.4 Portás

Kiskunfélegyházi Szent
Benedek PG Középiskola

TePo

Kezdőlap

Iskolai nyilvántartás

Tanulók

A portás csak a tanulókat látja, amit az oldalsávon a “Tanulók” gombbal érhet el. Az oldal a működése és felülete megegyezik azzal amit a rendszer szolgáltat, igazgató és igazgatóhelyettes lát, ugyanazzal a jogosultsággal.

3.11 Telefontároló működése

3.11.1 Telefon elhelyezése

A tanítás megkezdése előtt a kártyát vagy a bilétát hozzá kell érinteni a szekrény olvasójához. A rendszer felismeri az a kártyát vagy a bilétát, majd automatikusan kinyitja a hozzárendelt szekrényt. A telefon elhelyezése után a szekrényt be kell csukni. Amíg a szekrény nincs megfelelően visszazárva, a rendszer nem engedélyez további kártya vagy biléta beolvasást.

3.11.2 Használat tanítási idő alatt

Tanítási időben a szekrények zárolt állapotban vannak. A rendszer nem engedélyezi a nyitást, még az érvényes kártya vagy a biléta használatával sem. Kivételt kizárolag külön engedély alapján lehet tenni. Az engedélyezést egy tanár, igazgató, igazgatóhelyettes vagy rendszer szolgáltató végezheti el a rendszer webes felületén keresztül. Az engedély kizárolag az adott tanuló szekrényére érvényes.

3.11.3 Telefon kivétele

A tanítási nap végén a rendszer automatikusan feloldja a zárolást. A tanuló újra hozzá érinti a szekrény olvasójához a kártyát vagy a bilétát, majd pedig rendszer kinyitja a hozzárendelt szekrényt. A telefon kivétele után a szekrényajtót ismét be kell zárni.

3.11.4 Rendszer állapota

Alapértelmezett állapotban a rendszer „Korlátozott” módban működik. Ez azt jelenti, hogy a szekrények csak az órarend alapján, vagy az arra jogosult alkalmazottak engedélyével nyithatók.



Amennyiben a rendszer „Feloldott” állapotba kerül, a szekrények bármikor szabadon nyithatók, így a telefonok bármikor kivehetőek.

