PROIECT DE PRACTICA

***Tema proiectului: STUDENT’S MANAGEMENT APPLICATION***

**Cadrul didactic coordonator**: TIBERIU MARIAN GEORGESCU

**Student**: ALEXANDRU ARITON

**București**

**2023**

Cuprins

[Introducere 3](#_Toc135682384)

[Prezentarea generală a proiectului și tehnologiilor folosite 4](#_Toc135682385)

[1. Structura pachetelor și clasei: 4](#_Toc135682386)

[2. Controller-ere: 4](#_Toc135682387)

[3. Entități: 4](#_Toc135682388)

[4. Repository-uri: 4](#_Toc135682389)

[5. Configurarea bazei de date: 4](#_Toc135682390)

[6. Utilizarea Spring Framework: 4](#_Toc135682391)

[7. Utilizarea Eclipse IDE: 5](#_Toc135682392)

[Prezentarea detaliată a proiectului 5](#_Toc135682393)

[A. CONTROLLER 5](#_Toc135682394)

[B. ENTITY 7](#_Toc135682395)

[C. REPOSITORY 8](#_Toc135682396)

[D. IMPORTURILE 8](#_Toc135682397)

[a) Pentru clasa **„Controller”:** 8](#_Toc135682398)

[b) Pentru clasa **„Entity”:** 10](#_Toc135682399)

[c) Pentru clasa **“Repository”:** 11](#_Toc135682400)

[E. ALTE APLICATII FOLOSITE 11](#_Toc135682401)

[i. MySQL 11](#_Toc135682402)

[ii. Postman 11](#_Toc135682403)

[CONCLUZII 12](#_Toc135682404)

[EXEMPLIFICAREA FUNCTIONARII CODULUI PRIN POZE 12](#_Toc135682405)

[I. POST 13](#_Toc135682406)

[II. GET 14](#_Toc135682407)

[III. PUT 14](#_Toc135682408)

[IV. DELETE 16](#_Toc135682409)

# Introducere

Stagiul meu de practică a fost susținut în cadrul Ministerului Finanțelor, care asigură o administrare eficientă a finanțelor publice, colectarea și gestionarea veniturilor și cheltuielilor guvernamentale, precum și elaborarea și implementarea politicilor fiscale și bugetare, în departamentul informatic. Acest departament are rolul de a asigura funcționarea eficientă a sistemelor informatice și implementarea de soluții tehnologice inovatoare pentru îmbunătățirea proceselor financiare și fiscale.

Am avut oportunitatea să folosesc limbajul de programare Java, prin intermediul programului Eclipse, și framework-ul Spring pentru a dezvolta o aplicație de gestionare a informațiilor despre studenți. Aplicația permite adăugarea, citirea, actualizarea și ștergerea datelor despre studenți într-o bază de date.

Motivul pentru care am ales să fac stagiul de practică în departamentul IT al Ministerului Finanțelor este pentru a învăța și dezvolta abilități și competențe în lucrul cu tehnologii și instrumente avansate și pentru a câștiga experiență într-un domeniu critic.

# Prezentarea generală a proiectului și tehnologiilor folosite

Pentru realizarea proiectului au fost necesare următoarele:

1. Structura pachetelor și clasei: Aplicația are o structură organizată în pachete și clase coerente. Acestea includ pachete precum **com.example.proiect.controller** pentru controlerele care gestionează cererile HTTP, **com.example.proiect.entity** pentru entitățile care reprezintă obiectele din baza de date, **com.example.proiect.repo** pentru repository-urile care efectuează operațiile CRUD.
2. Controller-ere: Aplicația are un controller dedicat gestionării operațiilor CRUD pentru entitatea "Student". Acest controller conține metodele necesare pentru a trata cererile HTTP, cum ar fi adăugarea unui student nou, obținerea detaliilor unui student existent, actualizarea informațiilor despre un student și ștergerea unui student.
3. Entități: Pentru a reprezenta informațiile despre studenți în baza de date, aplicația utilizează clasa "Student" care este mapată la o tabelă în baza de date. Această clasă conține atribute relevante pentru un student, cum ar fi nume, adresă și email.
4. Repository-uri: Aplicația folosește un repository care gestionează interacțiunea cu baza de date pentru entitatea "Student". Acest repository extinde interfața JpaRepository furnizată de Spring Data JPA și oferă metode predefinite pentru operațiile CRUD, cum ar fi salvarea unui student, găsirea tuturor studenților și găsirea unui student după ID.
5. Configurarea bazei de date: Aplicația include configurările necesare pentru a se conecta la o bază de date. Aceste configurări includ informații precum URL-ul bazei de date, numele utilizatorului și parola.
6. Utilizarea Spring Framework: Aplicația beneficiază de caracteristicile și avantajele oferite de Spring Framework, cum ar fi injecția de dependențe, care permite separarea strânsă între diferitele componente ale aplicației și facilitează testarea și reutilizarea codului. De asemenea, Spring oferă gestionarea tranzacțiilor, care simplifică lucrul cu operațiuni de persistență în baza de date, asigurând consistența datelor și revenirea la starea anterioară în caz de erori. Spring facilitează dezvoltarea aplicației prin furnizarea de componente și abstracții care simplifică dezvoltarea și integrarea diferitelor părți ale aplicației.
7. Utilizarea Eclipse IDE: Eclipse este un mediu integrat de dezvoltare (IDE) pentru Java și este folosit pentru a dezvolta și gestiona codul sursă al aplicației.

# Prezentarea detaliată a proiectului

## CONTROLLER

In package-ul „com.example.demo.controller”, se regăseste clasa „Controller” care a avut rolul de a gestiona cererile HTTP legate de entitatea "Student".

In continuare voi prezenta o descriere a funcționalităților oferite de acest controller:

* Adnotarea **@RestController**, ceea ce indică faptul că este responsabil pentru gestionarea cererilor HTTP și returneaza răspunsurile în format JSON.
* Dependența către repository-ul **Repository** este indicate de adnotarea **@Autowired**. Aceasta permite controllerului să interacționeze cu baza de date pentru a efectua operațiile CRUD asupra entității "Student".
* Metoda **saveStudent()** este atribuita cererilor HTTP de tip POST pentru URL-ul "/api/students". Această metodă primește un obiect de tip **Student** în format JSON prin intermediul adnotării **@RequestBody**, apoi se salvează studentul în baza de date utilizând repository-ul **studentRepo.save()** și se returnează un răspuns cu starea HTTP 201 (CREATED) și studentul salvat în corpul răspunsului.
* Metoda **getStudents()** este atribuita cererilor HTTP de tip GET pentru URL-ul "/api/students". Această metodă returnează o listă de studenți din baza de date utilizând repository-ul **studentRepo.findAll()**. Răspunsul este salvat într-un obiect **ResponseEntity** și este returnat cu starea HTTP 200 (OK).
* Metoda **getStudent()** este atribuita cererilor HTTP de tip GET pentru URL-ul "/api/students/{id}", unde {id} reprezintă identificatorul unic al studentului. Această metodă caută studentul cu ID-ul specificat utilizând repository-ul **repository.findById()**. Dacă studentul este găsit, acesta este returnat într-un răspuns cu starea HTTP 200 (OK). Dacă studentul nu există, se returnează un răspuns cu starea HTTP 404 (NOT\_FOUND).
* Metoda **updateStudent()** este atribuita cererilor HTTP de tip PUT pentru URL-ul "/api/students/{id}", unde {id} reprezintă identificatorul unic al studentului care trebuie actualizat. Această metodă caută studentul cu ID-ul specificat utilizând repository-ul **repository.findById()**. Dacă studentul este găsit, se actualizează câmpurile acestuia cu valorile furnizate în corpul cererii (**stud**) și se salvează în baza de date utilizând **repository.save()**. Apoi, se returnează un răspuns cu starea HTTP 200 (OK) și studentul actualizat. Dacă studentul nu există, se returnează un răspuns cu starea HTTP 404 (NOT\_FOUND).
* Metoda **deleteStudent()** este atribuita cererilor HTTP de tip DELETE pentru URL-ul "/api/students/{id}", unde {id} reprezintă identificatorul unic al studentului care trebuie șters. Această metodă caută studentul cu ID-ul specificat utilizând repository-ul **repository.findById()**. Dacă studentul este găsit, acesta este șters din baza de date utilizând **repository.deleteById()**, iar un răspuns cu starea HTTP 204 (NO\_CONTENT) este returnat. Dacă studentul nu există, se returnează un răspuns cu starea HTTP 404 (NOT\_FOUND).

## ENTITY

In package-ul „com.example.demo.entity” se regăseste clasa „Student”, care reprezintă o entitate în cadrul sistemului.

In interiorul aceste clase regăsim:

* Adnotarea“**@Entity**”, care indică faptul că clasa **Student** este o entitate JPA (Java Persistence API), ceea ce înseamnă că este mapată direct pe o tabelă din baza de date.
* **Adnotarea** “**@Id**”, care marchează câmpul **id** ca fiind cheie primară în tabela asociată entității **Student**.
* Adnotarea “**@GeneratedValue”**, care specifică strategia de generare a valorilor pentru câmpul **id**. În cazul de față, se utilizează strategia “**GenerationType.AUTO”**, ceea ce înseamnă că baza de date va genera automat valorile pentru câmpul **id**.
* Câmpurile **studentName**, **studentEmail** și **studentAdress** reprezintă atributele unui student.
* Metodele getter și setter sunt folosite pentru a obține și a seta valorile câmpurilor private.
* Constructorul cu toate câmpurile este utilizat pentru a crea obiecte **Student** cu toate atributele setate.
* Constructorul fără parametri este folosit pentru a crea obiecte **Student** fără a specifica inițial valorile atributelor.
* Metoda “**toString()”** este suprascrisă pentru a returna o reprezentare sub forma unui șir de caractere a obiectului **Student**.

## REPOSITORY

In package-ul „com.example.demo.repository” regăsim o interfață ***Repository*** care extinde interfața **JpaRepository** din Spring Data JPA. Această interfață servește ca un repository (o componentă a design pattern-ului Repository) pentru operațiile de acces la date legate de entitatea **Student**.Mai mult, aceasta primește două parametri de tip generic: **Student** reprezintă tipul entității cu care lucrează repository-ul (în acest caz, clasa **Student**), iar **Long** reprezintă tipul de date al ID-ului entității

Totodată oferă si metode predefinite precum **save()**, **findById()**, **findAll()**, **deleteById()** etc., care permit efectuarea operațiilor CRUD (Create, Read, Update, Delete) asupra entității **Student** în baza de date.

## IMPORTURILE

Pe lângă toate funcționalitățile pe care fiecare clasă le poate avea, atât conexiunea dintre ele, cât și posibilitatea realizării unora dintre funcționalități, nu ar fi fost posibilă fără ajutorul "importurilor", care au fost:

### Pentru clasa **„Controller”:**

* **import java.util.List;**: Importă clasa **List** din pachetul **java.util** pentru a utiliza tipul de date List.
* **import java.util.Optional;**: Importă clasa **Optional** din pachetul **java.util** pentru a utiliza tipul de date Optional.
* **import org.springframework.beans.factory.annotation.Autowired;**: Importă clasa **Autowired** din framework-ul Spring pentru a realiza injectarea de dependențe.
* **import org.springframework.http.HttpStatus;**: Importă clasa **HttpStatus** din framework-ul Spring pentru a utiliza codurile de stare HTTP în răspunsurile controlerului.
* **import org.springframework.http.ResponseEntity;**: Importă clasa **ResponseEntity** din framework-ul Spring pentru a încapsula răspunsurile HTTP.
* **import org.springframework.web.bind.annotation.DeleteMapping;**: Importă anotarea **DeleteMapping** din framework-ul Spring pentru a mapa cererile HTTP de tip DELETE la metodele controlerului.
* **import org.springframework.web.bind.annotation.GetMapping;**: Importă anotarea **GetMapping** din framework-ul Spring pentru a mapa cererile HTTP de tip GET la metodele controlerului.
* **import org.springframework.web.bind.annotation.PathVariable;**: Importă anotarea **PathVariable** din framework-ul Spring pentru a obține valoarea unei variabile din șablonul URI.
* **import org.springframework.web.bind.annotation.PostMapping;**: Importă anotarea **PostMapping** din framework-ul Spring pentru a mapa cererile HTTP de tip POST la metodele controlerului.
* **import org.springframework.web.bind.annotation.PutMapping;**: Importă anotarea **PutMapping** din framework-ul Spring pentru a mapa cererile HTTP de tip PUT la metodele controlerului.
* **import org.springframework.web.bind.annotation.RequestBody;**: Importă anotarea **RequestBody** din framework-ul Spring pentru a obține corpul cererii HTTP.
* **import org.springframework.web.bind.annotation.RestController;**: Importă anotarea **RestController** din framework-ul Spring pentru a marca clasa ca fiind un controler REST.
* **import com.example.proiect.entity.Student;**: Importă clasa **Student** care reprezintă entitatea cu care lucrează controlerul.
* **import com.example.proiect.repo.StudentRepo;**: Importă interfața **StudentRepo** care reprezintă repository-ul pentru entitatea **Student**.

### Pentru clasa **„Entity”:**

* **import jakarta.persistence.Entity,** careimportă clasa **Entity** din pachetul **jakarta.persistence**.
* **import jakarta.persistence.GeneratedValue,** care importă adnotarea **GeneratedValue** din pachetul **jakarta.persistence**. Această adnotare este utilizată pentru a specifica că valoarea cheii primare a entității **Student** va fi generată automat de către baza de date.
* **import jakarta.persistence.GenerationType**, care importă enumerarea **GenerationType** din pachetul **jakarta.persistence**. Această enumerare este utilizată împreună cu adnotarea **GeneratedValue** pentru a specifica modul în care se generează cheia primară a entității. În acest caz, **GenerationType.AUTO** indică faptul că modul de generare a cheii primare este ales automat de către implementarea JPA.
* **import jakarta.persistence.Id**, care importă adnotarea **Id** din pachetul **jakarta.persistence**. Această adnotare este utilizată pentru a marca câmpul **id** din clasa **Student** ca fiind cheie primară a entității.

Java Persistence API (JPA) este o componentă a Java Enterprise Edition (Java EE) utilizată pentru a gestiona persistența obiectelor în baze de date relaționale.

### Pentru clasa **“Repository”:**

* **import org.springframework.data.jpa.repository.JpaRepository,** care importă clasa **JpaRepository** din Spring Data JPA. Această clasă furnizează metode predefinite pentru operațiile de bază CRUD și interacțiunea cu baza de date pentru o entitate.

**import com.example.proiect.entity.Student,** careimportă clasa **Student** care reprezintă entitatea pentru care se definește acest repository

## ALTE APLICATII FOLOSITE

Totodată, pe lângă aplicația Eclipse, unde am instalat framework-ul Spring, am mai folosit și alte 2 aplicații:

1. MySQL pentru crearea bazei de date, care m-a ajutat să stochez studenții fără a pierde datele.
2. Postman, care a contribuit la testarea cererilor POST, GET, DELETE, PUT, datorită faptului că aceste cereri sunt trimise către un endpoint, cum ar fi "/api/students/{id}" și "/api/students", astfel, asigurându-te că toate cererile HTTP sunt tratate corespunzător și că răspunsurile sunt conforme cu așteptările. În plus, Postman va afișa răspunsul primit de la server, inclusiv codul de stare HTTP.

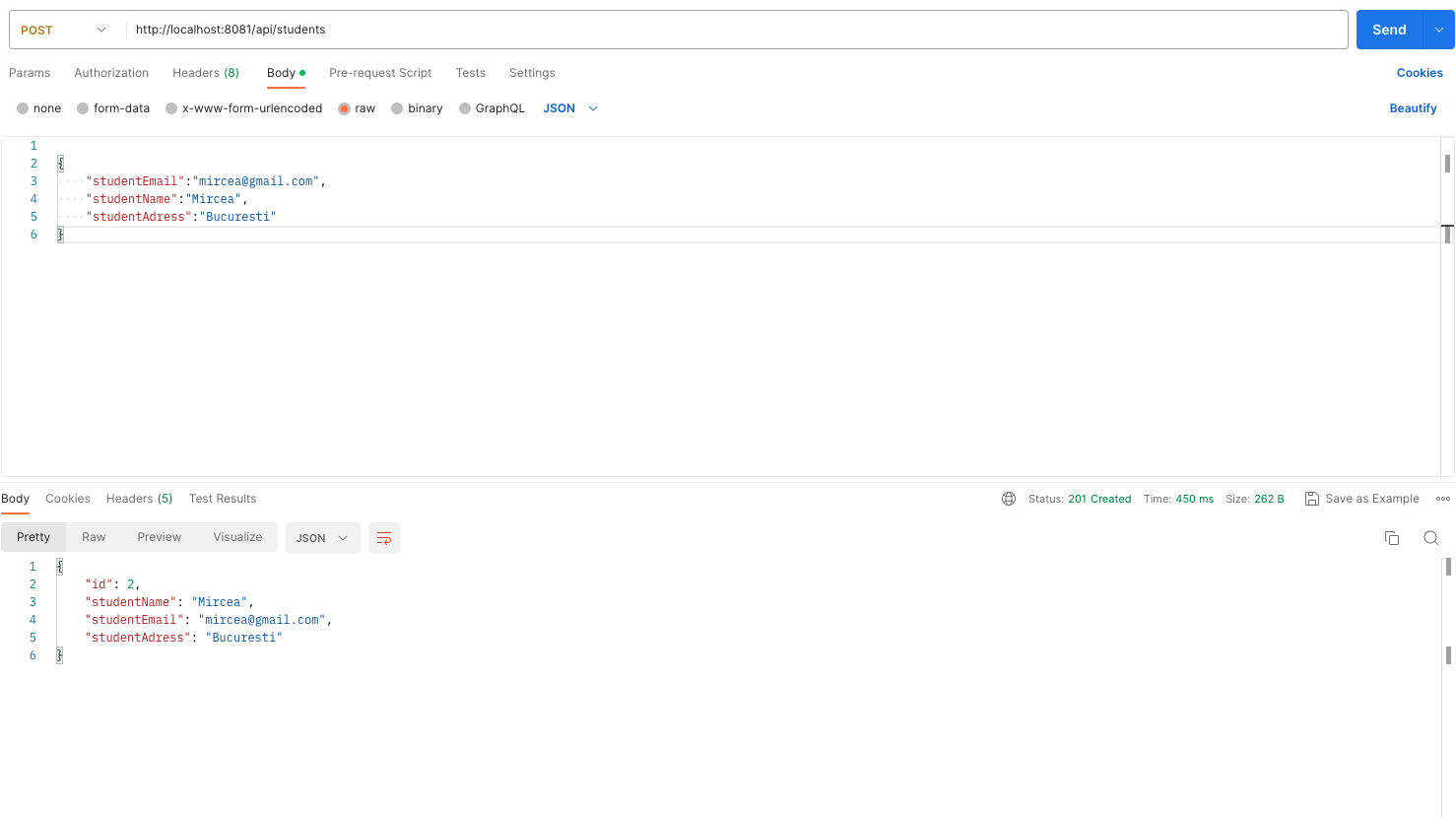
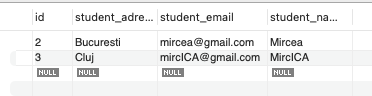
# CONCLUZII

Asadar, acest stagiu de practică a contribuit la dezvoltarea mea profesională prin faptul că am avut oportunitatea să stau lângă specialiștii din cadrul departamentului IT al Ministerului Finanțelor, acumulând informațiile și sfaturile pe care aceștia mi le dădeau, ceea ce cred că o să mă ajute mult în viitoarea mea carieră în domeniul IT.

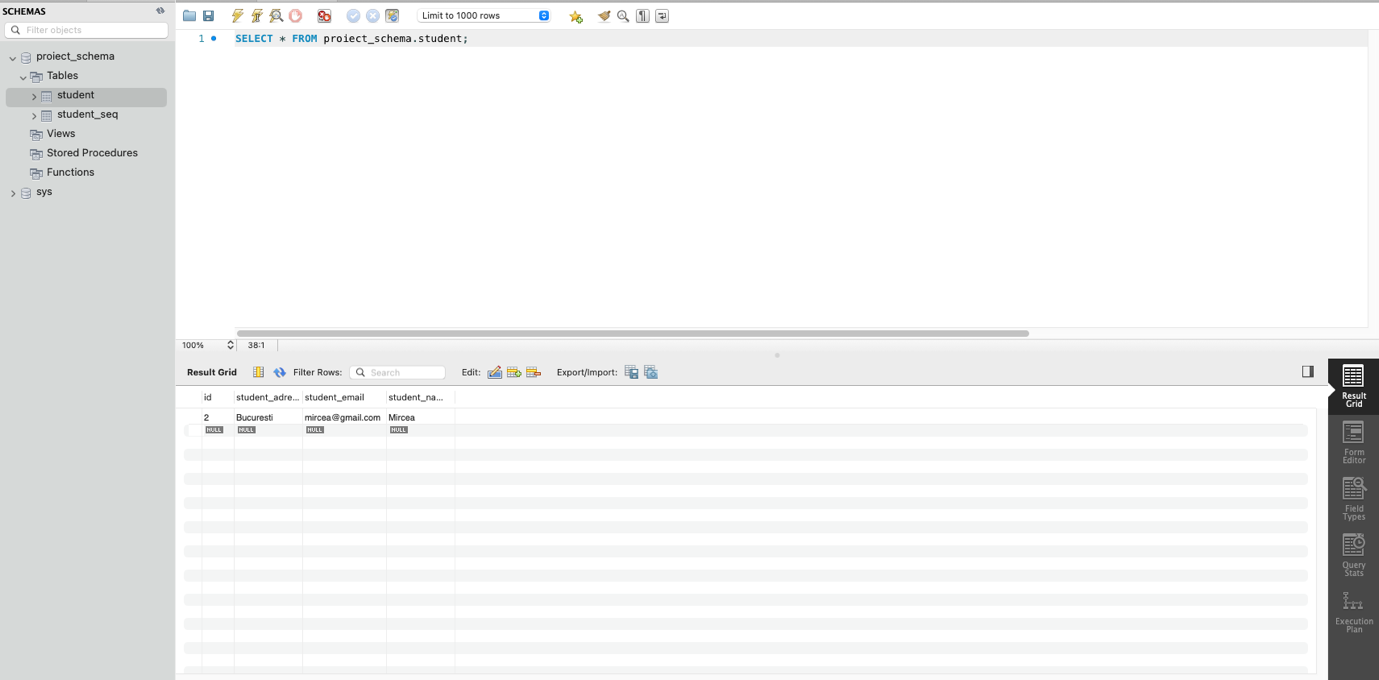
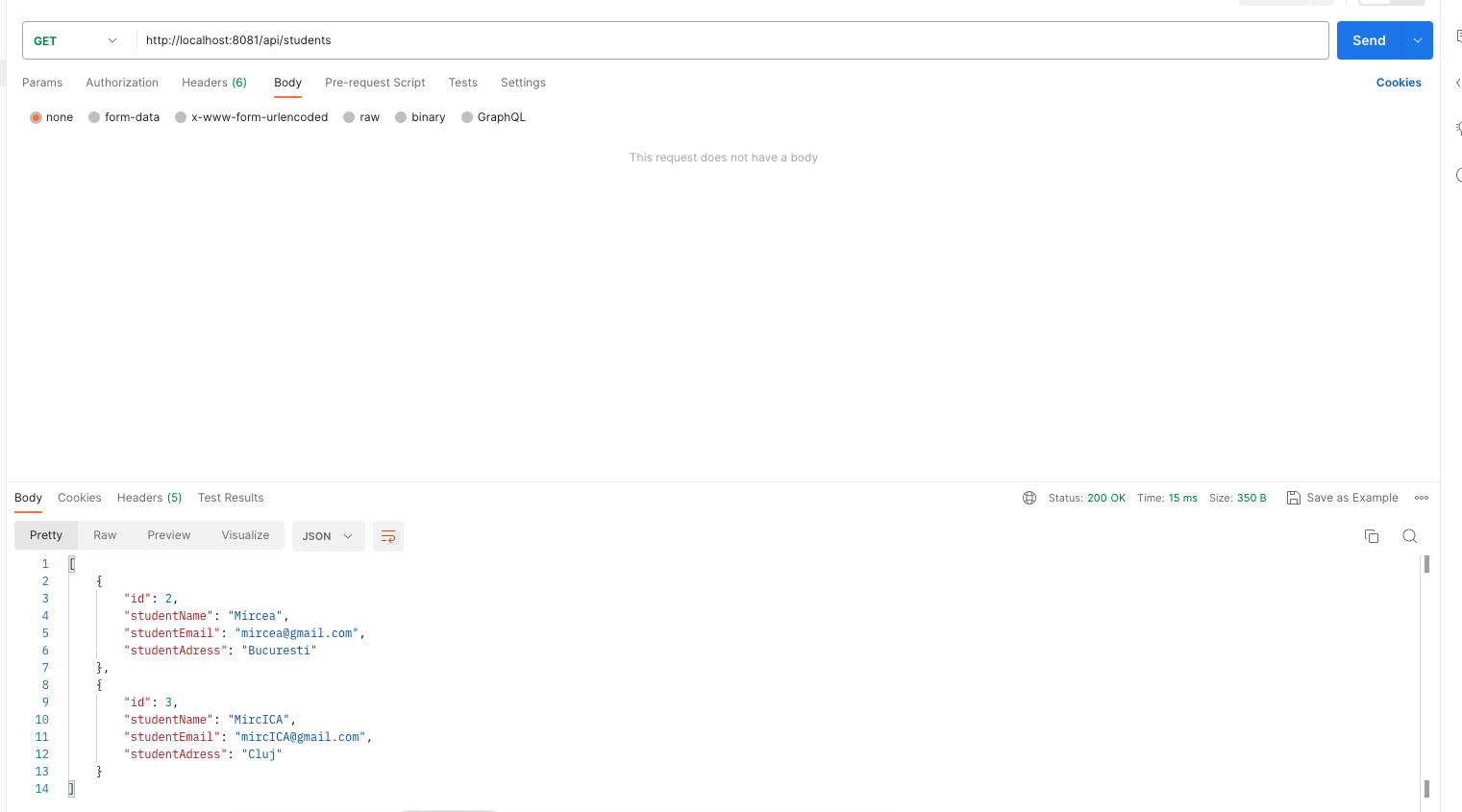
Din punct de vedere al aplicației, tind să sper că este doar un pilon pentru clădirea unei viitoare aplicații web care să aibă utilitate în viața reală mult mai mare decât o are prototipul pe care l-am expus eu pentru acest stagiu de practică.

# EXEMPLIFICAREA FUNCTIONARII CODULUI PRIN POZE

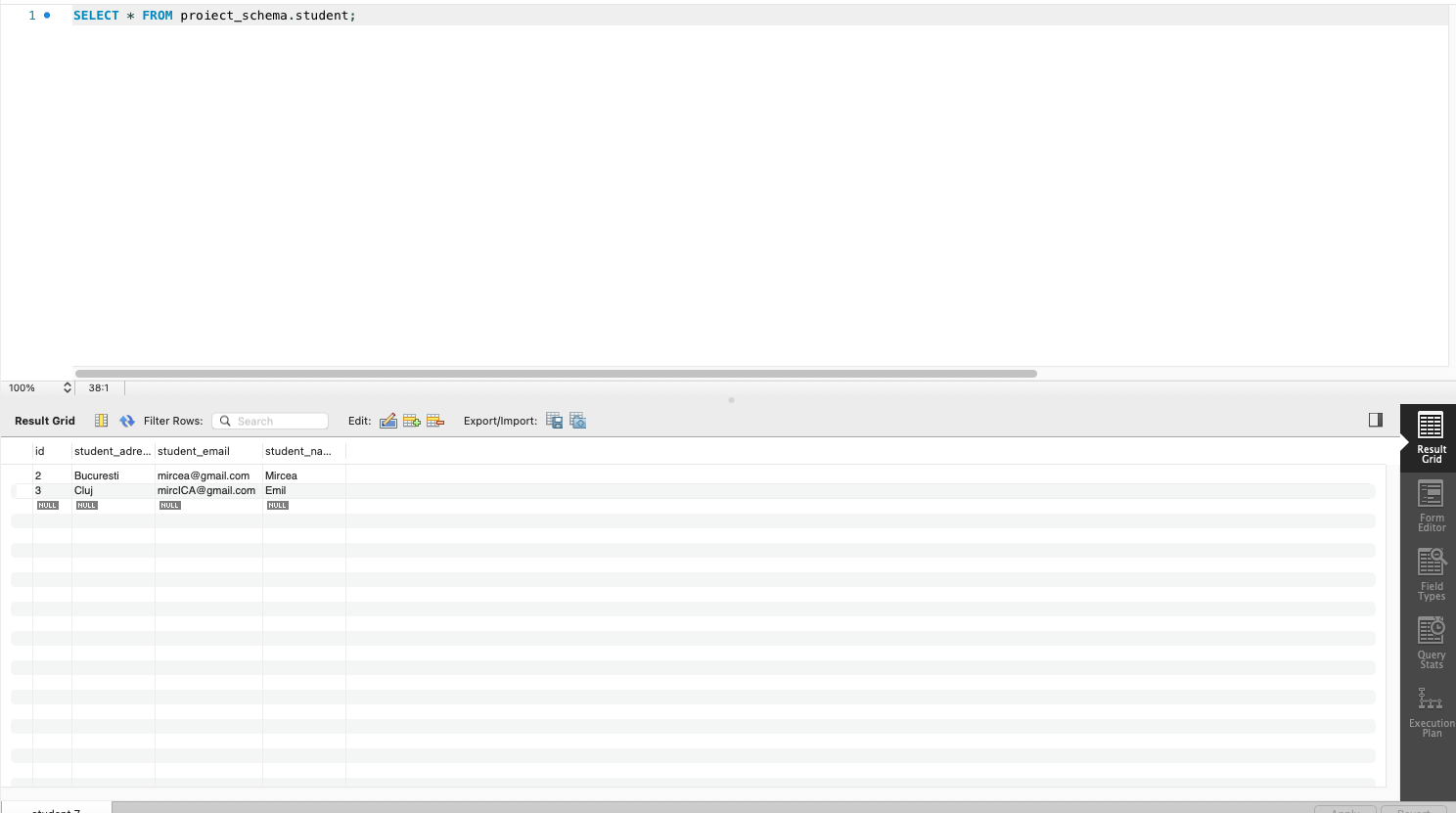
## POST

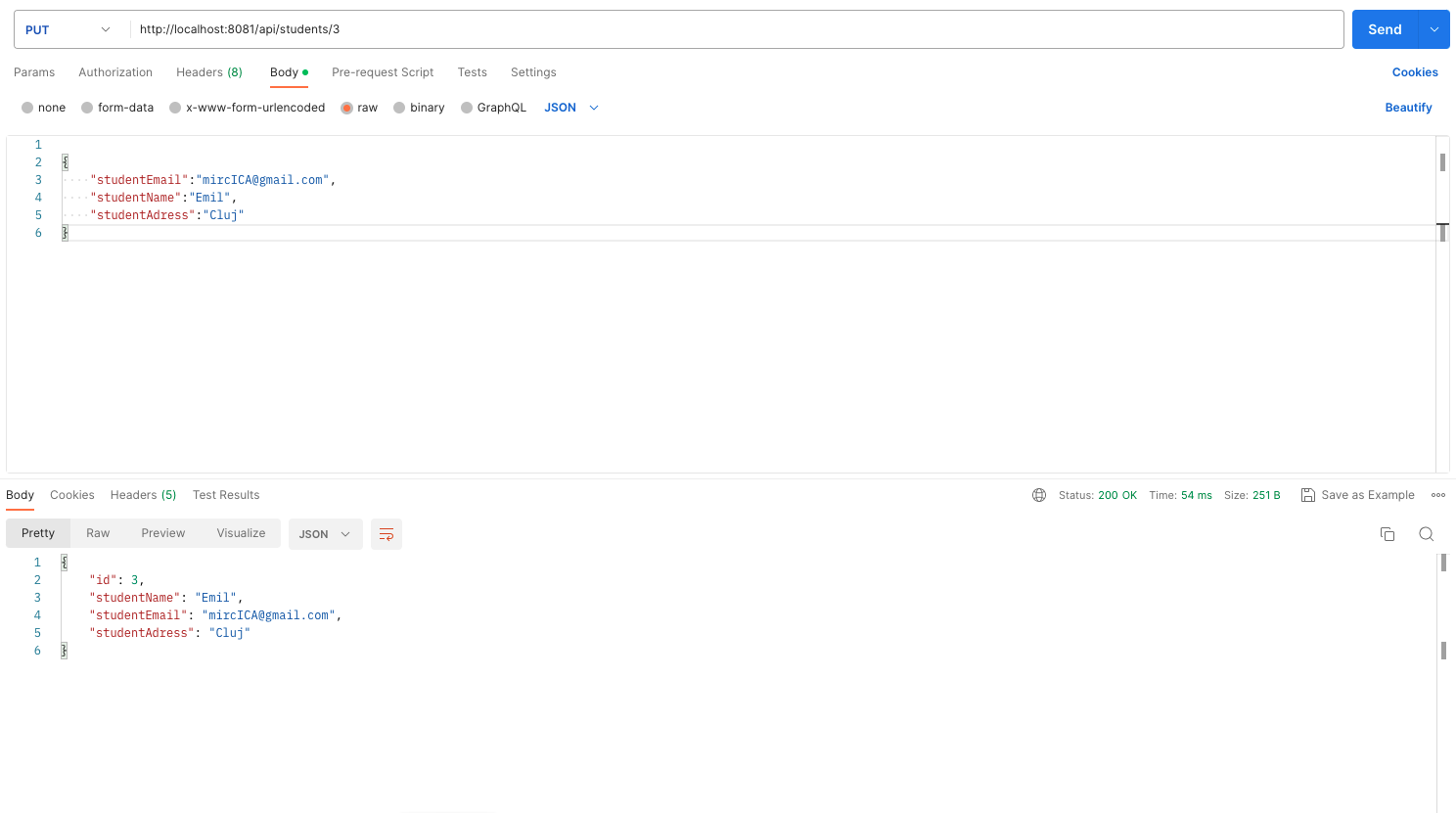


## GET



## PUT





## DELETE