A képen szöveg látható

Automatikusan generált leírás

NAPLÓ

*/gyakorlat beadandó/*

BALOGH NORBERT

I2I25Q

TORONTÁLI ESZTER

J3GE0B

2024

Tartalomjegyzék

[1. Az ellenőrzéshez szükséges linkek 3](#_Toc183951735)

[3. Project felépítése 4](#_Toc183951736)

[4. Felhasználói szerepkörök kezelése 5](#_Toc183951737)

[RESTful API 8](#_Toc183951738)

1. Az ellenőrzéshez szükséges linkek

GitHub link: <https://github.com/NickDale/naplo>

GitHub felhasználók:

* Torontáli Eszter – ***GHEszti***
* Balogh Norbert – ***NickDale***

A feladat felosztásokat és követéseket a GitHub projekt segítségével valósítottuk meg.

<https://github.com/users/NickDale/projects/3>

1. **Feladat meghatározása**

Egyre több iskolában használnak elektronikus naplót. Ebben a naplóban – sok más egyéb mellett – rögzítik az osztályzatokat is. A feladatban szereplő adatbázis a Városvégi tartal-mazza. Tudjuk, hogy az adathalmazban nincs két azonos nevű személy. Gimnázium elektronikus naplójának egy tanév első négy hónapjában bejegyzett jegyeit.

A jegy és a tárgy tábla között egy a többhöz kapcsolat van, mivel egy tárgyhoz több jegy is kapcsolódhat, azonban egy jegy az biztosan csak egy tárgyhoz tartozik. A jegy diák táblák között szintén egy a többhöz kapcsolat, mivel egy diákhoz több jegy is tartozik, de egy jegy az egy konkrét diákhoz tartozik.

A következő kép szemlélteti a feladat során elkészített adatbázis diagramja, az egyes táblák közötti kapcsolattal.

A képen szöveg, képernyőkép, Betűtípus, diagram látható

Automatikusan generált leírás

1. Project felépítése

A project megvalósítása során a következő technológiákat, nyelveket, toolokat használtuk.

* Spring boot 3.4.0
* JAVA 17
* maven
* MySQL adatbázis
* Használt dependencyk
  + Spring Data JPA az adatbázis leképezéséhez és az adatbázis műveletekhez
  + Spring Security a jogosultság kezeléshez
  + Thymeleaf template engine a megjelenítéshez

A spring bootnak köszönhetően a *JPA* és a T*hymeleaf* configurációt elegendő volt az *application.propertiesben* megadni és a keretrendszer létrehozta a szükséges beaneket, konfigurációkat.

Az adatbázis elérési URL-t a feladatkiírásban feltüntetett módon állítottuk be, így a MYSQL\_HOST könnyen lecserélhető teszteléshez egy másik elérési útvonalra.

A projektet legjobb tudásunk szerint igyekeztünk strukturáltan felépíteni, ahol MVC-t, vagyis a vizuális megjelenítést szolgáló API-kat és a REST API-kat is elkülönítettük egy mástól.  
A REST API-k kaptak egy /api/…. előtagot is. Ezek a REST standardnak megfelelően application/json formátumban adnak vissza adatokat a megfelelő HTTP státuszkódok használata mellett.

A REST API-kat a @RestController, míg a felület által használtakat a @Controller Spring komponensek biztosítják.

//TODO képcsere

1. Felhasználói szerepkörök kezelése

3 különböző felhasználói szerepkört hoztunk létre, az adminisztrátort, a tanárt, illetve a diákot.

Regisztrációnál szükséges a szerepkört beállítani, ez alapján végezzük később az adatbázisban szereplő jogkör vizsgálatot.

A képen szöveg, képernyőkép, Betűtípus, szám látható

Automatikusan generált leírás

1. ábra - felhasználó regisztráció

Regisztráció után bejelentkezésnél a felhasználó jogosultsági körét is azonosítjuk.

Helytelen adatok megadása esetén sikertelen a bejelentkezés.

A képen szöveg, képernyőkép, Betűtípus, szám látható

Automatikusan generált leírás

2. ábra - bejelentkezési felület

Bejelentkezett felhasználókat egy kezdőlap fogad, melyen tudnak tájékozódni az iskoláról, illetve a napló nyújtotta szolgáltatásokról, majd a menüben az általuk megtekinteni kívánt oldalra tudnak tovább haladni.

Egy funkció van az oldalon, amely bejelentkezés nélkül is elérhető a látogatóknak, ez pedig az üzenetküldő felület.

A képen szöveg, képernyőkép, Betűtípus, Téglalap látható

Automatikusan generált leírás

3. ábra - üzenetküldő funkció

4. ábra - Admin kezdőlap - TODO kicserélni

A tanároknak lehetőségük van tárgyak, diákok megtekintésére, illetve a jegyek teljeskörű (hozzáadás, módosítás, törlés) kezelésére. Itt van egy plusz funkció, amit admin felületen indokolatlannak éreztem, ez pedig a diákokra szűrés és a jegyeik exportálása pdf fileba.

A képen szöveg, képernyőkép, Betűtípus, Téglalap látható

Automatikusan generált leírás

5. ábra - Tanárok kezdőlap - TODO kicserélni

A 3. kategória, pedig a diákok, akik csak listázni tudják a tárgyakat, diákokat, illetve a jegyeket. Továbbá itt is lehetőség nyílik a diákok jegyeik exportálására.

# Külső reszponzív sablon alkalmazása

Projektemhez Bootstrapet használtam, mely segített a ez ergonomikus design kialakításában. Továbbá JavaScriptet használtam például a táblázatok megjelenítéséhez jquery datatables-t hívtam segítségül.

A képen szöveg, képernyőkép, Betűtípus látható

Automatikusan generált leírás

6. ábra - boostsrap, js hivatkozások - TODO kicserélni

1. RESTful API

Az alkalmazásban diák táblára valósítottuk meg, az alatvető CRUD műveleteket a RESTful elveknek megfelelően.

A megvalósított REST API-k kaptam egy /api/.. előtagot, ezzel jól elkülöníthetőek a MVC Controllerektől.

* 1. GET

Lekérdezésekhez használatok HTTP metódus

A feladatban 2 db GET lekérdezést valósítottunk meg.

5.1.1 Összes tanuló lekérdezése

HTTP metódus : GET

URL : /api/students/

Válasz: Listában visszaadja az adatbázisban szereplő diákok adatait.

Postman test:

A képen szöveg, képernyőkép, szoftver, Multimédiás szoftver látható

Automatikusan generált leírás

cUrl test:

A képen képernyőkép, szöveg látható

Automatikusan generált leírás

5.1.2 Egy adott tanuló lekérdezése (ID alapján)

HTTP metódus : GET

URL : /api/students/:studentId

Válasz: Visszaadja az adott ID-val rendelkező diák adatait.

postman test:

A képen szöveg, képernyőkép, szoftver, Multimédiás szoftver látható

Automatikusan generált leírás

cUrl teszt:

A képen szöveg, képernyőkép, Betűtípus, szoftver látható

Automatikusan generált leírás

* 1. POST

A POST metódus használatával egy új diákot tudunk felvenni a rendszerünkbe.

HTTP metódus : POST

URL : /api/students/

Válasz: Visszaadja a felvett diák adatait, beleértve az adatbázis által generált ID-t is.   
Ha sikeresen mentésre került a diák, akkor 201-es (CREATED) státuszkóddal tér vissza.

postman teszt:

A képen szöveg, képernyőkép, szoftver, Multimédiás szoftver látható

Automatikusan generált leírás

cUrl teszt:

A képen szöveg, Betűtípus, képernyőkép látható

Automatikusan generált leírás

* 1. PUT

A PUT metódus használatával egy már meglévő diák adatait tudjuk updatelni.

HTTP metódus : PUT

URL : /api/students/:id

Válasz: 200 OK státuszkóddal visszaadja az módosított diák adatait

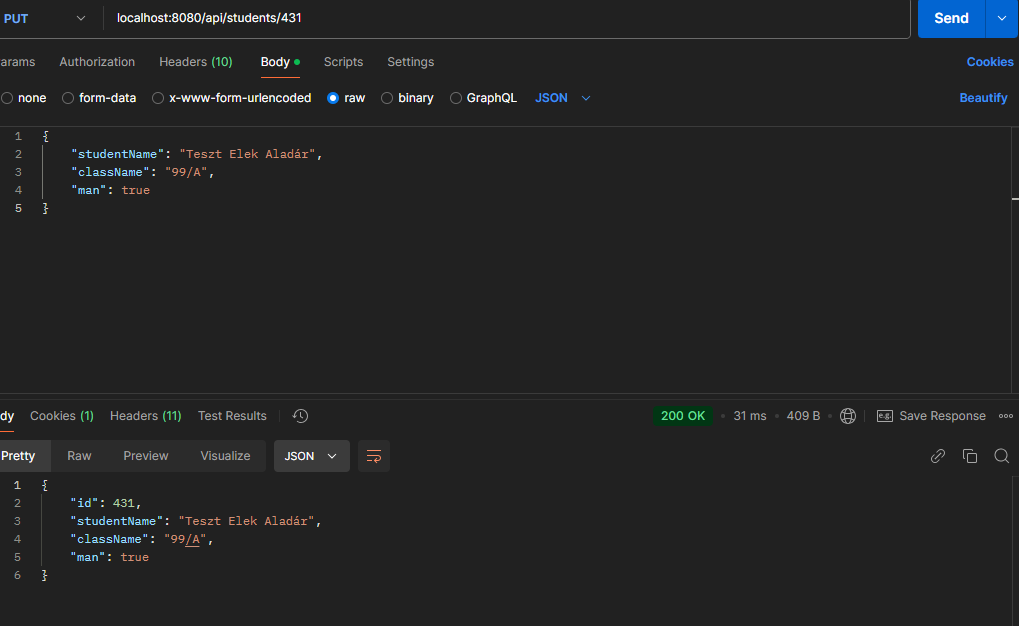
A teszt során az előzőleg létrehozott Teszt Eleket és a Teszt Elek\_curl diákot fogjuk módosítani.

Ha esetleg olyan diák adatait próbáljuk módosítani, akinek nem létezik az Id-ja az adatbázisban, akkor a következő hibát kapjuk.

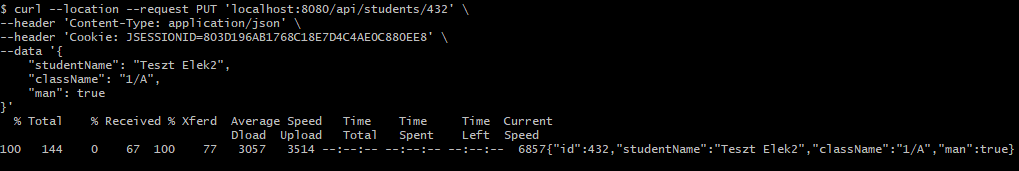
A képen szöveg, szoftver, Multimédiás szoftver, képernyőkép látható

Automatikusan generált leírás

postman teszt:



cUrl teszt:



* 1. DELETE

A delete metódust használjuk, ha egy diákot szeretnénk törölni a rendszerből.

Ilyenkor az adott diák az ID alapján törlésre kerül. Az előző két példával fogjuk tesztelni.

HTTP metódus : DELETE

URL : /api/students/:id

Sikeres törlést követően csak egy 204 (No content) HTTP státuszkóddal rendelkező választ kapunk vissza.

postman teszt:

A képen szöveg, képernyőkép, szoftver, Multimédiás szoftver látható

Automatikusan generált leírás

cUrl:

Természetesen törölni csak egyszer tudjuk, így ha újra megpróbáljuk, akkor szól a rendszer, hogy nem található ilyen diák.  
A képen az első a sikeres törlés, a második már a figyelmeztetést mutatja.  
