**КУРСОВА РОБОТА**

з дисципліни

**«Конструювання программного забезпечення Java»**

на тему

**«Електронний журнал для вчителів»**

Опис:

**«Створення системи для ведення оцінок, відвідуваності та планування уроків, що дозволяє вчителям ефективно управляти навчальним процесом. »**

**Виконав**:

Студент групи ПД-33

Адамович Владислав Олегович

**Керівник курсової роботи:**

Викладач: Довженко Тимур Павлович

Київ – 2024

З огляду на виклики та нагальні потреби, які постали перед системою загальної середньої освіти України, з метою її дерегуляції та дебюрократизації, важливою потребою для спрощення ведення документообігу закладів загальної середньої освіти є впровадження електронного журналу для вчителів.

Електронний журнал є зручним засобом для відслідковування оцінок та відвідуваності учнів, планування уроків вчителями. Його використання допоможе вчителям ефективно виконувати свої обов'язки та забезпечить більшу прозорість та ефективність управління освітнім процесом.

Впровадження даного програмного продукту надасть для педагогічним працівникам, батькам та учням ЗЗСО (закладів загальної середньої освіти) зручний онлайн-інструмент, який забезпечить доступ до оцінок, відвідуваності та інших ресурсів журналу. Впровадження такого ІТ рішення є особливо актуальним для тих ЗЗСО та громад, які не можуть собі дозволити закупівлю альтернативних комерційних програмних продуктів.

Впровадження системи має наступні переваги:

* зручний доступ до інформації: за допомогою електронного журналу вчителі можуть легко отримувати доступ до інформації про своїх учнів;
* автоматизація обчислень: електронний журнал може автоматично розраховувати середні оцінки, відсоток відвідуваності, та інші показники успішності учнів. Це зменшує час, який витрачається на ручні обчислення та робить процес оцінювання більш точним і об'єктивним;
* можливість спільної роботи: електронний журнал дозволяє вчителям спільно працювати над плануванням уроків, веденням оцінок та аналізом даних. Це сприяє збільшенню співпраці та обміну ідеями між колегами;
* більша зручність для батьків: батьки можуть мати доступ до електронного журналу для відстеження успішності своїх дітей без необхідності відвідувати школу або чекати на регулярні зустрічі з вчителями. Це дозволяє батькам бути більш інформованими про академічний прогрес своїх дітей.

# **1 ПРОЕКТУВАННЯ ІНФОРМАЦІЙНОЇ СИСТЕМИ**

1.1 Опис User stories з діаграмами послідовностей

Проектування системи почнемо з опису User Stories. User story - це неформальне, загальне пояснення функції програмного забезпечення, написане з точки зору кінцевого користувача. Її мета - пояснити, як функція програмного забезпечення буде корисною для користувача. User Stories є одним з основних компонентів методу гнучкої розробки програмного забезпечення (Agile software development). Вони допомагають створити орієнтовану на користувача основу для щоденної роботи, що сприяє співпраці, творчості та покращенню якості продукту в цілому.

За Майком Коном історії користувачів складаються з трьох аспектів:

* письмовий опис історії, який використовується для планування;
* розмови про історію, які служать для уточнення деталей історії;
* тести, які документують деталі, які можна використовувати для визначення, коли історія завершена.

Історії користувачів часто виражаються простим реченням, структурованим наступним чином: “As a <user/user role> I want <capability> so that <benefits description>”.

Що я, як користувач, хочу чи маю отримати від системи (мається на увазі функціональність), щоб розв’язати свої проблеми/закрити власні потреби.

Роздивимось структуру детальніше:

"Як [користувач]": Для кого ми це будуємо? Нам потрібна не просто посада, нам потрібна особистість людини. Команда повинна мати спільне розуміння того, ким є користувач, для якого будується система. Для цього розуміння можна проводити інтерв’ю, щоб зрозуміти, як ця людина працює, як вона мислить і що відчуває.

"Хочу": Тут описуються наміри користувача, а не функції, які він використовує. Чого він насправді намагається досягти? Це твердження не повинно залежати від реалізації – наприклад, описуючи частину інтерфейсу, а не мету користувача, втрачається сенс user story.

"Щоб": як безпосереднє бажання користувача зробити щось вписується в загальну картину? Яку загальну вигоду він намагаються отримати? Яка велика проблема потребує вирішення?

Для зручності відобразимо кожну user story в окремій таблиці:

Таблиця 1

|  |  |
| --- | --- |
| User Story Title | Відслідковування відвідуваності на кожному уроці. |
| User Story Statement | Як вчитель, я хочу мати можливість відмічати відвідуваність кожного учня на кожному уроці, щоб відстежувати їх прогрес і вчасно реагувати на відсутність. |
| Acceptance Criteria | У системі має бути доступний список всіх учнів, які мають відвідувати урок.  Для кожного уроку має бути можливість відмітити присутність або відсутність кожного учня.  Система повинна зберігати історію відвідуваності кожного учня, щоб вчитель міг переглянути динаміку присутності протягом семестру або навчального року.  Вчителю має бути надана можливість генерувати звіти про відвідуваність, які містять інформацію про пропущені уроки кожного учня. |

Таблиця 2

|  |  |
| --- | --- |
| User Story Title | Виставлення оцінок учням. |
| User Story Statement | Як вчитель, я хочу мати можливість виставляти оцінки, обираючи необхідний предмет і учня, щоб записувати оцінки за домашнє завдання, тести, контрольні роботи. |
| Acceptance Criteria | У системі має бути можливість створення окремого електронного журналу для кожного предмету, що веде вчитель.  Для кожного уроку або заняття має бути можливість введення оцінок за різні види оцінювання, такі як завдання, тести, контрольні роботи тощо.  Система повинна забезпечувати можливість введення числових оцінок.  Вчителю має бути доступний перегляд електронного журналу з оцінками кожного учня для кожного предмету. |

Таблиця 3

|  |  |
| --- | --- |
| User Story Title | Планування уроків. |
| User Story Statement | Як вчитель, я хочу мати можливість планувати уроки заздалегідь, додаючи матеріали, завдання та інші ресурси до кожного уроку у відповідний розділ. |
| Acceptance Criteria | У системі має бути доступний розділ для планування уроків для кожного предмету, що веде вчитель.  Вчителю має бути можливість створити новий план уроку. До кожного плану уроку має бути можливість додавати матеріали, такі як презентації, відео, документи тощо, а також завдання.  В системі має бути доступний ряд різних типів завдань, таких як тестові завдання, домашні завдання, проекти, практичні роботи тощо.  Вчителю має бути надана можливість вибору шкали оцінювання для кожного типу завдання, такої як числова шкала (від 1 до 12).  Система повинна забезпечувати можливість перегляду планів уроків на майбутні дати та редагування їх у будь-який момент.  Система повинна забезпечувати можливість спільного доступу до планів уроків для інших вчителів та адміністраторів школи, якщо це необхідно для спільної роботи та координації. |

Таблиця 4

|  |  |
| --- | --- |
| User Story Title | Доступ до електронного журналу класу свого класу. |
| User Story Statement | Як вчитель, я хочу мати можливість мати доступ до електронного журналу свого класу по всім предметам, щоб зручно вести оцінювання та відстежувати прогрес усіх учнів. |
| Acceptance Criteria | В системі має бути доступний окремий розділ для класу, класним керівником якого є вчитель, із можливістю перегляду оцінок та відвідуваності усіх учнів цього класу по всім предметам.  Вчителю має бути надана можливість швидкого переходу між різними предметами без необхідності виходу з системи та повторного входу.  Система повинна забезпечувати можливість фільтрації та сортування оцінок за різними критеріями.  Вчителю має бути доступний інтерактивний режим перегляду, який дозволяє швидко переглядати детальну інформацію про оцінки та відвідуваність кожного учня. |

Таблиця 5

|  |  |
| --- | --- |
| User Story Title | Генерація звітів про успішність учнів. |
| User Story Statement | Як вчитель, я хочу мати можливість генерувати звіти про успішність учнів, щоб мати зручний засіб для комунікації з батьками та адміністрацією школи. |
| Acceptance Criteria | Звіти повинні містити інформацію про оцінки кожного учня за вказаний період, включаючи середні оцінки, результати окремих завдань та інші показники успішності.  Звіти повинні бути зрозумілими та легкими для використання, навіть для тих, хто не має спеціалізованого навчання у використанні комп'ютерних програм. |

## **1.2 Діаграми послідовностей для user stories**

UML (Unified Modeling Language) – уніфікована мова моделювання – це система позначень, що застосовується для об'єктно-орієнтованого аналізу та проектування, це мова діаграм або позначень для специфікації, візуалізації та документації моделі об'єктно-орієнтованих програмних систем. UML не є методом розробки, тобто він не визначає послідовність дій при розробці ПЗ. Він допомагає описати свою ідею і взаємодіяти з іншими розробниками системи. UML управляється Object Management Group (OMG) і є промисловим стандартом, що описує моделі ПЗ.

UML визначає різні види діаграм, серед них: д

* діаграма прецедентів;
* діаграма класів;
* діаграма послідовностей;
* діаграма активності;
* діаграма взаємодії;
* діаграма станів;
* діаграма розгортання.

Діаграма послідовності (sequence diagram) — різновид діаграми в UML. Діаграма послідовності відображає взаємодії об'єктів впорядкованих за часом. Зокрема, такі діаграми відображають задіяні об'єкти та послідовність надісланих повідомлень.

На діаграмі послідовності паралельними вертикальними лініями ("лініями життя" - lifelines) показано різні процеси або об'єкти, які існують одночасно, а горизонтальними стрілками - повідомлення, якими вони обмінюються між собою, у порядку їх виникнення. Це дозволяє специфікувати прості сценарії виконання у графічній формі.

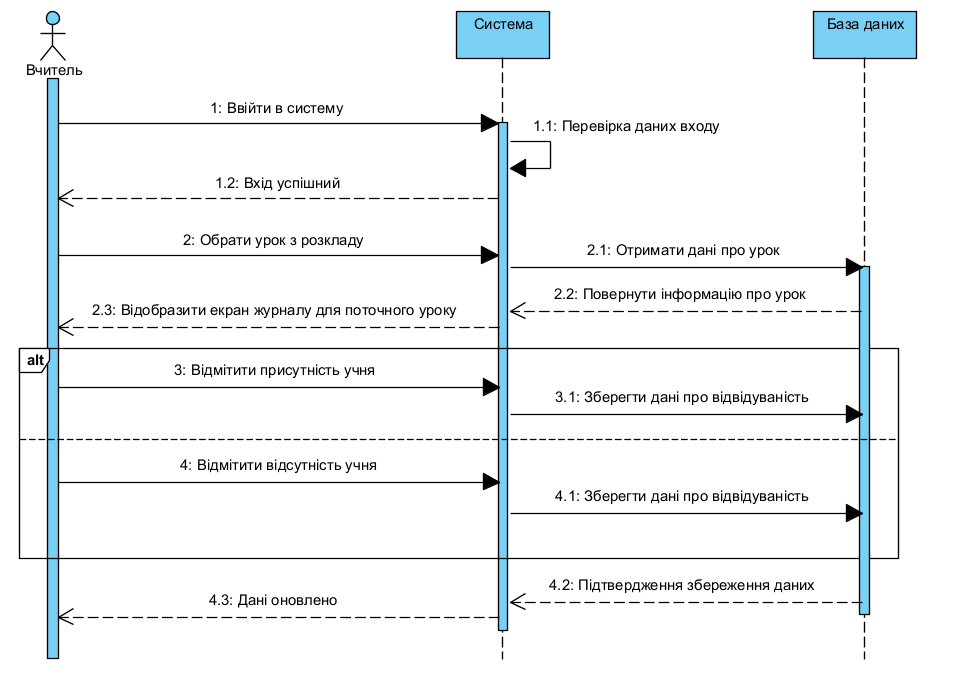


Рисунок 1.1 – Діаграма послідовності для відслідковування відвідуваності на уроці

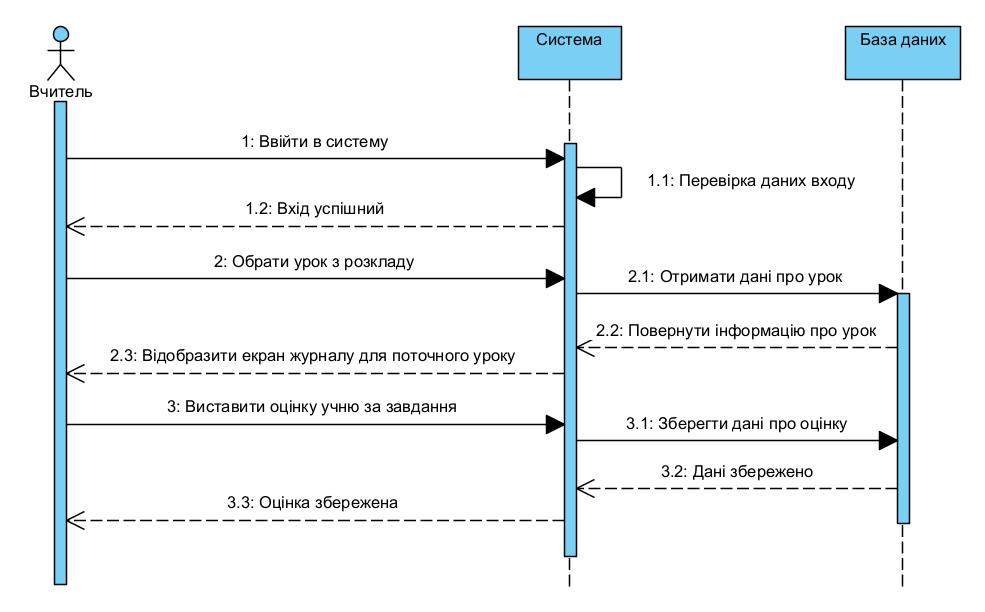


Рисунок 1.2 – Діаграма послідовності для оцінювання учнів

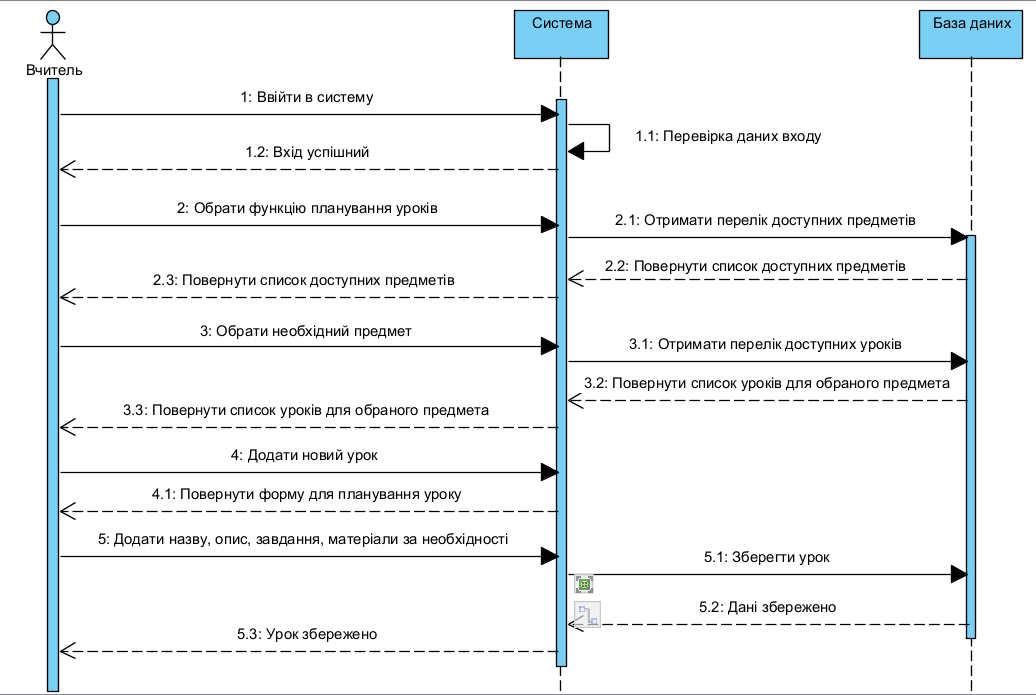


Рисунок 1.3 – Діаграма послідовності для планування уроків

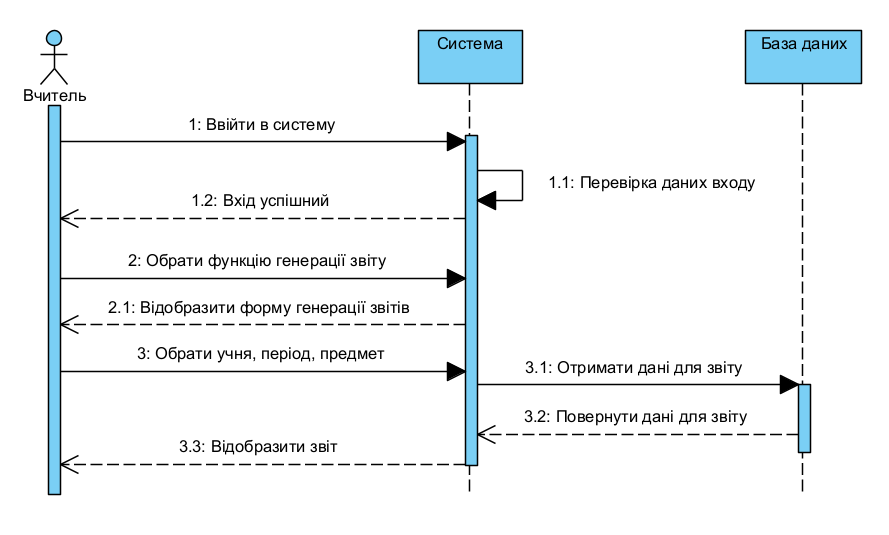


Рисунок 4 – Діаграма послідовності длягенерації звітів про успішність учнів.

1.3 Опис ERD бази даних

Далі розробимо ER-модель – це модель даних, яка дозволяє описувати концептуальні схеми за допомогою узагальнених конструкцій блоків. Модель сутність-зв'язок є результатом систематичного процесу, який описує та визначає деяку предметну область. Вона не визначає сам процес, а лише візуалізує його. Дані представлені у вигляді компонентів (сутностей), які пов'язані між собою певними зв'язками, які виражають залежності і вимоги між ними, такі як: одна будівля може бути розділена на декілька квартир, але одна квартира може бути розташована лише в одній будівлі. Сутності можуть мати різні властивості (атрибути), які характеризують їх. Діаграми, створені для представлення цих сутностей, атрибутів і зв'язків графічно, називають сутність-зв'язок діаграмами.

Почнемо з опису сутностей системи.

* + 1. **Сутність Вчитель**

**Короткий опис сутності.** Представляє інформацію про кожного вчителя.

**Атрибути.** Сутність характеризується наступними атрибутами:

* ідентифікатор;
* ім’я;
* прізвище;
* електронна пошта;
* пароль.

**Зв’язки.** Сутність Вчитель маєнаступні зв’язки з іншими сутностями:

* вчитель, як класний керівник, може вести один клас;
* вчитель може вести багато предметів;
* вчитель може проводити багато уроків.

**Бізнес-правила.** Відносно сутностідіють наступні бізнес-правила:

* ідентифікатор вчителя унікально ідентифікує його, оскільки не може бути більше одного вчителя з однаковим ідентифікатором.
* усі атрибути є обов’язковими.
  + 1. **Сутність Учень**

**Короткий опис сутності.** Представляє інформацію про кожного учня.

**Атрибути.** Сутність характеризується наступними атрибутами:

* ім'я;
* прізвище;
* батьківські контактні дані (ім’я, номер телефону, адреса електронної пошти).
* клас.

**Зв’язки.** Сутність маєнаступні зв’язки з іншими сутностями:

* кожен учень належить до одного конкретного класу;
* учень може мати багато оцінок;
* учень має багато відвідувань.

**Бізнес-правила.** Відносно сутностідіють наступні бізнес-правила:

* ідентифікатор учня унікально ідентифікує його, оскільки не може бути більше одного учня з однаковим ідентифікатором.
* усі атрибути, окрім електронної адреси, є обов’язковими.
  + 1. **Сутність Предмет**

**Короткий опис сутності.** Представляє інформацію про кожний навчальний предмет.

**Атрибути.** Сутність характеризується наступними атрибутами:

* ідентифікатор;
* назва;
* вчитель.

**Зв’язки.** Сутність маєнаступні зв’язки з іншими сутностями:

* предмет може вестися багатьма вчителями;
* предмет містить в собі багато планів уроків.

**Бізнес-правила.** Відносно сутностідіють наступні бізнес-правила:

* ідентифікатор предмету унікально ідентифікує його.
* усі атрибути, є обов’язковими.
  + 1. **Сутність Урок**

**Короткий опис сутності.** Представляє інформацію про урок.

**Атрибути.** Сутність характеризується наступними атрибутами:

* ідентифікатор;
* ідентифікатор вчителя;
* ідентифікатор класу, для якого проводиться урок;
* ідентифікатор плану уроку.

**Зв’язки.** Сутність маєнаступні зв’язки з іншими сутностями:

* урок може проводитися в багатьох класах;
* урок проводиться одним вчителем;
* урок має один план уроку;
* урок має багато оцінок;
* урок має багато відвідувань.

**Бізнес-правила.** Відносно сутностідіють наступні бізнес-правила:

* ідентифікатор уроку має бути унікальним для кожного року навчання;
* всі атрибути є обов’язковими.
  + 1. **Сутність Клас**

**Короткий опис сутності.** Представляє інформацію про клас, до якого належать учні.

**Атрибути.** Сутність характеризується наступними атрибутами:

* ідентифікатор;
* назва;
* початок навчання (рік);
* кінець навчання (рік);
* ідентифікатор вчителя (класний керівник).

**Зв’язки.** Сутність маєнаступні зв’язки з іншими сутностями:

* клас може мати багато учнів;
* клас може мати багато уроків.

**Бізнес-правила.** Відносно сутностідіють наступні бізнес-правила:

* номер класу має бути унікальним для кожного року навчання;
* всі атрибути є обов’язковими.
  + 1. **Сутність Оцінка**

**Короткий опис сутності.** Представляє інформацію про оцінки, які отримали учні за різними завданнями.

**Атрибути.** Сутність характеризується наступними атрибутами:

* оцінка;
* дата отримання;
* ідентифікатор учня;
* ідентифікатор уроку.

**Зв’язки.** Сутність маєнаступні зв’язки з іншими сутностями:

* кожна оцінка належить до одного конкретного учня;
* кілька оцінок можуть відноситися до одного конкретного уроку.

**Бізнес-правила.** Відносно сутностідіють наступні бізнес-правила:

* оцінка повинна бути в межах певного діапазону, який визначений шкалою оцінювання;
* дата виставлення оцінки повинна бути коректною та відповідати формату;
* всі атрибути є обов’язковими.
  + 1. **Сутність Відвідуваність**

**Короткий опис сутності.** Представляє інформацію про відвідуваність учнів на уроках.

**Атрибути.** Сутність характеризується наступними атрибутами:

* ідентифікатор студента;
* дата;
* статус відвідуваності (присутній, відсутній);
* ідентифікатор уроку.

**Зв’язки.** Сутність маєнаступні зв’язки з іншими сутностями:

* кожен запис про відвідуваність належить до одного конкретного учня.
* кожен запис про відвідуваність відноситься до одного конкретного уроку.

**Бізнес-правила.** Відносно сутностідіють наступні бізнес-правила:

* всі атрибути є обов’язковими.
  + 1. **Сутність План уроку**

**Короткий опис сутності.** Представляє інформацію про кожний план уроку.

**Атрибути.** Сутність характеризується наступними атрибутами:

* ідентифікатор;
* назва;
* опис;
* предмет.

**Зв’язки.** Сутність маєнаступні зв’язки з іншими сутностями:

* кожен план уроку може містити декілька завдань;
* кожен план уроку може містити декілька матеріалів;
* кожен план уроку може бути використовуватися багатьма уроками.

**Бізнес-правила.** Відносно сутностідіють наступні бізнес-правила:

* атрибути ідентифікатор, назва, предмет є обов’язковими.
  + 1. **Сутність Завдання**

**Короткий опис сутності.** Представляє інформацію про різні завдання для учнів.

**Атрибути.** Сутність характеризується наступними атрибутами:

* ідентифікатор завдання;
* назва;
* тип завдання;
* максимальна оцінка.

**Зв’язки.** Сутність маєнаступні зв’язки з іншими сутностями:

* завдання можуть відноситися до кількох планів уроків.

**Бізнес-правила.** Відносно сутностідіють наступні бізнес-правила:

* ідентифікатор завдання має бути унікальним;
* всі атрибути є обов’язковими.
  + 1. **Сутність Матеріали**

**Короткий опис сутності.** Представляє інформацію про матеріали та ресурси, які використовуються на уроках.

**Атрибути.** Сутність характеризується наступними атрибутами:

* ідентифікатор;
* назва;
* опис;
* тип матеріалу;
* посилання на файл.

**Зв’язки.** Сутність маєнаступні зв’язки з іншими сутностями:

* матеріал може бути використаний у багатьох різних планах уроків.

**Бізнес-правила.** Відносно сутностідіють наступні бізнес-правила:

* ідентифікатор матеріалу має бути унікальною для ідентифікації та посилання на нього.
* опис матеріалу може бути необов'язковим.
* всі інші атрибути є обов’язковими.

Зобразимо всі вище перераховані сутності за допомогою діаграми:

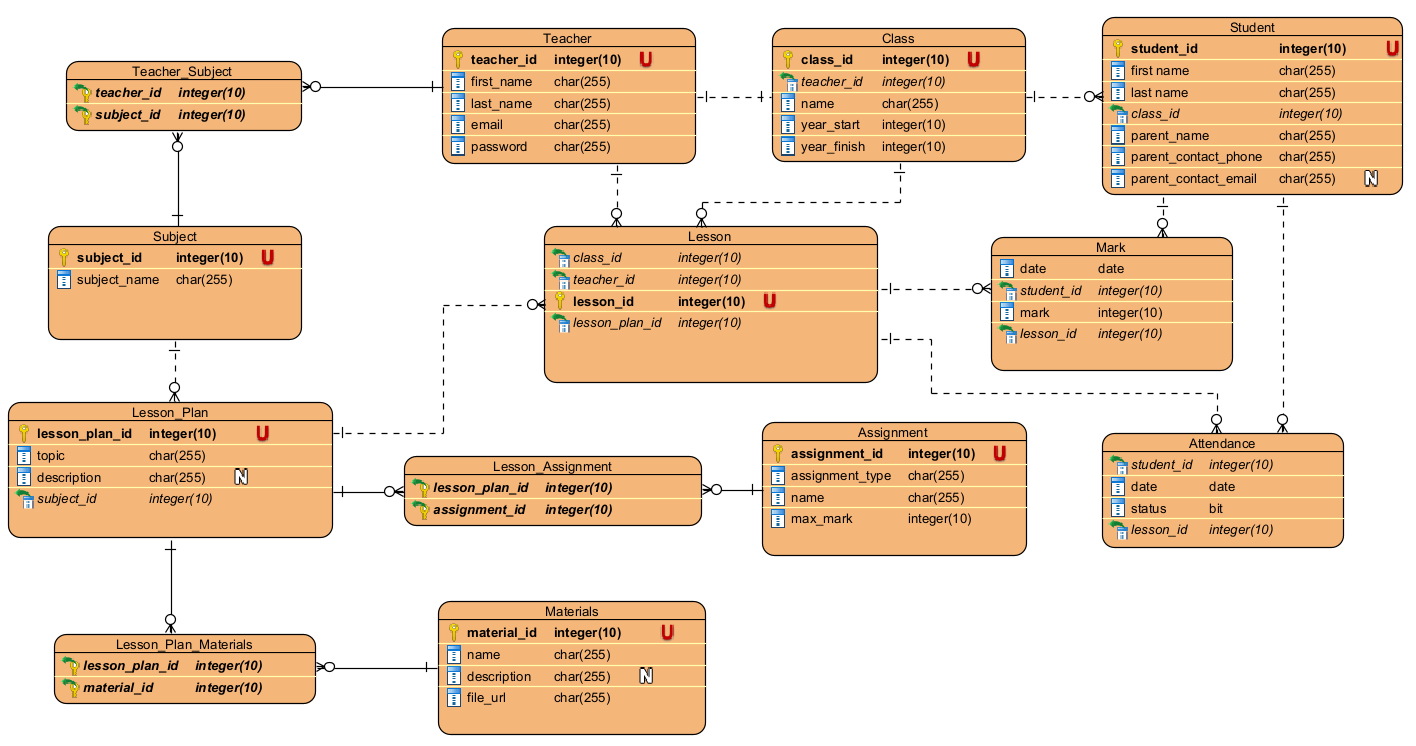


Рисунок 1.5 – ER-діаграма

1.4 Опис діаграми класів

Діаграма класів – це діаграма, на якій показані класи, інтерфейси та відносини між ними. Головний елемент діаграми класів – клас. При проектуванні об'єктно-орієнтованих систем діаграми класів обов'язкові. Класи використовуються в процесі аналізу предметної області для складання словника предметної області системи, що розробляється. Це можуть бути як абстрактні поняття предметної області, так і класи, на які спирається розробка та які описують програмні або апаратні сутності. Діаграма класів є набором статичних, декларативних елементів моделі. Класи зображуються у вигляді прямокутників, зазвичай розділених на дві або три частини. У верхній частині знаходиться ім'я класу. Середня частина містить список змінних класу, а нижня частина – методи класу. Символи, зазначені перед кожною змінною або методом, представляють собою індикатори видимості.

Виділимо класів в системі. Опишемо кожний з них.

**Вчитель (Teacher)**

Атрибути:

* teacher\_id: Int – унікальний ідентифікатор вчителя;
* name: String – ім’я
* email: String – електронна адреса, використовується для входу в систему;
* password: String – пароль, використовується для входу в систему;
* subjects: List<Subjects> – список предметів, які викладає вчитель;
* class: Class – клас, класним керівником якого є вчитель.

Методи:

* viewScheduledLessons() : List<Lesson> – переглянути уроки вчителя
* markAttendance(student : Student, attendance : Attendance) – відмітити відсутність або присутність учнів
* markStudent(student : Student, mark : Mark) – поставити оцінку учню
* planLesson() – спланувати урок
* generateReports() – створити звіти по успішності/відвідуваності

**Клас (Class)**

Атрибути:

* class\_id: Int – унікальний ідентифікатор класу;
* name: String – назва класу;
* startYear: Int – початок навчання;
* finishYear: Int – кінець навчання;
* studentList: List<Student> – список учнів, що належать класу;
* responsibleTeacher: Teacher – класний керівник.

Методи:

* getStudentList() : List<Student> – отримати список учнів
* getClassAttendance() : List<Attendance> – отримати відвідуваність класу
* getClassMarks() : List<Mark> – отримати оцінки учнів

**Учень (Student)**

Атрибути:

* student\_id : Int – унікальний ідентифікатор класу;
* name : String – ім’я учня;
* class : Class – клас, до якого належить учень;
* parentName : String – ім’я когось із батьків;
* parentPhone : String – номер телефону когось із батьків;
* parentEmail : String – електронна адреса когось із батьків.

Методи:

* getParentInformation(): String – отримати інформацію про батьків:
* getStudentAttendance() : List<Attendance> – отримати інформацію про відвідуваність учня:
* getStudentMarks() : List<Mark> – отримати інформацію про оцінки учня

**Предмет (Subject)**

Атрибути:

* subject\_id : Int – унікальний ідентифікатор предмету;
* name : String – назва предмету;
* lessonPlans : List<Lesson Plan> – перелік планів уроків для предмету.

Методи:

* getSubject() : Subject – отримати інформацію про предмет;
* getLessonPlans() : List<Lesson Plan> – отримати список планів уроків предмета;
* addLessonPlan() – додати новий план уроку;
* editLessonPlan(lessonPlan : Lesson Plan) – редагувати план уроку;
* deleteLessonplan(lessonPlan : Lesson Plan) – видалити план уроку.

**Урок (Lesson)**

Атрибути:

* lesson\_id: Int – унікальний ідентифікатор уроку;
* teacher: Teacher – вчитель, що провів урок;
* class : Class – клас, у якого був урок;
* lessonPlan : Lesson Plan – план уроку;
* date : Date – дата уроку.

Методи:

* viewLessonInformation() – отримати інформацію про урок

**Оцінка (Mark)**

Атрибути:

* value: Int – значення оцінки;
* date : Date – дата отримання оцінки;
* student : Student – учень, що отримав оцінку;
* assignent : Assignment – завдання, за яке учень отримав оцінку.

Методи:

* getMark() : Mark – отримати інформацію про оцінку.

**Відвідуваність (Attendance)**

Атрибути:

* status : Boolean – присутній/відсутній;
* date : Date – дата;
* student : Student – учень;
* lesson: Lesson – урок.

Методи:

* get Attendance() : Attendance – отримати інформацію про відвідуваність

**План уроку (Lesson plan)**

Атрибути:

* lesson\_plan\_id : Int – унікальний ідентифікатор плану уроку;
* topic : String – тема уроку;
* description : String – опис;
* materials : List<Material> – матеріали;
* assignments : List<Assignment> – завдання.

Методи:

* getMarks() : List<Mark> – отримати інформацію про батьків

**Завдання (Assignment)**

Атрибути:

* assignment\_id : Int – унікальний ідентифікатор завдання;
* type : String – тип завдання;
* maxMark : Int – максимальна оцінка за завдання.

Методи:

* getLessonPlan() : LessonPlan – отримати інформацію про план уроку;
* addNewMaterial() – додати матеріал до плану уроку;
* editMaterial(material : Material) – редагувати матеріал до плану уроку;
* deleteMaterial(material : Material) – видалити матеріал з плану уроку;
* addNewAssignment() – додати завдання до плану уроку;
* editAssignment(assignment : Assignment) – редагувати завдання до плану уроку;
* deleteAssignment(assignment : Assignment) – видалити завдання з плану уроку.

**Матеріал (Material)**

Атрибути:

* material\_id : Int – унікальний ідентифікатор матеріалу;
* name : String – назва матеріалу;
* description : String – опис матеріалу;
* file\_url : String – шлях до файлу.

Методи:

* getMaterial() : Material – отримати інформацію про матеріал.

Діаграма класів зображена на рисунку 1. 6.

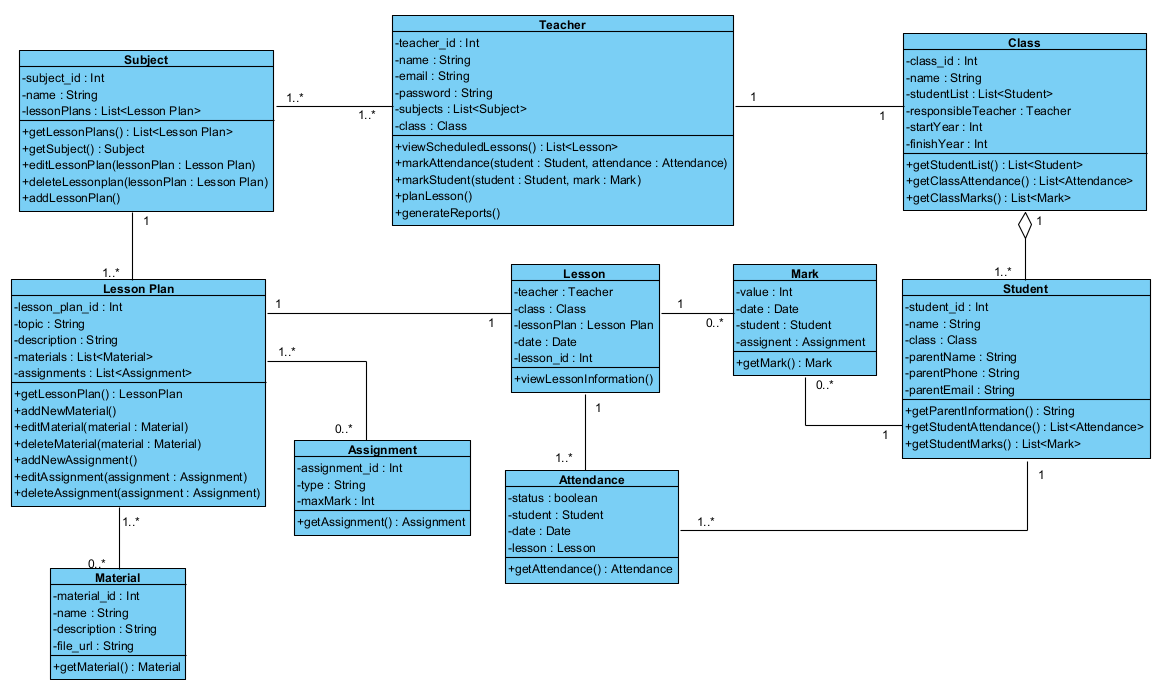


Рисунок 1.6 – Діаграма класів

1.5 Написання тестових сценаріїв

1. Перевірка входу в систему:

Крок 1: Введення електронної адреси і паролю.

Очікуваний результат: Вхід в систему успішний.

1. Перевірка входу в систему з неправильними даними:

Крок 1: Введення неправильної електронної адреси або паролю.

Очікуваний результат: Система коректно обробляє неправильні облікові дані та забезпечує обмеження доступу до конфіденційної інформації.

1. Перевірка можливості перегляду електронного журналу для класу, де вчитель є класним керівником:

Крок 1: Увійти в систему як учитель.

Крок 2: Обрати вкладку зі своїм класом.

Крок 3: Обрати предмет.

Крок 4: Переглянути електронний журнал.

Очікуваний результат: Учителю відображається список учнів з їхніми оцінками та відмітками відвідуваності для обраного предмету.

1. Перевірка можливості перегляду батьківських даних для класу, де вчитель є класним керівником:

Крок 1: Увійти в систему як учитель.

Крок 2: Обрати вкладку зі своїм класом.

Крок 3: Обрати учня.

Крок 4: Переглянути батьківські дані.

Очікуваний результат: Учителю відображається інформація про батьків учня, що містить в собі ім’я, контактні дані (електронна адреса та номер телефону).

1. Перевірка можливості перегляду електронного журналу для конкретного уроку вчителем:

Крок 1: Увійти в систему як учитель.

Крок 2: Обрати потрібний урок.

Крок 3: Переглянути електронний журнал.

Очікуваний результат: Учителю відображається список учнів з їхніми оцінками та відмітками відвідуваності для обраного уроку.

1. Перевірка можливості створення учителем нового запису в електронному журналі для конкретного уроку:

Крок 1: Увійти в систему як учитель.

Крок 2: Обрати потрібний урок.

Крок 3: Створити новий запис з оцінкою або відміткою відвідуваності.

Очікуваний результат: Запис успішно додано до електронного журналу.

1. Перевірка можливості редагування запису в електронному журналі:

Крок 1: Увійти в систему як учитель.

Крок 2: Обрати потрібний урок або предмет.

Крок 3: Знайти запис, який потрібно відредагувати.

Крок 4: Відредагувати оцінку або відмітку відвідуваності.

Очікуваний результат: Запис успішно відредаговано та збережено.

1. Перевірка можливості видалення запису з електронного журналу:

Крок 1: Увійти в систему як учитель.

Крок 2: Обрати потрібний урок або предмет.

Крок 3: Знайти запис, який потрібно видалити.

Крок 4: Видалити оцінку або відмітку відвідуваності.

Очікуваний результат: Запис успішно видалено.

1. Перевірка можливості генерації звіту про успішність учнів:

Крок 1: Увійти в систему як учитель.

Крок 2: Обрати функцію "Генерація звіту".

Крок 3: Вибрати параметри звіту (період, клас, предмет тощо).

Крок 4: Згенерувати звіт.

Очікуваний результат: Звіт про успішність учнів відображається коректно згідно вибраних параметрів.

# **2 РЕАЛІЗАЦІЯ API**

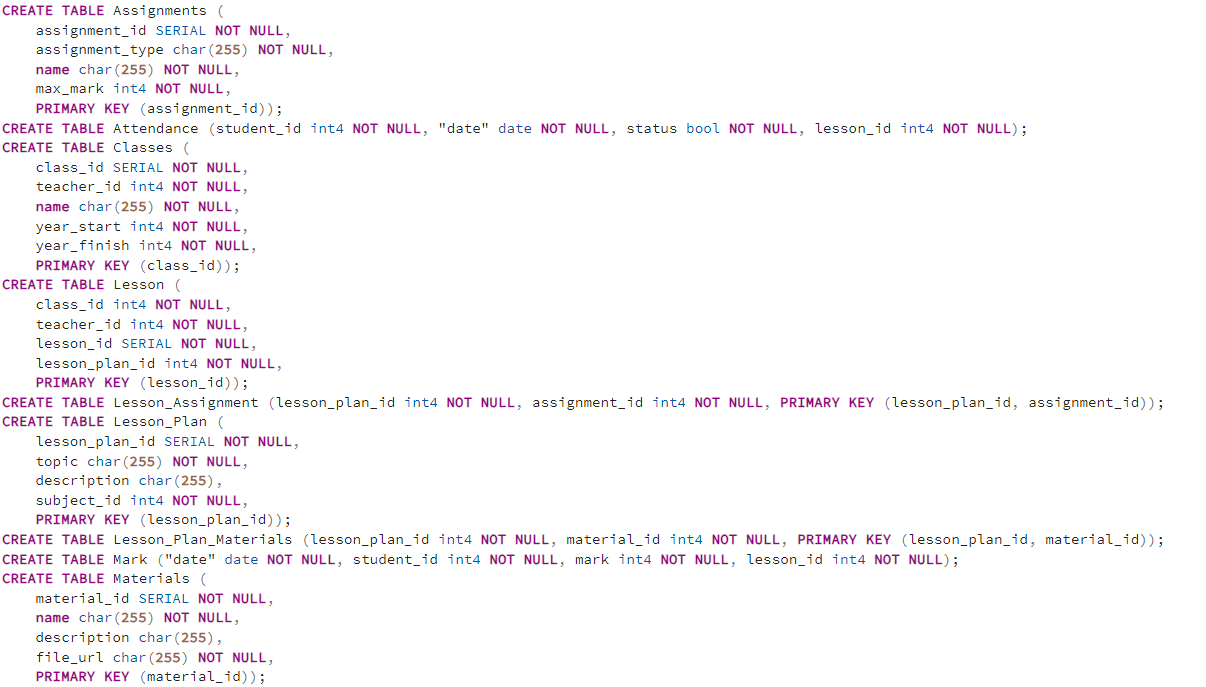
**2.1 Підготовка бази даних**

Оскільки схема бази даних спроектована у попередньому розділі, перейдемо до вибору СКБД. У проєкті будемо використовувати PostgreSQL. PostgreSQL (скорочено Postgres) - це вільна та відкрита реляційна система керування базами даних, яка цінується за свою надійність, гнучкість та масштабованість. Вона використовується широким колом організацій, від стартапів до великих підприємств, для зберігання та аналізу даних.

Ось деякі з ключових характеристик PostgreSQL:

* відкритий код: вільна та відкрита платформа, що дає користувачам свободу використовувати, вивчати та модифікувати її код без будь-яких обмежень;
* надійність: стабільність та стійкість до збоїв, що робить її надійним вибором для критично важливих застосунків;
* гнучкість: широкий спектр функцій та можливостей робить PostgreSQL придатною для вирішення різних завдань, пов'язаних з базами даних.
* масштабованість: можливість вертикального та горизонтального масштабування;
* сумісність із SQL: підтримка стандарту мови запитів SQL;
* безпека: широкий спектр функцій безпеки для захисту даних від несанкціонованого доступу та модифікації.

Перейдемо до створення бази даних.



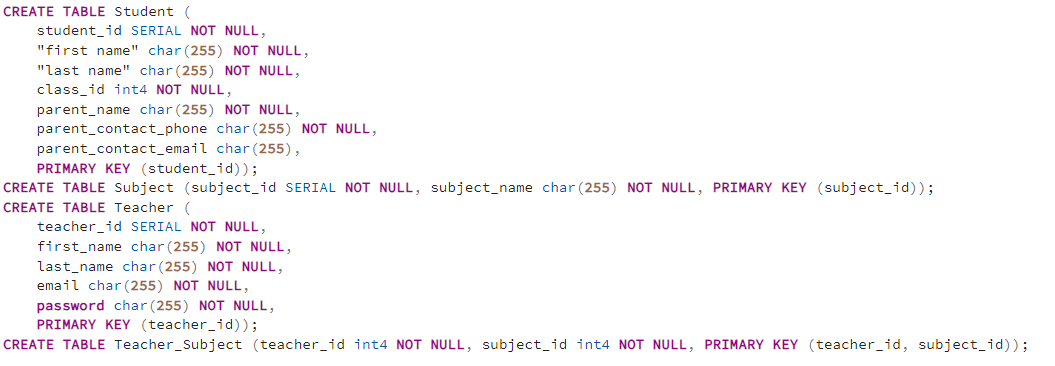




Рисунок 2.1 – Створення таблиць БД

Після цього заповнимо створені таблиці даними. Для цього частково згенеруємо дані за допомогою сервісу [Mockaroo](https://www.mockaroo.com/).

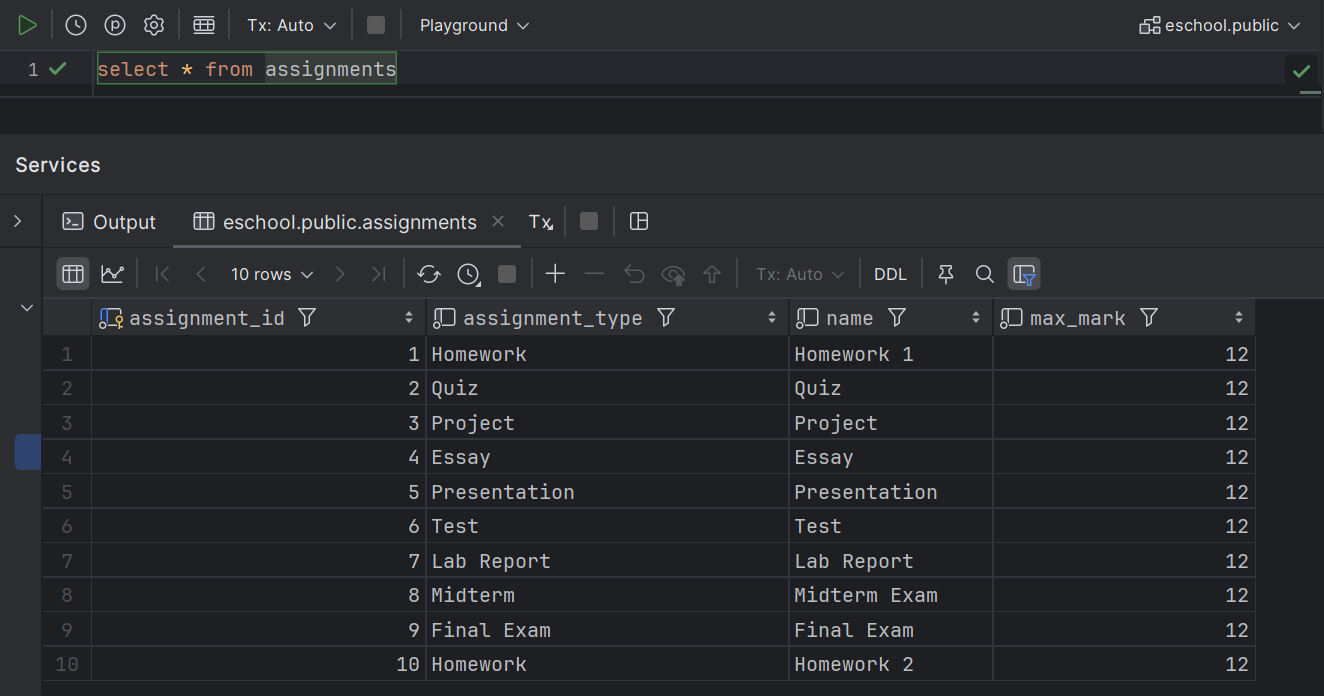


Рисунок 2.2 – Заповнена даними таблиця Assigments

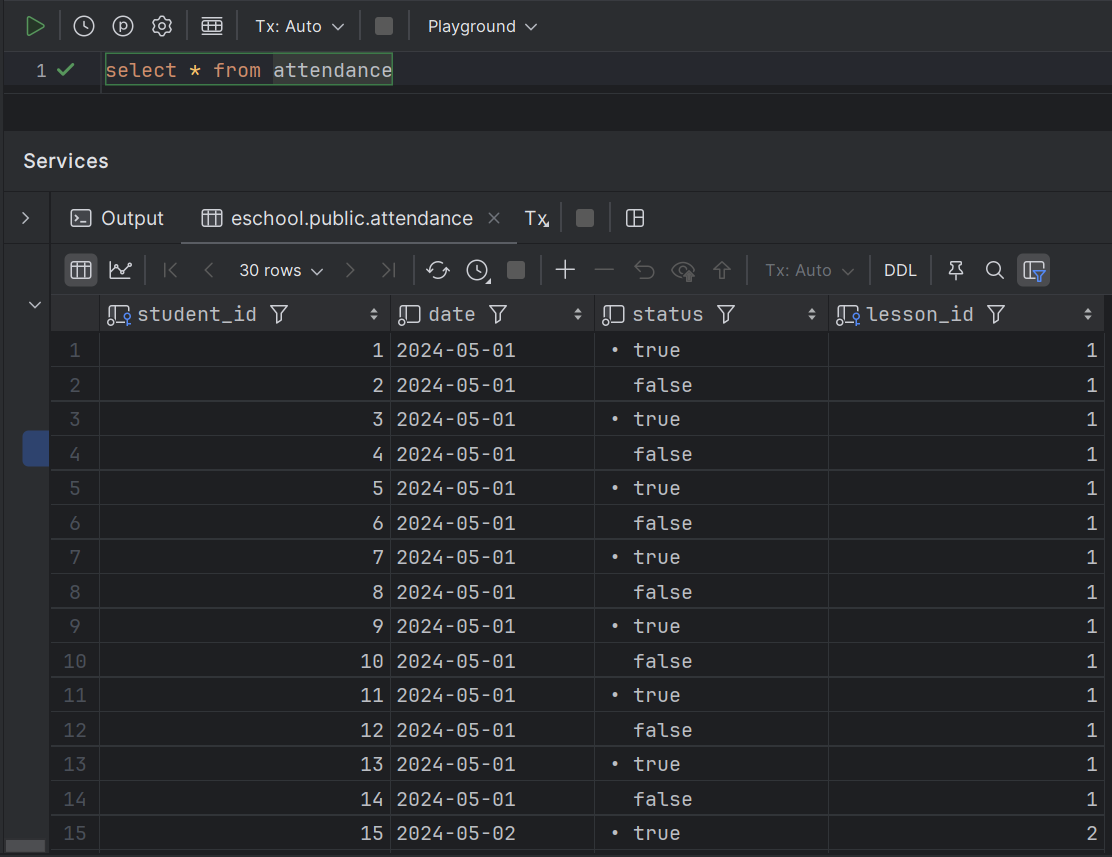


Рисунок 2.3 – Заповнена даними таблиця Attendance

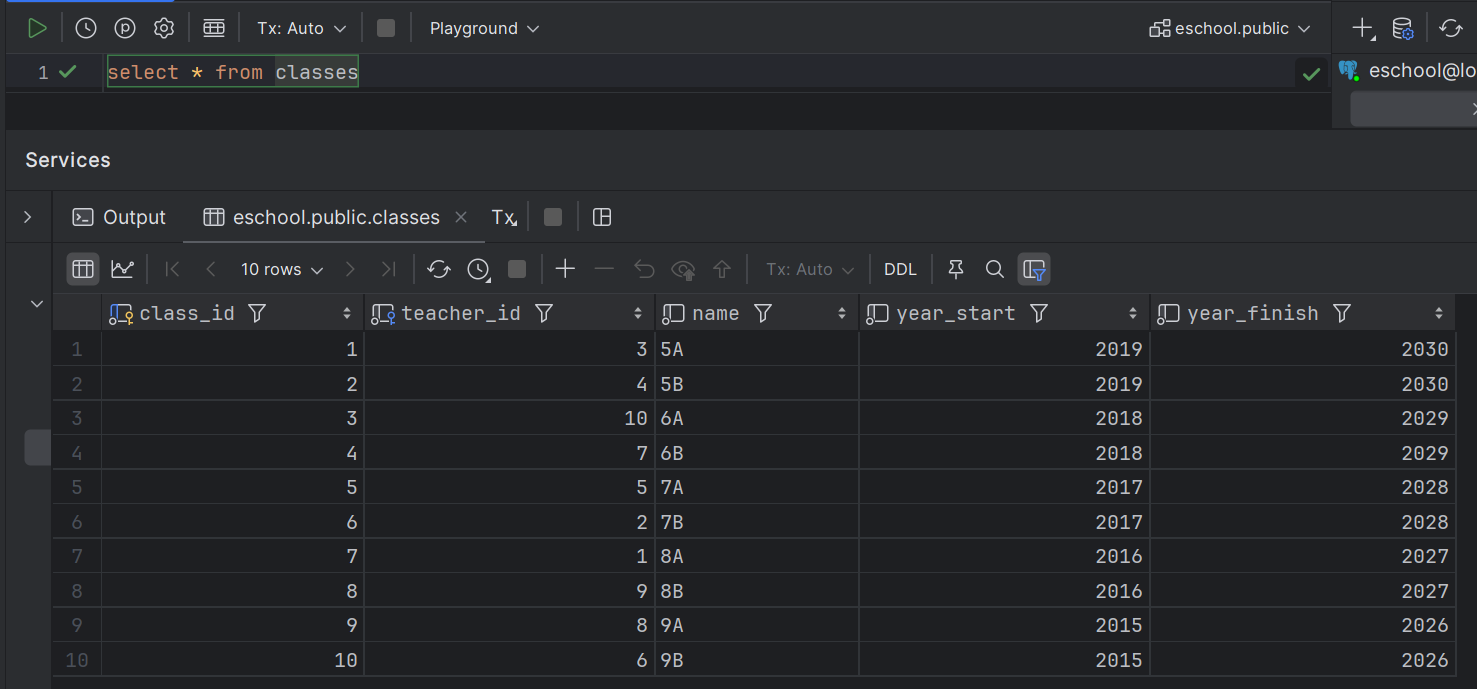


Рисунок 2.4 – Заповнена даними таблиця Classes

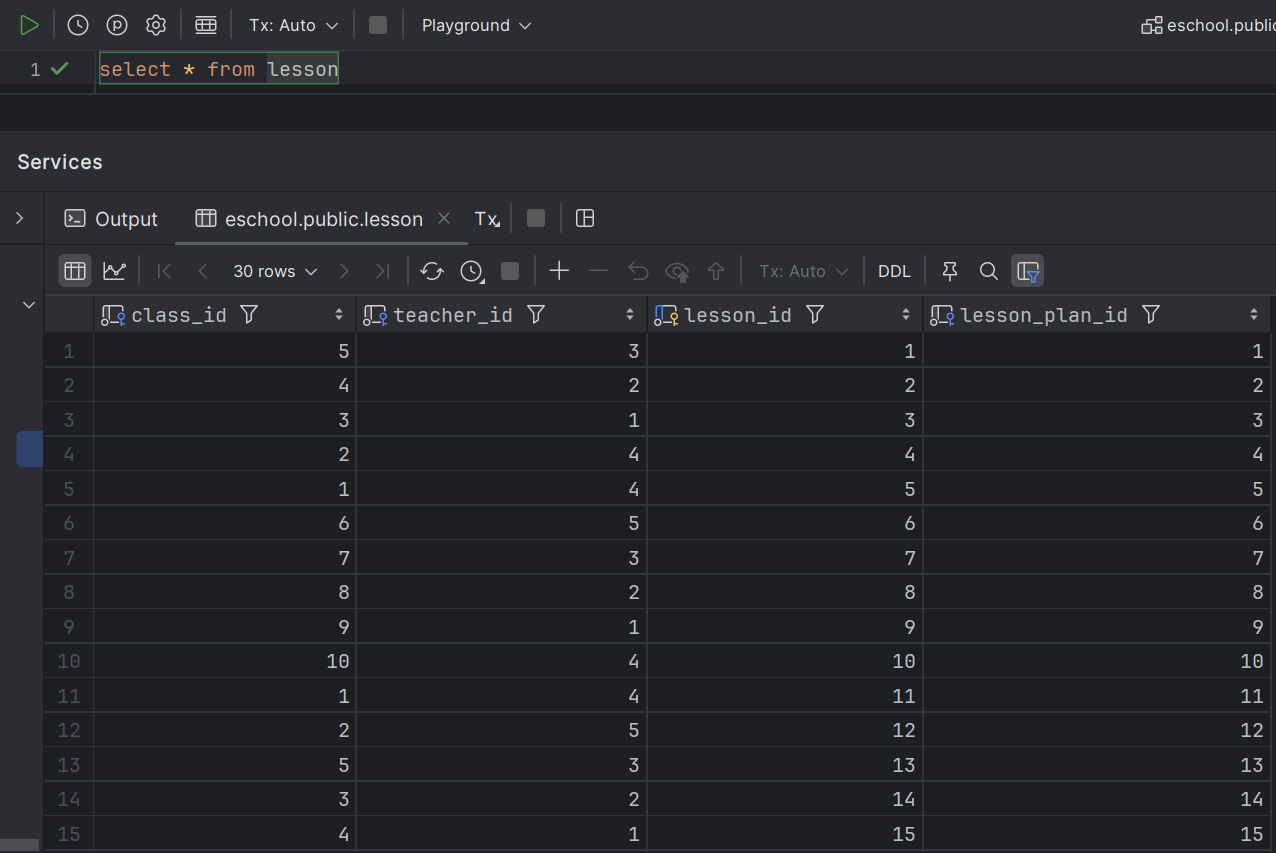


Рисунок 2.5 – Заповнена даними таблиця Lesson

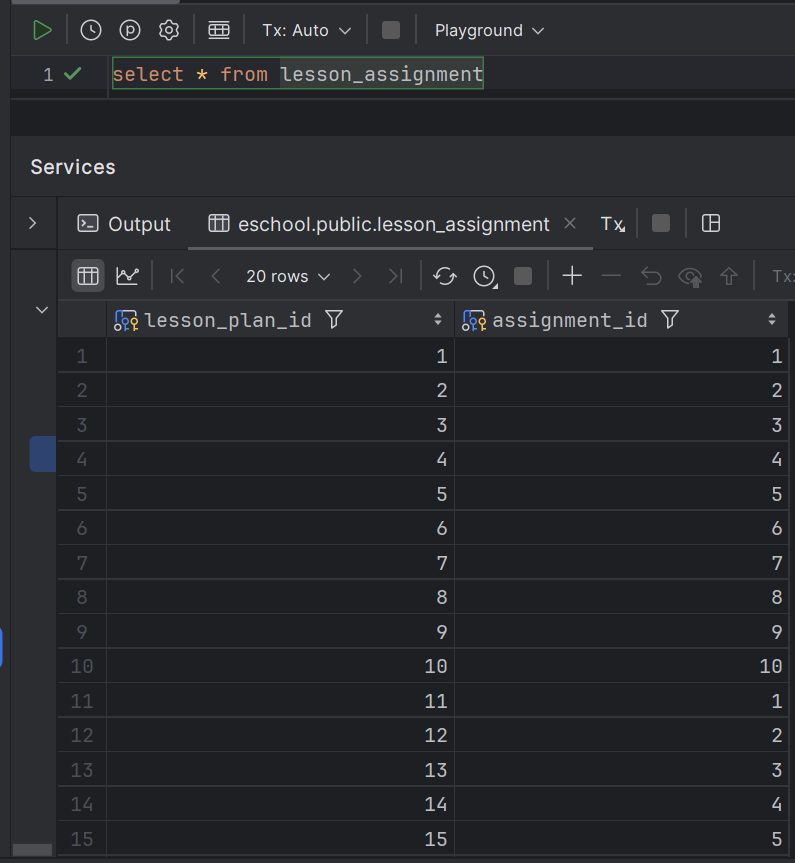


Рисунок 2.6 – Заповнена даними таблиця Lesson\_Assignment

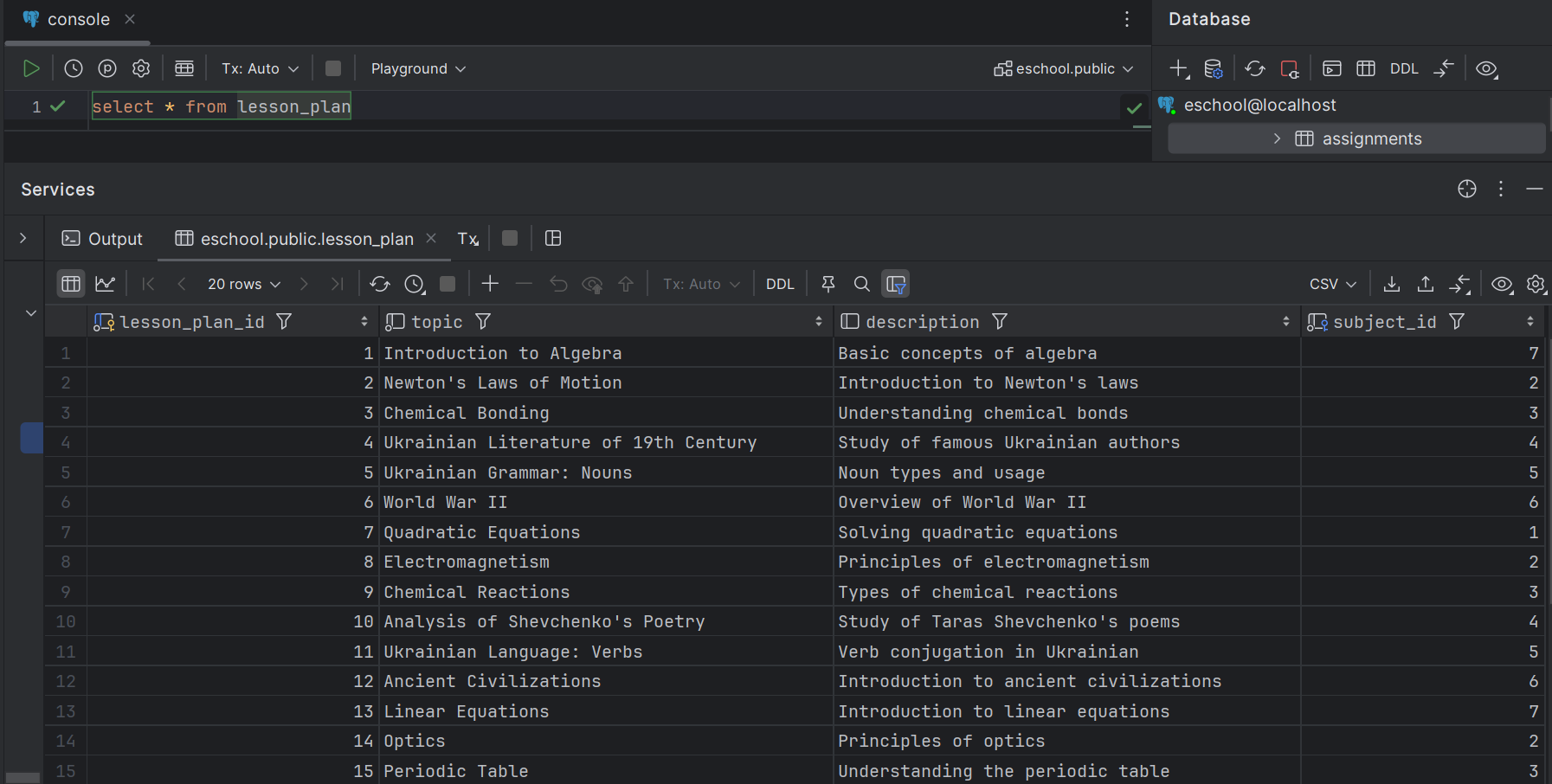


Рисунок 2.7 – Заповнена даними таблиця Lesson\_plan

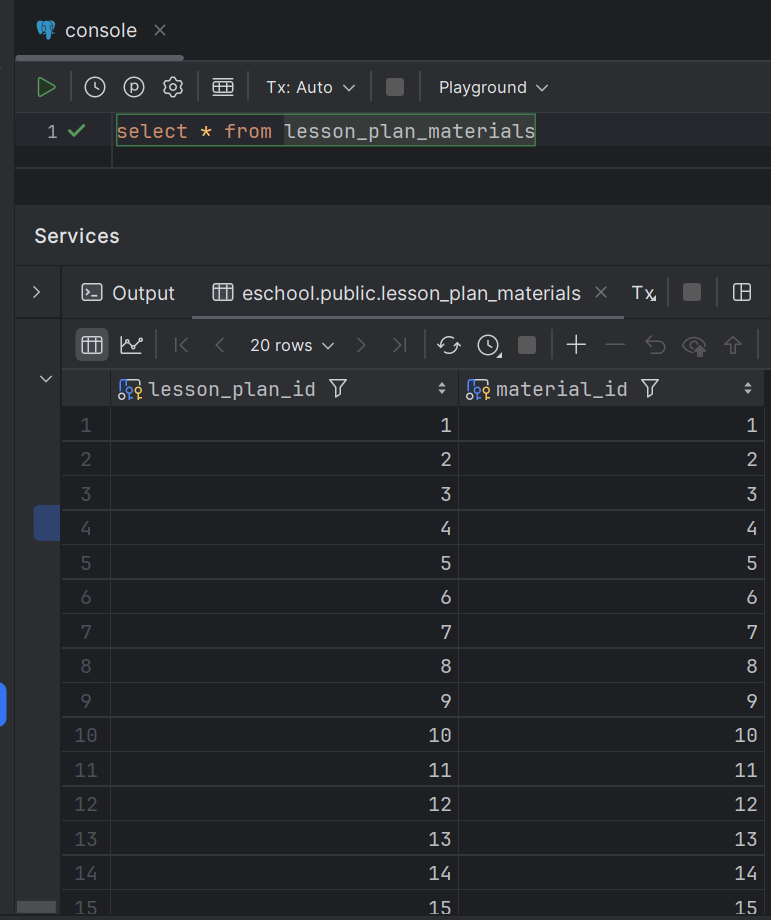


Рисунок 2.8 – Заповнена даними таблиця Lesson\_plan\_materials

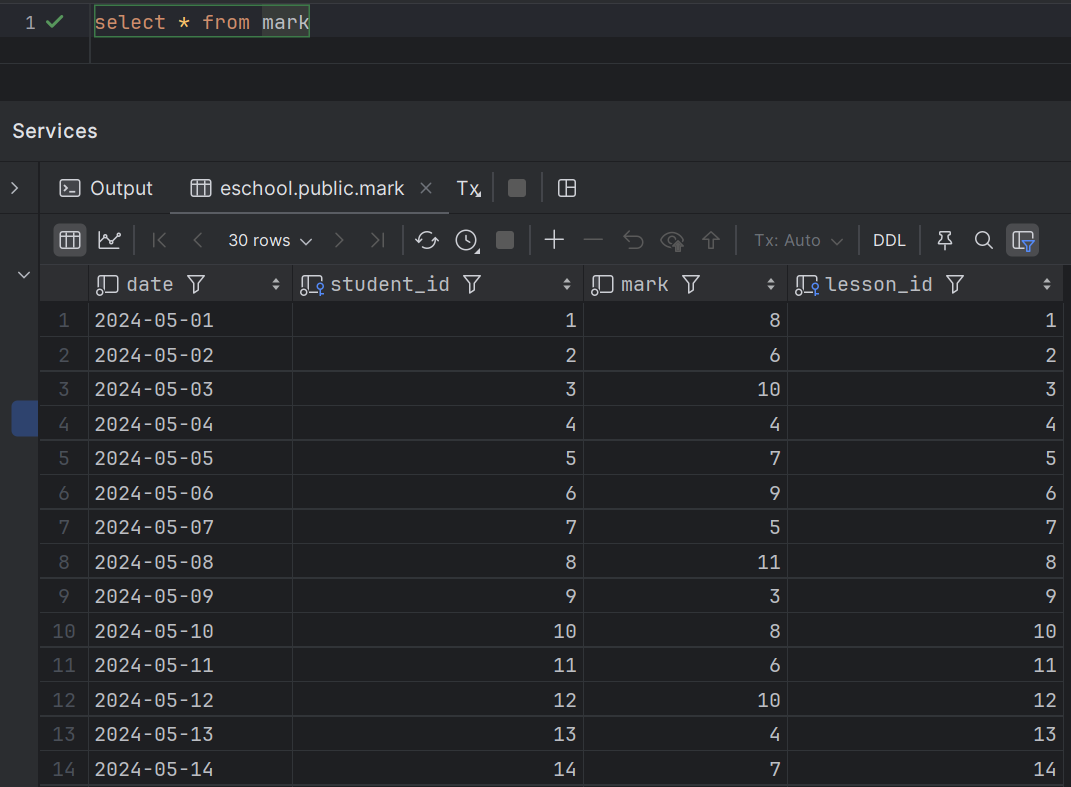


Рисунок 2.9 – Заповнена даними таблиця Mark

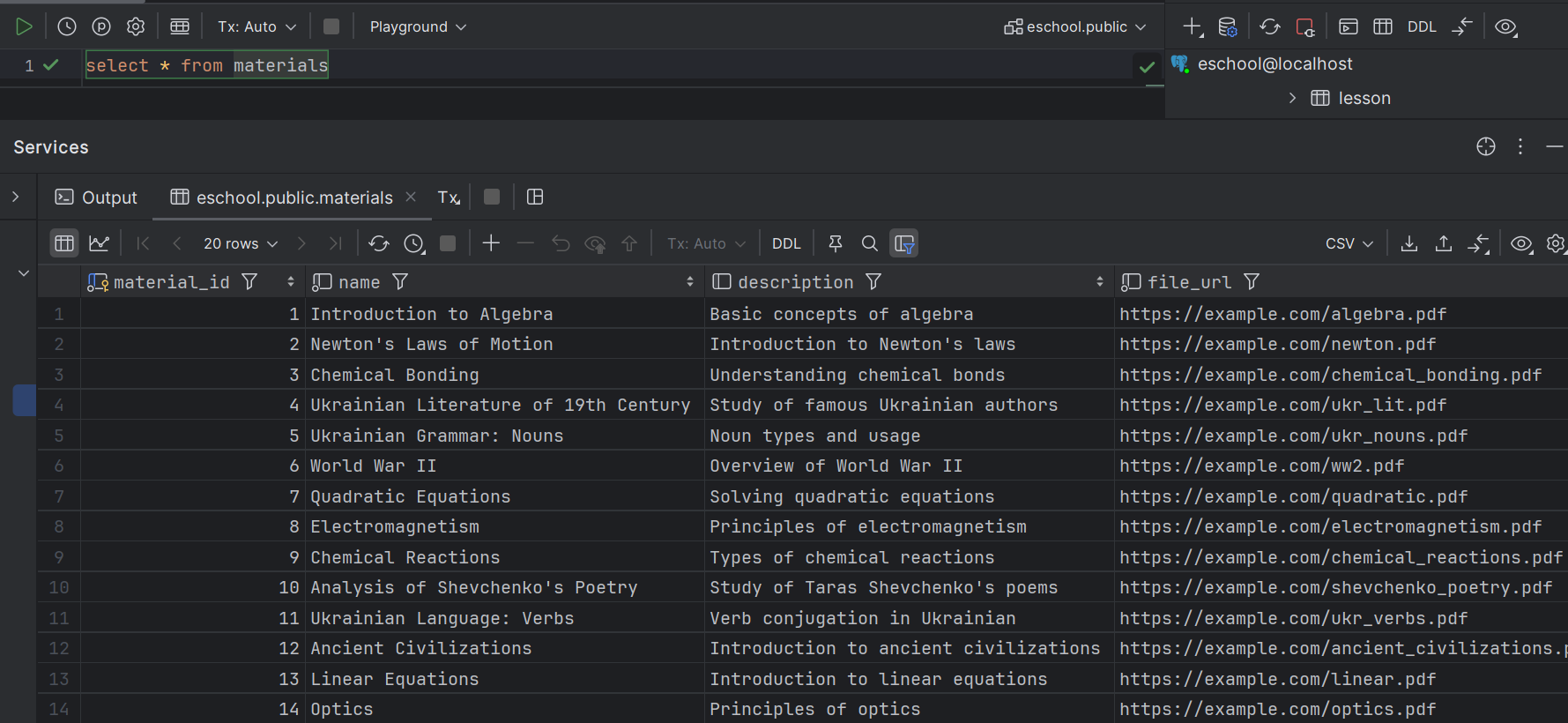


Рисунок 2.10 – Заповнена даними таблиця Materials

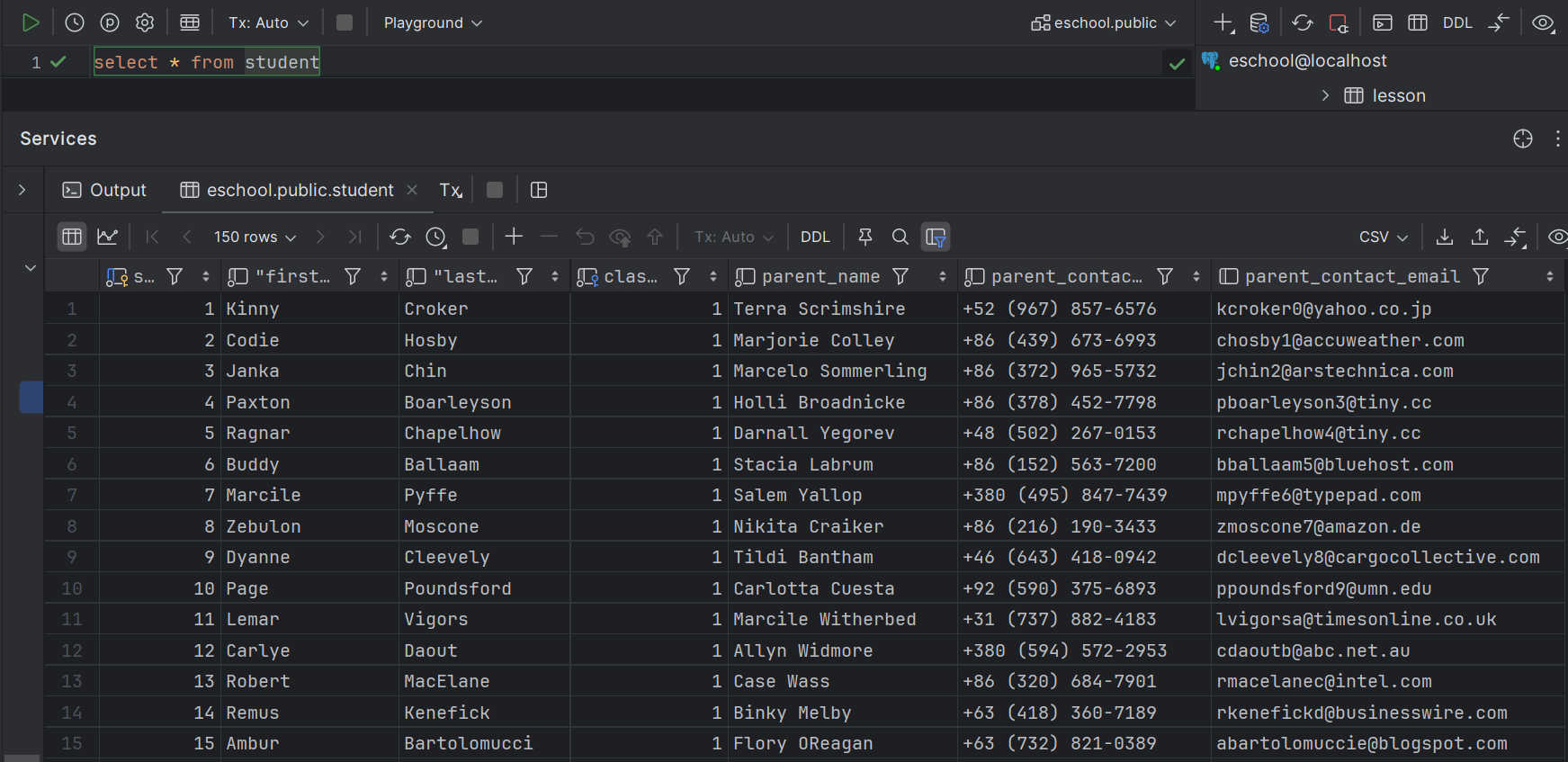


Рисунок 2.11– Заповнена даними таблиця Student

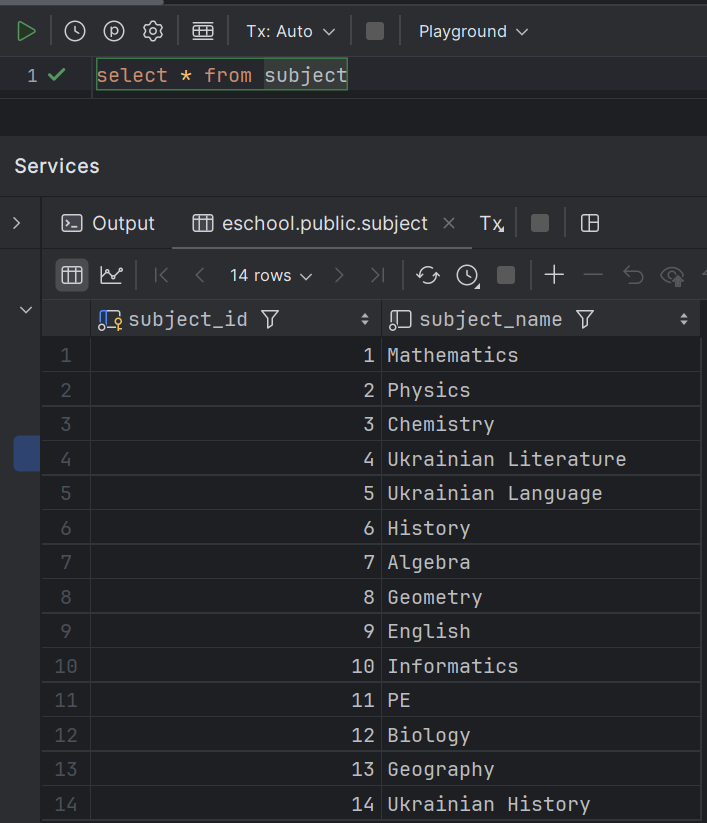


Рисунок 2.12 – Заповнена даними таблиця Subject

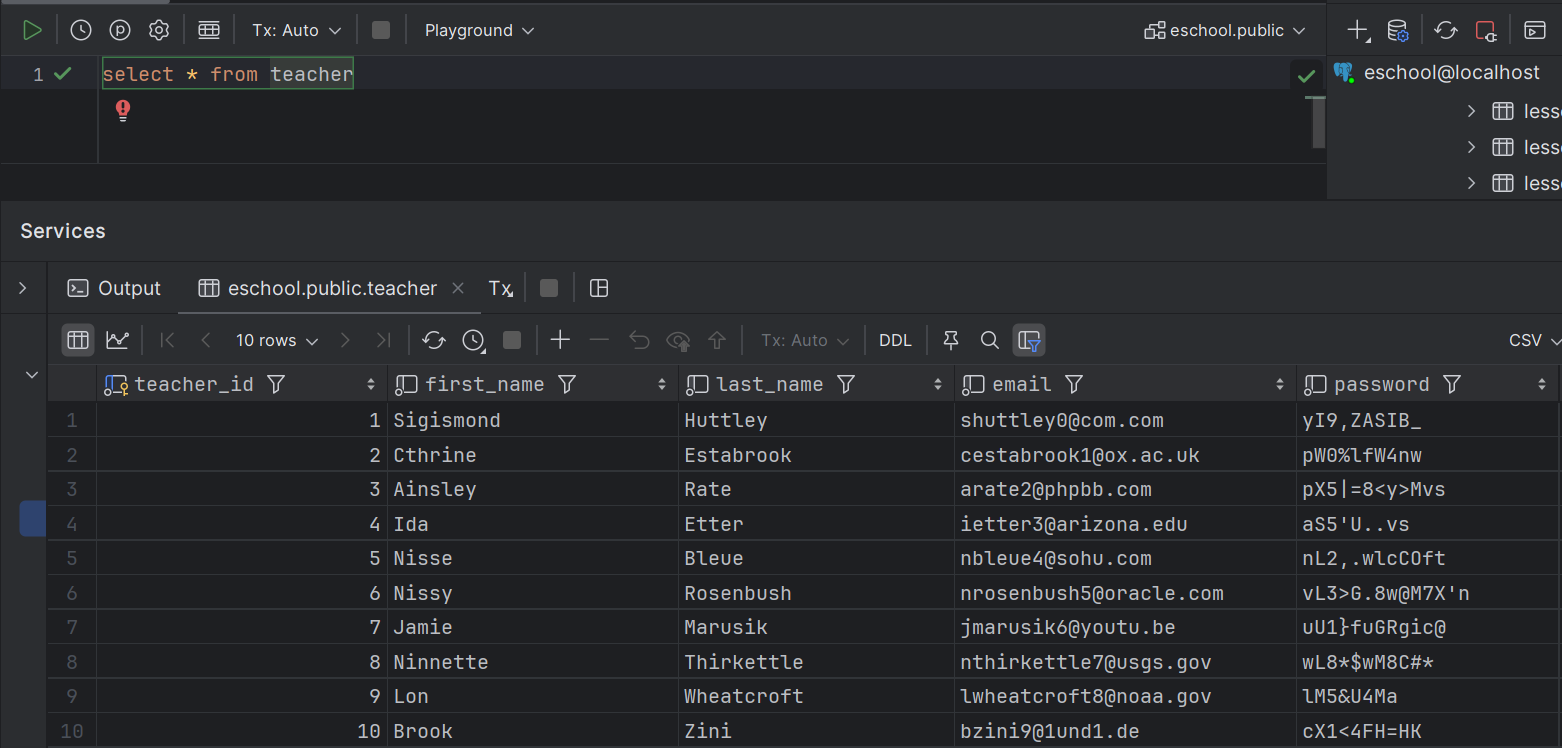


Рисунок 2.13 – Заповнена даними таблиця Teacher

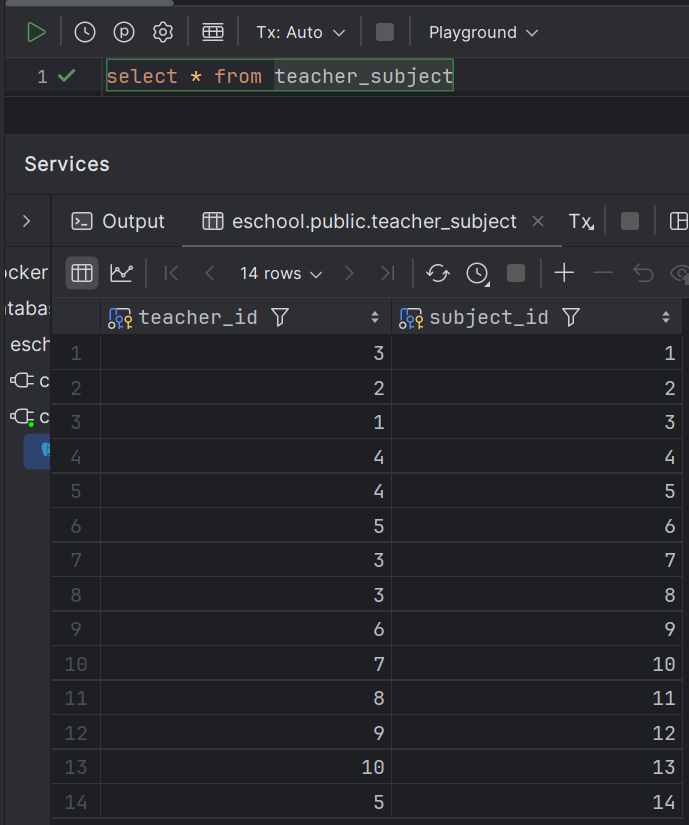


Рисунок 2.14 – Заповнена даними таблиця Teacher\_subject

**2.2 Створення Spring Boot Application**

Для створення Spring Boot Application використаємо інструмент spring initializr, який генерує стартовий код проекту з вибраними нами залежностями.

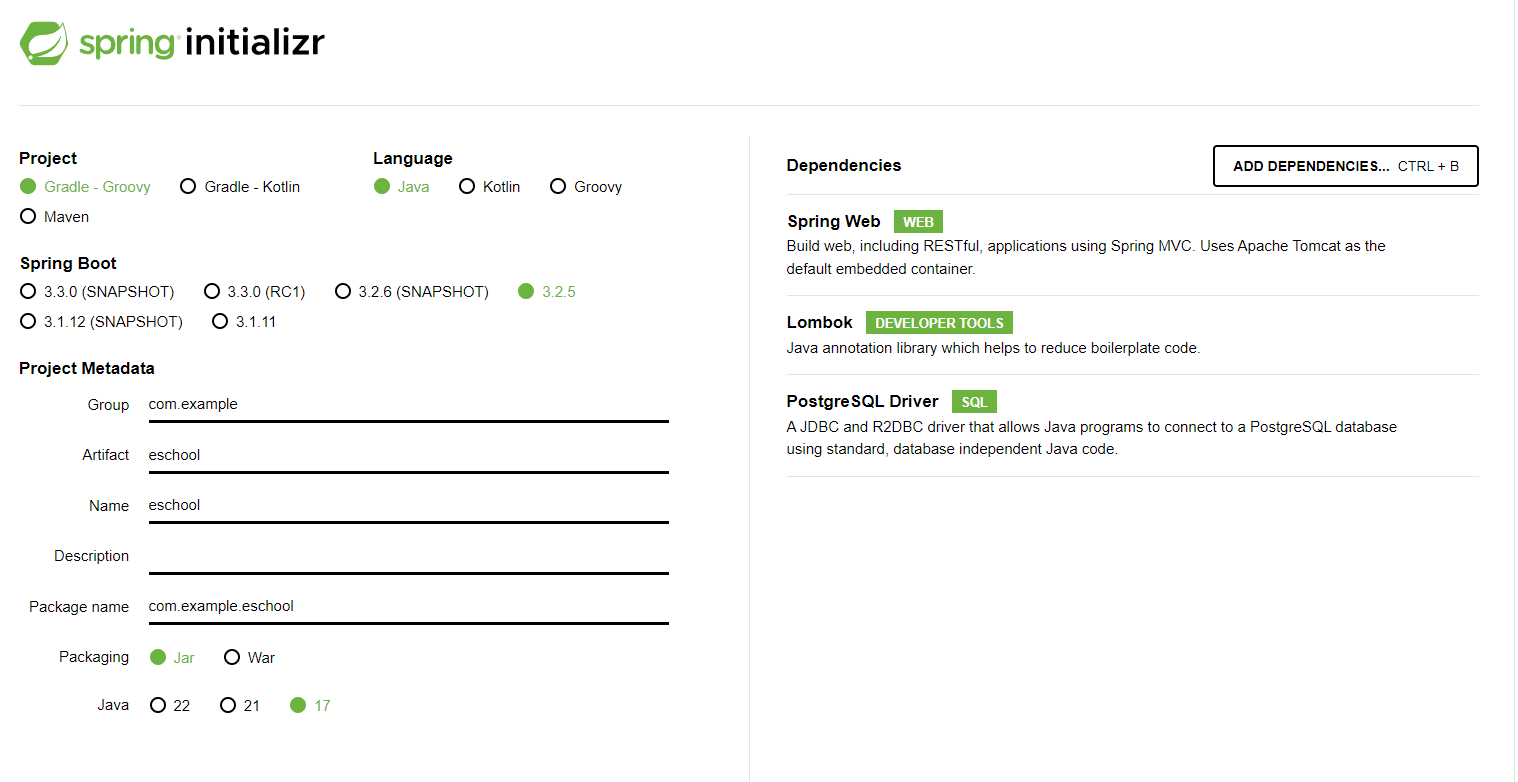


Рисунок 2.15 – Генерація стартового коду

Для розробки будемо використовувати InteliJ Idea. Архітектуру проекту побудуємо за (Model-View-Controller), який часто використовується веб-додатках.

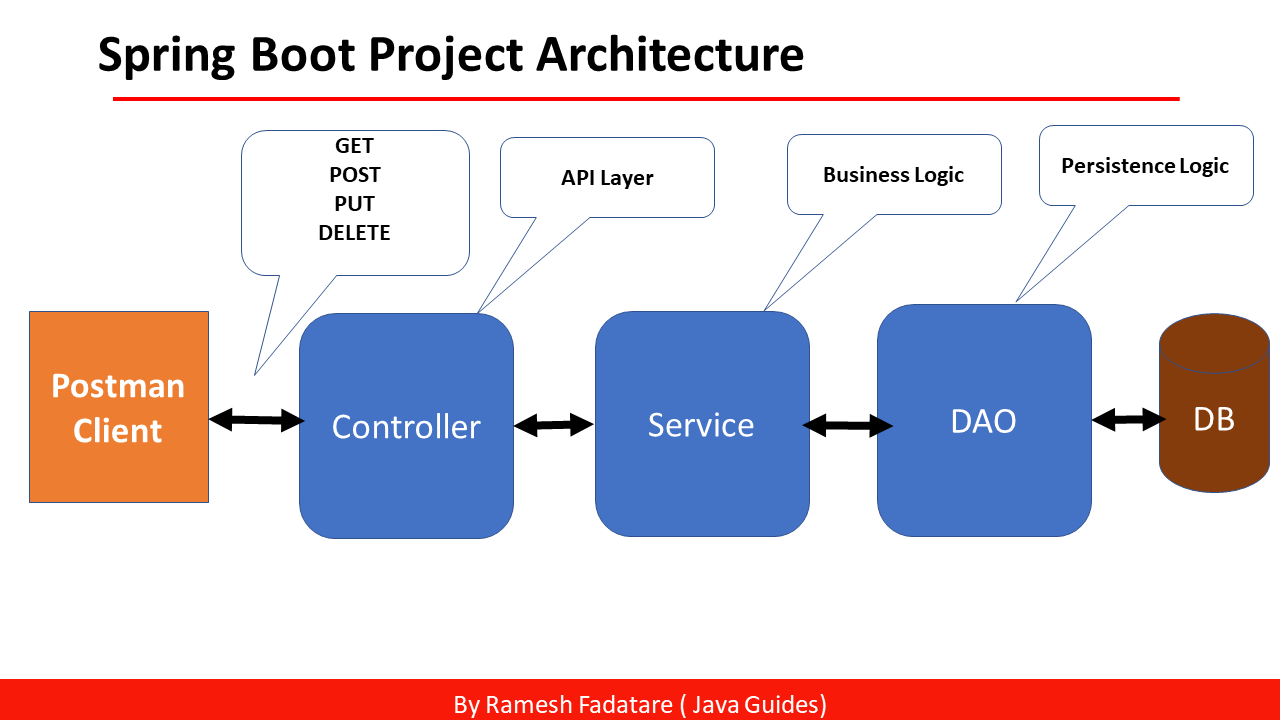


Рисунок 2.16 – Архітектура проекту

Кожен компонент взаємодіє з іншими компонентами за допомогою відповідних інтерфейсів або класів. Наприклад, контролер може викликати методи сервісів, а сервіси в свою чергу використовують репозиторії для доступу до даних. DTO використовуються для передачі даних між цими компонентами, щоб розділити модель даних (Entity) та внутрішню логіку додатку від зовнішнього інтерфейсу (Controller).

Створимо Entity для кожного об’єкту даних, що зберігаються в базі даних. У нашому випадку це будуть:

* Assignment (Завдання),
* Attendance (Відвідування),
* Class (Клас),
* Lesson (Урок),
* LessonPlan (План уроку),
* Mark (Оцінка),
* Material (Матеріал),
* Student (Учень),
* Subject (Предмет),
* Teacher (Вчитель).

До кожної з моделі створимо DTO (Об'єкти переносу даних) - об'єкти, які використовуються для передачі даних між різними частинами додатка, зазвичай між контролером та сервісом.

Також створимо репозиторії - це компоненти, які взаємодіють з базою даних. Вони надають методи для отримання, збереження, оновлення та видалення об'єктів.

Створимо сервіси - компоненти, які виконують бізнес-логіку застосунка. Вони використовують репозиторії для доступу до даних та виконання операцій з ними.

І в кінці створимо контролери - компоненти, які приймають HTTP-запити від клієнта, обробляють їх та повертають відповідь. Вони викликають методи сервісів для виконання потрібних операцій.

Відповідно до визначених в попередній частині User Stories нам необхідно розробити наступний функціонал (представлений в таблиці 1).

Таблиця 1

|  |  |
| --- | --- |
| User Stories | Функціонал |
| Планування уроків. | Користувач може створювати план уроків, додаючи до них додаткові матеріали, завдання, а також редагувати та видаляти непотрібні плани. |
| Відслідковування відвідуваності на кожному уроці. | Користувач може переглядати відвідуваність на уроці та відмічати учнів (присутній чи відсутній). |
| Виставлення оцінок учням. | Користувач може додавати нові оцінки, коригувати або видаляти оцінки. |
| Доступ до електронного журналу класу свого класу. | Користувач може переглядати інформацію про свій клас, його успішність та відвідуваність. |
| Генерація звітів про успішність учнів. | Користувач може генерувати звіти про успішність учня. |

Для планування уроків нам необхідно створити моделі LessonPlan, Assignment та Material. При створенні LessonPlan є можливість прикріпити до нього Assignment та Material.

LessonPlan

@Getter  
@Setter  
@NoArgsConstructor  
@AllArgsConstructor  
@Entity  
@Table(name = "lesson\_plan")  
public class LessonPlan {  
@Id  
 @GeneratedValue(strategy = GenerationType.*AUTO*)  
@Column(name = "lesson\_plan\_id")  
private Long lessonId;  
  
@Column(name = "topic")  
private String topic;  
  
@Column(name = "description")  
private String description;  
  
@OneToOne  
 @JoinColumn(name = "subject\_id")  
private Subject subject;  
  
@ManyToMany  
 @JoinTable(  
 name = "lesson\_assignment",  
 joinColumns = @JoinColumn(name = "lesson\_plan\_id"),  
 inverseJoinColumns = @JoinColumn(name = "assignment\_id")  
 )  
private List<Assignment>assignments;  
  
@ManyToMany  
 @JoinTable(  
 name = "lesson\_plan\_materials",  
 joinColumns = @JoinColumn(name = "lesson\_plan\_id"),  
 inverseJoinColumns = @JoinColumn(name = "material\_id")  
 )  
private List<Material>materials;  
  
}

Assignment

package com.example.eschool.entities;  
  
import jakarta.persistence.\*;  
import lombok.AllArgsConstructor;  
import lombok.Getter;  
import lombok.NoArgsConstructor;  
import lombok.Setter;  
  
import java.util.List;  
  
@Getter  
@Setter  
@NoArgsConstructor  
@AllArgsConstructor  
@Entity  
@Table(name = "assignments")  
public class Assignment {  
@Id  
 @GeneratedValue(strategy = GenerationType.*AUTO*)  
@Column(name = "assignment\_id")  
private Long Id;  
  
@Column(name = "assignment\_type")  
private String type;  
  
@Column(name = "name")  
private String name;  
  
@Column(name = "max\_mark")  
private Integer maxMark;  
  
@ManyToMany(mappedBy = "assignments")  
private List<LessonPlan>lessonPlans;  
  
public Assignment(Long id, String type, String name, Integer maxMark) {  
this.Id = id;  
this.type = type;  
this.name = name;  
this.maxMark = maxMark;  
 }  
}

Material

package com.example.eschool.entities;  
  
import jakarta.persistence.\*;  
import lombok.AllArgsConstructor;  
import lombok.Getter;  
import lombok.NoArgsConstructor;  
import lombok.Setter;  
  
import java.util.List;  
  
@Getter  
@Setter  
@NoArgsConstructor  
@AllArgsConstructor  
@Entity  
@Table(name = "materials")  
public class Material {  
@jakarta.persistence.Id  
 @GeneratedValue(strategy = GenerationType.*AUTO*)  
@Column(name = "material\_id")  
private Long Id;  
  
@Column(name = "name")  
private String name;  
  
@Column(name = "description")  
private String description;  
  
@Column(name = "file\_url")  
private String fileURL;  
  
@ManyToMany(mappedBy = "materials")  
private List<LessonPlan>lessonPlans;  
  
public Material(Long id, String name, String description, String fileURL) {  
this.Id = id;  
this.name = name;  
this.description = description;  
this.fileURL = fileURL;  
 }  
}

Додатково створимо DTO для всіх моделей.

LessonPlanDto

@Getter  
@Setter  
@NoArgsConstructor  
@AllArgsConstructor  
public class LessonPlanDto {  
private Long lessonId;  
private String topic;  
private String description;  
private SubjectDto subject;  
private List<AssignmentDto>assignments;  
private List<MaterialDto>materials;  
}

AssignmentDto

package com.example.eschool.dto;  
  
import lombok.AllArgsConstructor;  
import lombok.Getter;  
import lombok.NoArgsConstructor;  
import lombok.Setter;  
  
@Getter  
@Setter  
@NoArgsConstructor  
@AllArgsConstructor  
public class AssignmentDto {  
private Long Id;  
private String type;  
private String name;  
private Integer maxMark;  
}

MaterialDto

@Getter  
@Setter  
@NoArgsConstructor  
@AllArgsConstructor  
public class MaterialDto {  
private Long Id;  
private String name;  
private String description;  
private String fileURL;  
}

Додатково створимо використаємо Mapper - компонент, який використовується для перетворення об'єктів одного типу на об'єкти іншого типу. В контексті веб-додатків зазвичай використовуються для конвертації між сутностями (Entity) та об'єктами переносу даних (DTO). (Mapper використовується і для інших класів, але лістинг коду не надаємо, бо він аналогічний тому, що нижче).

public class LessonPlanMapper {  
public static LessonPlanDto mapToLessonPlanDto (LessonPlan lessonPlan){  
if (lessonPlan == null) {  
return null;  
 }  
return new LessonPlanDto(  
 lessonPlan.getLessonId(),  
 lessonPlan.getTopic(),  
 lessonPlan.getDescription(),  
 SubjectMapper.*mapToSubjectDto*(lessonPlan.getSubject()),  
 lessonPlan.getAssignments().stream().map(AssignmentMapper::*mapToAssignmentDto*).toList(),  
 lessonPlan.getMaterials().stream().map(MaterialMapper::*mapToMaterialDto*).toList()  
  
 );  
 }  
  
  
public static LessonPlan mapToLessonPlan(LessonPlanDto lessonPlanDto) {  
if (lessonPlanDto == null) {  
return null;  
 } return new LessonPlan(  
 lessonPlanDto.getLessonId(),  
 lessonPlanDto.getTopic(),  
 lessonPlanDto.getDescription(),  
 SubjectMapper.*mapToSubject*(lessonPlanDto.getSubject()),  
 lessonPlanDto.getAssignments().stream().map(AssignmentMapper::*mapToAssignment*).toList(),  
 lessonPlanDto.getMaterials().stream().map(MaterialMapper::*mapToMaterial*).toList()  
 );  
  
 }  
}

Далі створимо репозиторій, який розширює функціонал JpaRepository, та інтерфейс для сервісу. Для зручності сервіс розділимо на інтерфейс та його реалізацію. (Аналогічно далі не будемо надавати лістинги для інших класів).

LessonPlanService інтерфейс:

public interface LessonPlanService {  
 LessonPlanDto createLessonPlan(String topic, String description, Long subjectId, List<Long> assignmentIds, List<Long> materialIds);  
 LessonPlanDto getLessonPlanById(Long id);  
 List<LessonPlanDto>getAllLessonPlans();  
 LessonPlanDto updateLessonPlan(Long lessonPlanId, String topic, String description, Long subjectId, List<Long> assignmentIds, List<Long> materialIds);  
void deleteLessonPlan(Long id);  
}

Реалізація LessonPlanService:

package com.example.eschool.services.implementation;  
  
import com.example.eschool.dto.LessonPlanDto;  
import com.example.eschool.entities.Assignment;  
import com.example.eschool.entities.LessonPlan;  
import com.example.eschool.entities.Material;  
import com.example.eschool.entities.Subject;  
import com.example.eschool.mapper.LessonPlanMapper;  
import com.example.eschool.repositories.AssignmentRepository;  
import com.example.eschool.repositories.LessonPlanRepository;  
import com.example.eschool.repositories.MaterialRepository;  
import com.example.eschool.repositories.SubjectRepository;  
import com.example.eschool.services.LessonPlanService;  
import jakarta.persistence.EntityNotFoundException;  
import lombok.AllArgsConstructor;  
import org.springframework.stereotype.Service;  
  
import java.util.List;  
import java.util.stream.Collectors;  
  
@AllArgsConstructor  
@Service  
public class LessonPlanServiceImpl implements LessonPlanService {  
private LessonPlanRepository lessonPlanRepository;  
private AssignmentRepository assignmentRepository;  
private MaterialRepository materialRepository;  
private SubjectRepository subjectRepository;  
  
*/\*\*  
 \* Creates a new LessonPlan.  
 \*  
 \* @param topic the topic of the lesson plan  
 \* @param description the description of the lesson plan  
 \* @param subjectId the ID of the subject  
 \* @param assignmentIds the list of assignment IDs  
 \* @param materialIds the list of material IDs  
 \* @return the created LessonPlanDto  
 \*/*@Override  
public LessonPlanDto createLessonPlan(String topic, String description, Long subjectId, List<Long> assignmentIds, List<Long> materialIds) {  
try {  
 LessonPlan lessonPlan = new LessonPlan();  
 lessonPlan.setLessonId(lessonPlan.getLessonId());  
 lessonPlan.setTopic(topic);  
 lessonPlan.setDescription(description);  
  
 Subject subject = subjectRepository.findById(subjectId)  
 .orElseThrow(() ->new EntityNotFoundException("Subject not found with ID: " + subjectId));  
 lessonPlan.setSubject(subject);  
  
 List<Assignment> assignments = assignmentIds.stream()  
 .map(id ->assignmentRepository.findById(id)  
 .orElseThrow(() ->new EntityNotFoundException("Assignment not found with ID: " + id)))  
 .collect(Collectors.*toList*());  
 lessonPlan.setAssignments(assignments);  
  
 List<Material> materials = materialIds.stream()  
 .map(id ->materialRepository.findById(id)  
 .orElseThrow(() ->new EntityNotFoundException("Material not found with ID: " + id)))  
 .collect(Collectors.*toList*());  
 lessonPlan.setMaterials(materials);  
  
 LessonPlan savedLessonPlan = lessonPlanRepository.save(lessonPlan);  
return LessonPlanMapper.*mapToLessonPlanDto*(savedLessonPlan);  
 } catch (Exception e) {  
throw new RuntimeException("Failed to create lesson plan", e);  
 }  
 }  
  
*/\*\*  
 \* Gets a LessonPlan by its ID.  
 \*  
 \* @param id the ID of the lesson plan  
 \* @return the LessonPlanDto  
 \*/*@Override  
public LessonPlanDto getLessonPlanById(Long id) {  
try {  
 LessonPlan lessonPlan = lessonPlanRepository.findById(id)  
 .orElseThrow(() ->new EntityNotFoundException("LessonPlan not found with ID: " + id));  
return LessonPlanMapper.*mapToLessonPlanDto*(lessonPlan);  
 } catch (Exception e) {  
throw new RuntimeException("Failed to get lesson plan by ID: " + id, e);  
 }  
 }  
  
*/\*\*  
 \* Gets all LessonPlans.  
 \*  
 \* @return the list of LessonPlanDto  
 \*/*@Override  
public List<LessonPlanDto>getAllLessonPlans() {  
try {  
return lessonPlanRepository.findAll().stream()  
 .map(LessonPlanMapper::*mapToLessonPlanDto*).toList();  
 } catch (Exception e) {  
throw new RuntimeException("Failed to get all lesson plans", e);  
 }  
 }  
  
*/\*\*  
 \* Updates an existing LessonPlan.  
 \*  
 \* @param lessonPlanId the ID of the lesson plan to update  
 \* @param topic the new topic of the lesson plan  
 \* @param description the new description of the lesson plan  
 \* @param subjectId the ID of the new subject  
 \* @param assignmentIds the list of new assignment IDs  
 \* @param materialIds the list of new material IDs  
 \* @return the updated LessonPlanDto  
 \*/*@Override  
public LessonPlanDto updateLessonPlan(Long lessonPlanId, String topic, String description, Long subjectId, List<Long> assignmentIds, List<Long> materialIds) {  
try {  
 LessonPlan existingLessonPlan = lessonPlanRepository.findById(lessonPlanId)  
 .orElseThrow(() ->new EntityNotFoundException("LessonPlan not found with ID: " + lessonPlanId));  
  
 existingLessonPlan.setTopic(topic);  
 existingLessonPlan.setDescription(description);  
  
 Subject subject = subjectRepository.findById(subjectId)  
 .orElseThrow(() ->new EntityNotFoundException("Subject not found with ID: " + subjectId));  
  
 List<Assignment> assignments = assignmentIds.stream()  
 .map(id ->assignmentRepository.findById(id)  
 .orElseThrow(() ->new EntityNotFoundException("Assignment not found with ID: " + id)))  
 .collect(Collectors.*toList*());  
  
 List<Material> materials = materialIds.stream()  
 .map(id ->materialRepository.findById(id)  
 .orElseThrow(() ->new EntityNotFoundException("Material not found with ID: " + id)))  
 .collect(Collectors.*toList*());  
  
 existingLessonPlan.setSubject(subject);  
 existingLessonPlan.setAssignments(assignments);  
 existingLessonPlan.setMaterials(materials);  
  
 LessonPlan updatedLessonPlan = lessonPlanRepository.save(existingLessonPlan);  
return LessonPlanMapper.*mapToLessonPlanDto*(updatedLessonPlan);  
 } catch (Exception e) {  
throw new RuntimeException("Failed to update lesson plan", e);  
 }  
 }  
  
*/\*\*  
 \* Deletes a LessonPlan by its ID.  
 \*  
 \* @param id the ID of the lesson plan to delete  
 \*/*@Override  
public void deleteLessonPlan(Long id) {  
try {  
if (!lessonPlanRepository.existsById(id)) {  
throw new EntityNotFoundException("LessonPlan not found with ID: " + id);  
 }  
lessonPlanRepository.deleteById(id);  
 } catch (Exception e) {  
throw new RuntimeException("Failed to delete lesson plan", e);  
 }  
 }  
}

Також розробимо контролер, за допомогою якого можна створити новий план уроку, отримати перелік всіх уроків, отримати певний урок за його id, оновити план уроку та видалити урок.:

@NoArgsConstructor  
@RestController  
@AllArgsConstructor  
@RequestMapping("/api/lesson\_plans")  
public class LessonPlanController {  
@Autowired  
private LessonPlanService lessonPlanService;  
  
  
@PostMapping("/add")  
public ResponseEntity<LessonPlanDto>createLessonPlan(@RequestBody LessonPlanRequest request) {  
try {  
 LessonPlanDto createdLessonPlan = lessonPlanService.createLessonPlan(request.getTopic(), request.getDescription(), request.getSubjectId(), request.getAssignmentIds(), request.getMaterialIds());  
return new ResponseEntity<>(createdLessonPlan, HttpStatus.*CREATED*);  
 } catch (RuntimeException e) {  
throw e;  
 }  
  
 }  
  
  
@GetMapping("/get/{id}")  
public ResponseEntity<LessonPlanDto>getLessonPlanById(@PathVariable Long id) {  
 LessonPlanDto lessonPlanDto = lessonPlanService.getLessonPlanById(id);  
return new ResponseEntity<>(lessonPlanDto, HttpStatus.*OK*);  
 }  
  
  
@GetMapping("/getAll")  
public ResponseEntity<List<LessonPlanDto>>getAllLessonPlans() {  
 List<LessonPlanDto> lessonPlans = lessonPlanService.getAllLessonPlans();  
return new ResponseEntity<>(lessonPlans, HttpStatus.*OK*);  
 }  
  
  
@PutMapping("/update/{lessonId}")  
public ResponseEntity<LessonPlanDto>updateLessonPlan(  
@PathVariable Long lessonId,  
@RequestBody LessonPlanRequest request) {  
 LessonPlanDto updatedLessonPlan = lessonPlanService.updateLessonPlan(lessonId, request.getTopic(), request.getDescription(), request.getSubjectId(), request.getAssignmentIds(), request.getMaterialIds());  
return new ResponseEntity<>(updatedLessonPlan, HttpStatus.*OK*);  
 }  
  
  
@DeleteMapping("/delete/{id}")  
public ResponseEntity<Void>deleteLessonPlan(@PathVariable Long id) {  
lessonPlanService.deleteLessonPlan(id);  
return new ResponseEntity<>(HttpStatus.*NO\_CONTENT*);  
 }  
}

Перевіримо роботу:

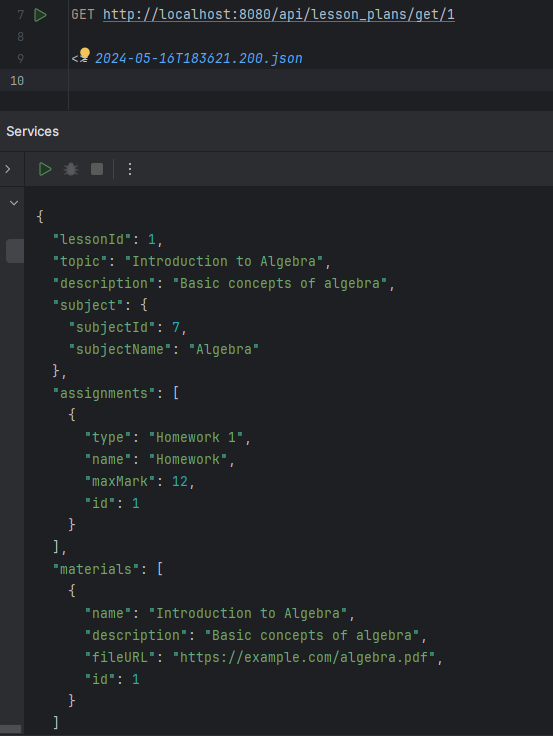


Рисунок 2.17 – Перегляд плану уроку по id

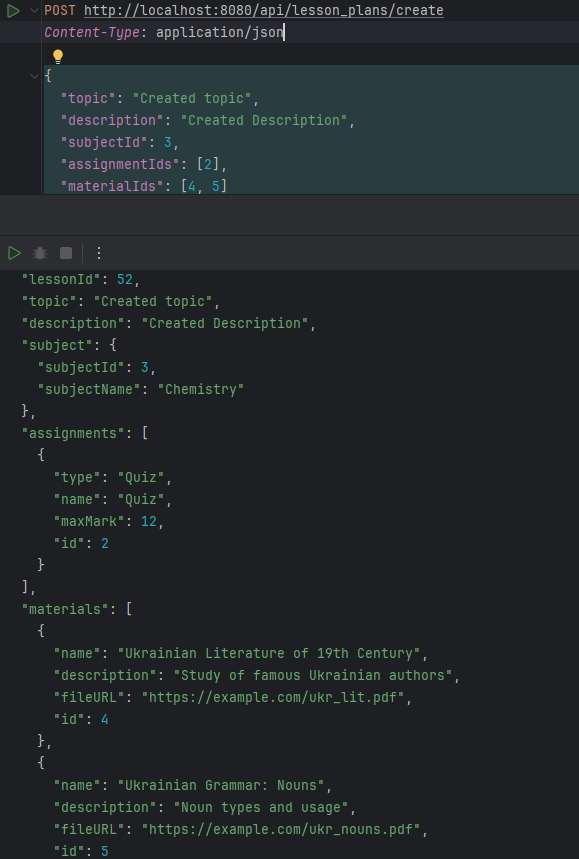


Рисунок 2.18 – Створення нового плану уроку

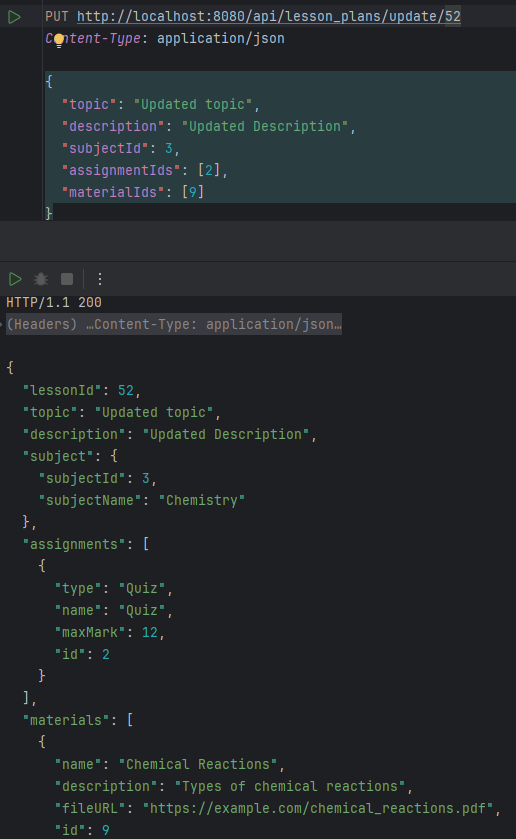


Рисунок 2.19 – Редагування інформації плану уроку

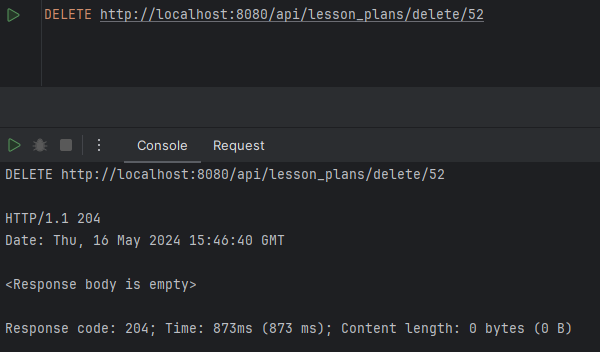


Рисунок 2.20 – Видалення плану уроку

Для роботи з уроками створимо Lesson, LessonService, LessonServiceImplementation, LessonRepository, LessonController.Урок містить в собі інформацію про його id, який клас був присутній, який в уроку був план та вчитель, що його проводив.

Lesson

public class Lesson {  
@Id  
 @GeneratedValue(strategy = GenerationType.*IDENTITY*)  
@Column(name = "lesson\_id")  
private Long lessonId;  
  
@OneToOne  
 @JoinColumn(name = "class\_id")  
private com.example.eschool.entities.Class classLesson;  
  
@OneToOne  
 @JoinColumn(name = "lesson\_plan\_id")  
private LessonPlan lessonPlan;  
  
@OneToOne  
 @JoinColumn(name = "teacher\_id")  
private Teacher teacher;  
}

LessonService

public interface LessonService {  
 List<String>getListofClassStudents(Long lessonId);  
 List<LessonDto>getAllLessonPlans();  
 LessonDto getLessonById(Long lessonId);  
 LessonDto createLesson(String className, Long teacherId, Long lessonPlanId);  
 LessonDto editLesson(Long lessonId, String className, Long teacherId, Long lessonPlanId);  
void deleteLesson(Long lessonId);  
}

LessonServiceImplementation

@Setter  
@Getter  
@Service  
public class LessonServiceImpl implements LessonService {  
private ClassRepository classRepository;  
private StudentRepository studentRepository;  
private LessonRepository lessonRepository;  
private TeacherRepository teacherRepository;  
private LessonPlanRepository lessonPlanRepository;  
  
*/\*\*  
 \* Retrieves list of students that should be present at a lesson.  
 \*  
 \* @param lessonId The ID of the lesson.  
 \* @return The List of the retrieved students' names.  
 \*/*@Override  
public List<String>getListofClassStudents(Long lessonId) {  
try {  
 Lesson lesson = lessonRepository.findById(lessonId)  
 .orElseThrow(() ->new EntityNotFoundException("Lesson not found with ID: " + lessonId));  
  
 List<Student> students = studentRepository.findStudentsByStudentClass(lesson.getClassLesson());  
return students.stream().map(student -> student.getFirstName() + " " + student.getLastName()).toList();  
 } catch (EntityNotFoundException e) {  
throw e;  
 } catch (Exception e) {  
throw new RuntimeException("Failed to get students that should be present at the lesson");  
 }  
 }  
  
*/\*\*  
 \* Retrieves all lessons.  
 \*  
 \* @return the list of LessonDto  
 \*/*@Override  
public List<LessonDto>getAllLessonPlans() {  
try {  
return lessonRepository.findAll().stream()  
 .map(LessonMapper::*mapToLessonDto*).toList();  
 } catch (Exception e) {  
throw new RuntimeException("Failed to get all lessons", e);  
 }  
 }  
  
*/\*\*  
 \* Retrieves lesson by ID.  
 \*  
 \* @param lessonId The ID of the lesson.  
 \* @return The DTO of the retrieved lesson.  
 \*/*@Override  
public LessonDto getLessonById(Long lessonId) {  
try {  
 Lesson lesson = lessonRepository.findById(lessonId)  
 .orElseThrow(() ->new EntityNotFoundException("Lesson not found with ID: " + lessonId));  
return LessonMapper.*mapToLessonDto*(lesson);  
 } catch (Exception e) {  
throw new RuntimeException("Failed to get lesson by ID");  
 }  
 }  
  
*/\*\*  
 \* Creates a new lesson.  
 \*  
 \* @param className, teacherId, lessonPlanId for the new lesson.  
 \* @return The DTO of the created lesson.  
 \*/*@Override  
public LessonDto createLesson(String className, Long teacherId, Long lessonPlanId) {  
try {  
 Lesson lesson = new Lesson();  
 lesson.setLessonId(lesson.getLessonId());  
  
 LessonPlan lp = lessonPlanRepository.findById(lessonPlanId)  
 .orElseThrow(() ->new EntityNotFoundException("Lesson plan not found with ID: " + lesson.getLessonPlan().getLessonId()));  
 lesson.setLessonPlan(lp);  
  
 Teacher teacher = teacherRepository.findById(teacherId)  
 .orElseThrow(() ->new EntityNotFoundException("Teacher not found with ID: " + teacherId));  
 lesson.setTeacher(teacher);  
  
 com.example.eschool.entities.Class cl = classRepository.findById(teacherId)  
 .orElseThrow(() ->new EntityNotFoundException("Class not found with name: " + className));  
 lesson.setClassLesson(cl);  
  
 Lesson savedLesson = lessonRepository.save(lesson);  
return LessonMapper.*mapToLessonDto*(savedLesson);  
 } catch (Exception e) {  
throw new RuntimeException("Failed to create lesson", e);  
 }  
 }  
  
*/\*\*  
 \* Edits an existing lesson.  
 \*  
 \* @param lessonId The ID of the lesson to edit.  
 \* @param className, teacherId, lessonPlanId containing the updated details of the lesson.  
 \* @return The DTO of the edited lesson.  
 \*/*@Override  
public LessonDto editLesson(Long lessonId, String className, Long teacherId, Long lessonPlanId) {  
try {  
 Lesson existingLesson = lessonRepository.findById(lessonId)  
 .orElseThrow(() ->new EntityNotFoundException("Lesson not found with ID: " + lessonId));  
  
 LessonPlan lp = lessonPlanRepository.findById(lessonPlanId)  
 .orElseThrow(() ->new EntityNotFoundException("Lesson plan not found with ID: " + lessonPlanId));  
 existingLesson.setLessonPlan(lp);  
  
 Teacher teacher = teacherRepository.findById(teacherId)  
 .orElseThrow(() ->new EntityNotFoundException("Teacher not found with ID: " + teacherId));  
 existingLesson.setTeacher(teacher);  
  
 com.example.eschool.entities.Class cl = classRepository.findById(teacherId)  
 .orElseThrow(() ->new EntityNotFoundException("Class not found with name: " + className));  
 existingLesson.setClassLesson(cl);  
  
 Lesson updatedLesson = lessonRepository.save(existingLesson);  
return LessonMapper.*mapToLessonDto*(updatedLesson);  
 } catch (Exception e) {  
throw new RuntimeException("Failed to update lesson", e);  
 }  
 }  
  
*/\*\*  
 \* Deletes a lesson by its ID.  
 \*  
 \* @param lessonId The ID of the lesson to delete.  
 \*/*@Override  
public void deleteLesson(Long lessonId) {  
if (!lessonRepository.existsById(lessonId)) {  
throw new EntityNotFoundException("Lesson not found with ID: " + lessonId);  
 }  
lessonRepository.deleteById(lessonId);  
 }  
}

@Getter  
class LessonRequest {  
private String className;  
private Long teacherId;  
private Long lessonPlanId;  
}  
  
@RestController  
@RequestMapping("/api/lessons")  
@Tag(name = "Інформація про уроки")  
public class LessonController {  
  
private static LessonService *lessonService*;  
  
  
@GetMapping("/{lessonId}/students")  
public ResponseEntity<List<String>>getListOfClassStudents(@PathVariable Long lessonId) {  
 List<String> students = *lessonService*.getListofClassStudents(lessonId);  
return ResponseEntity.*ok*(students);  
 }  
  
  
@GetMapping("/{lessonId}")  
public ResponseEntity<LessonDto>getLessonById(@PathVariable Long lessonId) {  
 LessonDto lessonDto = *lessonService*.getLessonById(lessonId);  
return ResponseEntity.*ok*(lessonDto);  
 }  
  
@GetMapping("/getAll")  
public ResponseEntity<List<LessonDto>>getAllLessons() {  
 List<LessonDto> lessons = *lessonService*.getAllLessonPlans();  
return new ResponseEntity<>(lessons, HttpStatus.*OK*);  
 }  
  
@PostMapping("add")  
public ResponseEntity<LessonDto>createLesson(@RequestBody LessonRequest lesson) {  
 LessonDto createdLesson = *lessonService*.createLesson(lesson.getClassName(), lesson.getTeacherId(), lesson.getLessonPlanId());  
return new ResponseEntity<>(createdLesson, HttpStatus.*CREATED*);  
 }  
  
  
@PutMapping("/update/{lessonId}")  
public ResponseEntity<LessonDto>editLesson(@PathVariable Long lessonId, @RequestBody LessonRequest lesson) {  
 LessonDto editedLesson = *lessonService*.editLesson(lessonId, lesson.getClassName(), lesson.getTeacherId(), lesson.getLessonPlanId());  
return ResponseEntity.*ok*(editedLesson);  
 }  
  
@DeleteMapping("/delete/{id}")  
public ResponseEntity<Void>deleteLesson(@PathVariable Long id) {  
*lessonService*.deleteLesson(id);  
return new ResponseEntity<>(HttpStatus.*NO\_CONTENT*);  
 }  
  
}

Для відслідковування відвідуваності створимо класи Attendance, AttendanceDto, AssignmentService, AssignmentServiceImpl, AssignmentController.

Відвідування містить в собі інформацію про студента на уроці, урок, статус (присутній, відсутній – true, false), дата проведення.

Attendance

@Getter  
@Setter  
@NoArgsConstructor  
@AllArgsConstructor  
@Entity  
@Table(name = "attendance")  
public class Attendance {  
  
@Id  
 @GeneratedValue(strategy = GenerationType.*IDENTITY*)  
private Long Id;  
  
@OneToOne  
 @JoinColumn(name = "student\_id")  
private Student student;  
  
@OneToOne  
 @JoinColumn(name = "lesson\_id")  
private Lesson lesson;  
  
@Column(name = "status")  
private Boolean status;  
  
@Column(name = "date")  
private LocalDate date;  
}

AttendanceDto

@Getter  
@Setter  
@AllArgsConstructor  
public class AttendanceDto {  
private Long id;  
private StudentDto student;  
private LessonDto lessonDto;  
private Boolean status;  
private LocalDate date;  
}

Для більш зручного виводу даних створимо ще один DTO:

@Getter  
@Setter  
@NoArgsConstructor  
@AllArgsConstructor  
public class AttendanceResponseDto {  
private String studentName;  
private String studentClass;  
private Boolean presence;  
private LocalDate date;  
private String lessonTeacherName;  
private String lessonTopic;  
}

AttendanceService

@AllArgsConstructor  
@Service  
public class AttendanceServiceImpl implements AttendanceService {  
  
private AttendanceRepository attendanceRepository;  
private StudentRepository studentRepository;  
private LessonRepository lessonRepository;  
  
*/\*\*  
 \* Marks attendance for a student in a lesson.  
 \*  
 \* @param studentId The ID of the student.  
 \* @param lessonId The ID of the lesson.  
 \* @param status The attendance status.  
 \* @throws EntityNotFoundException If the student or lesson is not found.  
 \*/*@Override  
public void markAttendanceByStudent(Long studentId, Long lessonId, Boolean status) {  
try {  
 Student student = studentRepository.findById(studentId)  
 .orElseThrow(() ->new EntityNotFoundException("Student not found with ID: " + studentId));  
  
 Lesson lesson = lessonRepository.findById(lessonId)  
 .orElseThrow(() ->new EntityNotFoundException("Lesson not found with ID: " + lessonId));  
  
 Attendance attendance = new Attendance();  
 attendance.setStudent(student);  
 attendance.setLesson(lesson);  
 attendance.setStatus(status);  
 attendance.setDate(LocalDate.*now*());  
  
attendanceRepository.save(attendance);  
 } catch (EntityNotFoundException e) {  
throw e;  
 } catch (Exception e) {  
throw new RuntimeException("Failed to mark attendance by student");  
 }  
 }  
  
*/\*\*  
 \* Retrieves attendance by ID.  
 \*  
 \* @param attendanceId The ID of the attendance.  
 \* @return The DTO of the retrieved attendance.  
 \*/*@Override  
public AttendanceDto getAttendanceById(Long attendanceId) {  
try {  
 Attendance attendance = attendanceRepository.findById(attendanceId)  
 .orElseThrow(() ->new EntityNotFoundException("Attendance not found with ID: " + attendanceId));  
return AttendanceMapper.*mapToAttendanceDto*(attendance);  
 } catch (Exception e) {  
throw new RuntimeException("Failed to get attendance by ID");  
 }  
 }  
  
*/\*\*  
 \* Retrieves all attendances.  
 \*  
 \* @return The list of attendance DTOs.  
 \*/*@Override  
public List<AttendanceDto>getAllAttendances() {  
try {  
 List<Attendance> attendances = attendanceRepository.findAll();  
return attendances.stream().map(AttendanceMapper::*mapToAttendanceDto*).toList();  
 } catch (Exception e) {  
throw new RuntimeException("Failed to get all attendances");  
 }  
 }  
  
*/\*\*  
 \* Retrieves attendance for a student by student ID.  
 \*  
 \* @param studentId The ID of the student.  
 \* @return The list of attendance DTOs for the student.  
 \*/*@Override  
public List<AttendanceDto>getAttendanceByStudentId(Long studentId) {  
try {  
return attendanceRepository.findAllByStudentId(studentId).stream()  
 .map(AttendanceMapper::*mapToAttendanceDto*).toList();  
 } catch (Exception e) {  
throw new RuntimeException("Failed to get attendance by student ID");  
 }  
 }  
  
*/\*\*  
 \* Retrieves attendance for a lesson by lesson ID.  
 \*  
 \* @param lessonId The ID of the lesson.  
 \* @return The list of attendance DTOs for the lesson.  
 \*/*@Override  
public List<AttendanceDto>getAttendanceByLessonId(Long lessonId) {  
try {  
 Lesson lesson = lessonRepository.findById(lessonId)  
 .orElseThrow(() ->new EntityNotFoundException("Lesson not found with ID: " + lessonId));  
  
return attendanceRepository.findByLesson(lesson).stream()  
 .map(AttendanceMapper::*mapToAttendanceDto*).toList();  
 } catch (Exception e) {  
throw new RuntimeException("Failed to get attendance by lesson ID");  
 }  
 }  
  
*/\*\*  
 \* Deletes attendance by ID.  
 \*  
 \* @param attendanceId The ID of the attendance to delete.  
 \*/*@Override  
public void deleteAttendance(Long attendanceId) {  
try {  
attendanceRepository.deleteById(attendanceId);  
 } catch (Exception e) {  
throw new RuntimeException("Failed to delete attendance");  
 }  
 }

AttendanceController

@AllArgsConstructor  
@RestController  
@RequestMapping("/api/attendance")  
public class AttendanceController {  
  
private AttendanceService attendanceService;  
  
@PostMapping("/markByStudent")  
public ResponseEntity<String>markAttendanceByStudent (@RequestBody MarkAttendanceByStudentRequest request) {  
try {  
if (request.getStatus() == null) {  
return ResponseEntity.*status*(HttpStatus.*BAD\_REQUEST*).body("Status must be either true or false");  
 }  
attendanceService.markAttendanceByStudent(request.getStudentId(), request.getLessonId(), request.getStatus());  
return ResponseEntity.*status*(HttpStatus.*OK*).body("Attendance marked successfully");  
 } catch (RuntimeException e) {  
throw e;  
 }  
  
 }  
  
@GetMapping("/getAll")  
public ResponseEntity<List<AttendanceResponseDto>>getAllAttendance() {  
 List<AttendanceDto> attendance = attendanceService.getAllAttendances();  
return ResponseEntity.*ok*(attendance.stream().map(AttendanceMapper::*convertToResponseDto*).toList());  
 }  
  
  
@GetMapping("/get/{id}")  
public ResponseEntity<AttendanceResponseDto>getAttendanceById(@PathVariable("id") Long id) {  
 AttendanceDto attendanceDto = attendanceService.getAttendanceById(id);  
return ResponseEntity.*ok*(AttendanceMapper.*convertToResponseDto*(attendanceDto));  
 }  
  
  
@GetMapping("/student/{id}")  
public ResponseEntity<List<AttendanceResponseDto>>getAttendanceByStudentId(@PathVariable("id") Long id) {  
 List<AttendanceDto> attendanceDto = attendanceService.getAttendanceByStudentId(id);  
return ResponseEntity.*ok*(attendanceDto.stream().map(AttendanceMapper::*convertToResponseDto*).toList());  
 }  
  
  
@GetMapping("/lesson/{id}")  
public ResponseEntity<List<AttendanceResponseDto>>getAttendanceByLesson(@PathVariable("id") Long id) {  
 List<AttendanceDto> attendanceDto = attendanceService.getAttendanceByLessonId(id);  
return ResponseEntity.*ok*(attendanceDto.stream().map(AttendanceMapper::*convertToResponseDto*).toList());  
 }  
  
  
@DeleteMapping("/delete/{id}")  
public ResponseEntity<String>deleteAttendanceById(@PathVariable("id") Long attendanceId) {  
attendanceService.deleteAttendance(attendanceId);  
return ResponseEntity.*ok*("Attendance deleted successfully");  
 }  
}

Перевіримо роботу:

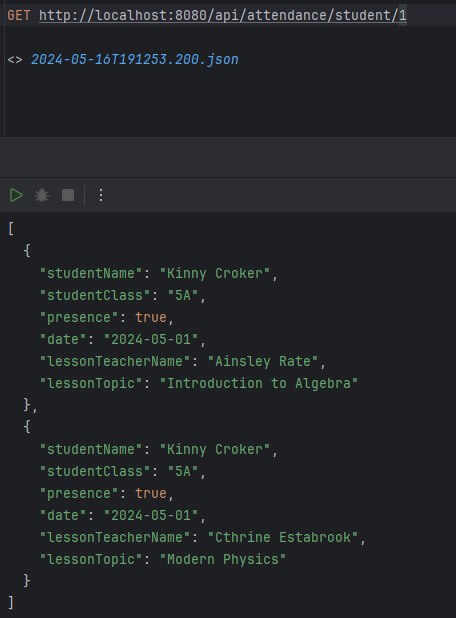


Рисунок 2.21 – Перегляд відвідуваності студента



Рисунок 2.22 – Перегляд присутності учнів на уроці

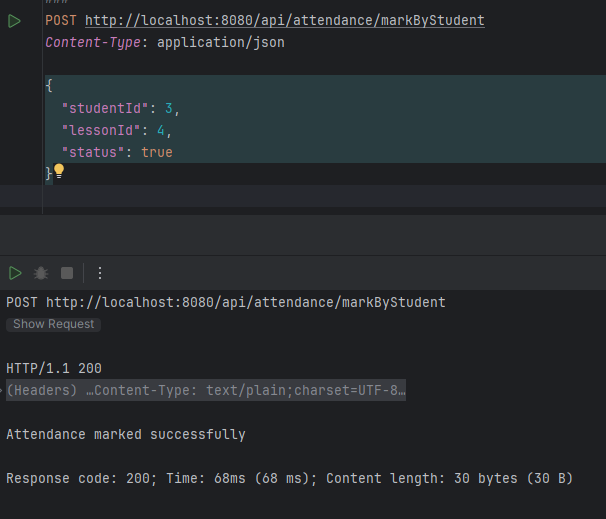


Рисунок 2.23 – Відмітка присутності студента

Для управління оцінками створимо Mark, MarkDto, MarkService, MarkController. Mark містить в собі інформацію про студента, урок, дату проведення.

Mark, MarkDto

@Getter  
@Setter  
@NoArgsConstructor  
@AllArgsConstructor  
@Entity  
@Table(name = "mark")  
public class Mark {  
@jakarta.persistence.Id  
 @GeneratedValue(strategy = GenerationType.*IDENTITY*)  
private Long Id;  
  
@OneToOne  
 @JoinColumn(name = "student\_id")  
private Student student;  
  
@OneToOne  
 @JoinColumn(name = "lesson\_id")  
private Lesson lesson;  
  
@Column(name = "date")  
private LocalDate date;  
  
@Column(name = "mark")  
private Integer mark;  
  
}

@Getter  
@Setter  
@NoArgsConstructor  
@AllArgsConstructor  
public class MarkDto {  
private Long id;  
private StudentDto studentDto;  
private LessonDto lessonDto;  
private LocalDate date;  
private Integer mark;  
}

MarkService

@AllArgsConstructor  
@Service  
public class MarkServiceImpl implements MarkService {  
  
private StudentRepository studentRepository;  
private LessonRepository lessonRepository;  
private MarkRepository markRepository;  
  
*/\*\*  
 \* Retrieves a Mark by its ID.  
 \*  
 \* @param markId the ID of the Mark entity  
 \* @return the MarkDto corresponding to the given ID  
 \*/*@Override  
public MarkDto getMarkById(Long markId) {  
try {  
 Mark mark = markRepository.findById(markId)  
 .orElseThrow(() ->new EntityNotFoundException("Mark not found with ID: " + markId));  
return MarkMapper.*mapToMarkDto*(mark);  
 } catch (Exception e) {  
throw new RuntimeException("Failed to get mark by ID");  
 }  
 }  
  
*/\*\*  
 \* Retrieves Marks by Student ID.  
 \*  
 \* @param studentId the ID of the Student entity  
 \* @return the list of MarkDto objects associated with the given Student ID  
 \*/*@Override  
public List<MarkDto>getMarksByStudentId(Long studentId) {  
  
return markRepository.findAllByStudentId(studentId).stream()  
 .map(MarkMapper::*mapToMarkDto*).toList();  
  
 }  
  
*/\*\*  
 \* Marks a Student for a Lesson with a given Mark value.  
 \*  
 \* @param studentId the ID of the Student entity  
 \* @param lessonId the ID of the Lesson entity  
 \* @param markValue the value of the Mark to be assigned  
 \*/*@Override  
public void markStudent(Long studentId, Long lessonId, Integer markValue) {  
try {  
if (markValue >12) {  
throw new IllegalArgumentException("Mark value cannot be more than 12 or less than 1");  
 } else if (markValue <1) {  
throw new IllegalArgumentException("Mark value cannot be more less than 1");  
 }  
  
 Student student = studentRepository.findById(studentId)  
 .orElseThrow(() ->new EntityNotFoundException("Student not found with ID: " + studentId));  
 Lesson lesson = lessonRepository.findById(lessonId)  
 .orElseThrow(() ->new EntityNotFoundException("Lesson not found with ID: " + lessonId));  
  
 Mark mark = new Mark();  
 mark.setStudent(student);  
 mark.setLesson(lesson);  
 mark.setMark(markValue);  
 mark.setDate(LocalDate.*now*());  
  
markRepository.save(mark);  
 } catch (IllegalArgumentException | EntityNotFoundException e) {  
throw e;  
 } catch (Exception e) {  
throw new RuntimeException("Failed to save new mark");  
 }  
 }  
  
*/\*\*  
 \* Deletes a Mark by its ID.  
 \*  
 \* @param markId the ID of the Mark entity to be deleted  
 \*/*@Override  
public void deleteMark(Long markId) {  
try {  
markRepository.deleteById(markId);  
 } catch (Exception e) {  
throw new RuntimeException("Failed to delete mark");  
 }  
 }

MarkController

@AllArgsConstructor  
@RestController  
@RequestMapping("/api/marks")

public class MarkController {  
  
public MarkService markService;  
  
@GetMapping("/{id}")  
  
public ResponseEntity<MarkResponseDto>getMarkById(@PathVariable("id") Long id) {  
 MarkDto markDto = markService.getMarkById(id);  
return ResponseEntity.*ok*(MarkMapper.*convertToResponseDto*(markDto));  
 }  
  
  
@GetMapping("/student/{id}")  
public ResponseEntity<List<MarkResponseDto>>getMarkByStudentId(@PathVariable("id") Long id) {  
 List<MarkDto> markDto = markService.getMarksByStudentId(id);  
return ResponseEntity.*ok*(markDto.stream().map(MarkMapper::*convertToResponseDto*).toList());  
 }  
  
  
@PostMapping("/addMark")  
public ResponseEntity<String>addMark (@RequestBody MarkRequest request) {  
try {  
markService.markStudent(request.getStudentId(), request.getLessonId(), request.getMark());  
return ResponseEntity.*status*(HttpStatus.*OK*).body("Mark added successfully");  
 } catch (EntityNotFoundException e) {  
return ResponseEntity.*status*(HttpStatus.*NOT\_FOUND*).body("Lesson or student not found");  
 } catch (IllegalArgumentException e) {  
return ResponseEntity.*status*(HttpStatus.*BAD\_REQUEST*).body("Invalid mark value");  
 } catch (RuntimeException e) {  
throw e;  
 }  
 }  
  
  
@DeleteMapping("/delete/{id}")  
public ResponseEntity<String>deleteMarkById(@PathVariable("id") Long markId) {  
markService.deleteMark(markId);  
return ResponseEntity.*ok*("Mark deleted successfully");  
 }

Перевіримо роботу:

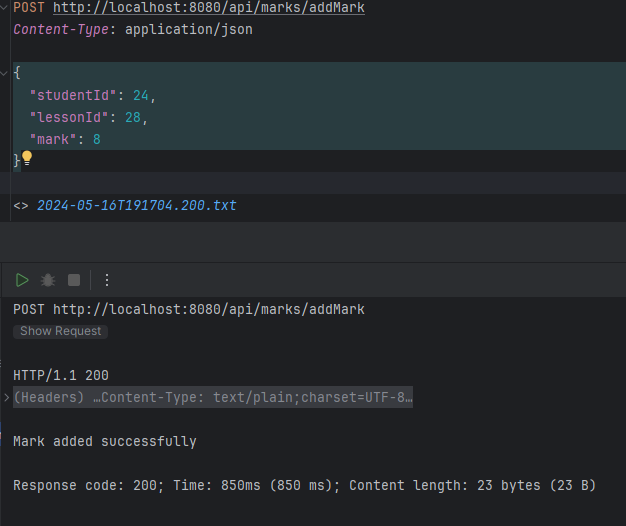


Рисунок 2.24 – Додавання оцінки учню

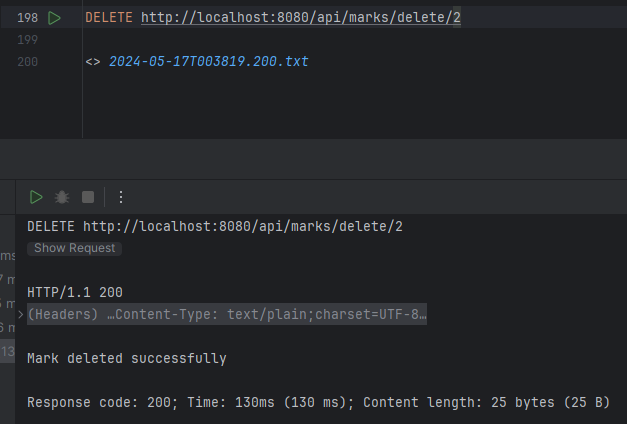


Рисунок 2.25 – Видалення оцінки

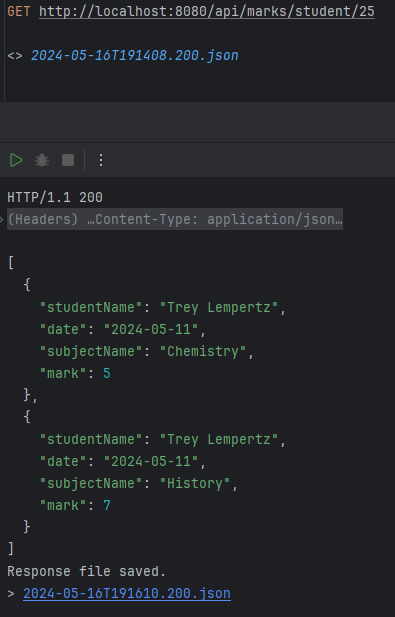


Рисунок 2.26 – Переглянути оцінки учня

Для роботи з класом створимо клас Class, який містить в собі інформацію назву класу, класного керівника, рік початку навчання і кінець.

Class

@Getter  
@Setter  
@NoArgsConstructor  
@AllArgsConstructor  
@Entity  
@Table(name = "classes")  
public class Class {  
@Id  
 @GeneratedValue(strategy = GenerationType.*IDENTITY*)  
@Column (name = "class\_id")  
private Long id;  
  
@Column (name = "name")  
private String className;  
  
@OneToOne  
 @JoinColumn (name = "teacher\_id")  
private Teacher teacher;  
  
@Column (name = "year\_start")  
private Integer yearStart;  
  
@Column (name = "year\_finish")  
private Integer yearFinish;  
}

ClassDto

@Getter  
@Setter  
@NoArgsConstructor  
@AllArgsConstructor  
public class ClassDto {  
private Long id;  
private String className;  
private TeacherDto teacherDto;  
private Integer yearStart;  
private Integer yearFinish;  
}

ClassService

@AllArgsConstructor  
@Service  
public class ClassServiceImpl implements ClassService {  
private final AttendanceRepository attendanceRepository;  
private final StudentRepository studentRepository;  
private ClassRepository classRepository;  
private final MarkRepository markRepository;  
  
*/\*\*  
 \* Retrieves all classes.  
 \*  
 \* @return a list of ClassDto representing all classes.  
 \*/*@Override  
public List<ClassDto>getAllClasses() {  
 List<Class> classes = classRepository.findAll();  
return classes.stream().map(ClassMapper::*mapToClassDto*).toList();  
 }  
  
*/\*\*  
 \* Retrieves a class by its ID.  
 \*  
 \* @param id the ID of the class.  
 \* @return the ClassDto representing the class.  
 \*/*@Override  
public ClassDto getClassById(Long id) {  
 Class classes = classRepository.findById(id)  
 .orElseThrow(() ->new EntityNotFoundException("Class not found with ID: " + id));  
return ClassMapper.*mapToClassDto*(classes);  
 }  
  
*/\*\*  
 \* Retrieves information about students.  
 \*  
 \* @param classId the ID of the class.  
 \* @return a list of StudentResponseDto representing the students info records for the class.  
 \*/*@Override  
public List<StudentResponseDto>getStudentsInfo(Long classId){  
 Class aClass = classRepository.findById(classId)  
 .orElseThrow(() ->new EntityNotFoundException("Class not found with ID: " + classId));  
  
 List<Student> students = studentRepository.findStudentsByStudentClass(aClass);  
if (students.isEmpty()) throw new EntityNotFoundException("Class has no students");  
  
return students.stream()  
 .map(student ->new StudentResponseDto(  
 student.getFirstName() + " " + student.getLastName(),  
 student.getParentName(),  
 student.getParentContactPhone(),  
 student.getParentContactEmail())).toList();  
 }  
  
  
  
  
*/\*\*  
 \* Retrieves the attendance for a specific class.  
 \*  
 \* @param classId the ID of the class.  
 \* @return a list of AttendanceDto representing the attendance records for the class.  
 \*/*public List<AttendanceDto>getClassAttendance(Long classId) {  
try {  
 Class aClass = classRepository.findById(classId)  
 .orElseThrow(() ->new EntityNotFoundException("Class not found with ID: " + classId));  
  
 List<Student> students = studentRepository.findStudentsByStudentClass(aClass);  
if (students.isEmpty()) throw new RuntimeException("Class has no students");  
  
 List<AttendanceDto> attendanceDtoList = new ArrayList<>();  
for (Student student : students) {  
 List<Attendance> studentAttendance = attendanceRepository.findAllByStudentId(student.getId());  
for (Attendance attendance : studentAttendance) {  
 AttendanceDto attendanceDto = AttendanceMapper.*mapToAttendanceDto*(attendance);  
 attendanceDtoList.add(attendanceDto);  
 }  
 }  
  
return attendanceDtoList;  
 } catch (Exception e) {  
throw new RuntimeException("Failed to get class attendance");  
 }  
 }  
  
*/\*\*  
 \* Retrieves the marks for a specific class.  
 \*  
 \* @param classId the ID of the class.  
 \* @return a list of MarkDto representing the marks for the class.  
 \*/*public List<MarkDto>getClassMarks(Long classId) {  
 Class aClass = classRepository.findById(classId)  
 .orElseThrow(() ->new EntityNotFoundException("Class not found with ID: " + classId));  
  
 List<Student> students = studentRepository.findStudentsByStudentClass(aClass);  
 List<MarkDto> markDtoList = new ArrayList<>();  
for (Student student : students) {  
 List<Mark> studentMarks = markRepository.findAllByStudentId(student.getId());  
for (Mark mark : studentMarks) {  
 MarkDto markDto = MarkMapper.*mapToMarkDto*(mark);  
 markDtoList.add(markDto);  
 }  
 }  
return markDtoList;  
 }  
  
*/\*\*  
 \* Retrieves all classes taught by a specific teacher.  
 \*  
 \* @param teacherId the ID of the teacher.  
 \* @return a list of ClassDto representing the classes taught by the teacher.  
 \*/*@Override  
public List<ClassDto>getClassesByTeacher(Long teacherId) {  
try {  
 List<Class> classes = classRepository.findByTeacherId(teacherId);  
return classes.stream().map(ClassMapper::*mapToClassDto*).toList();  
 } catch (Exception e) {  
throw new RuntimeException("Failed to get classes by teacher");  
 }  
 }  
  
*/\*\*  
 \* Generates a report for a specific student.  
 \*  
 \* @param studentId the ID of the student.  
 \* @return a StudentReportDto representing the student's report.  
 \*/*public StudentReportDto generateStudentReport(Long studentId) {  
 Student student = studentRepository.findById(studentId)  
 .orElseThrow(() ->new EntityNotFoundException("Student not found with ID: " + studentId));  
  
 List<Attendance> attendanceList = attendanceRepository.findAllByStudentId(studentId);  
 List<Mark> markList = markRepository.findAllByStudentId(studentId);  
  
long totalLessons = attendanceList.size();  
long missedLessons = attendanceList.stream().filter(attendance -> !attendance.getStatus()).count();  
  
 Map<String, List<Mark>> marksBySubject = markList.stream()  
 .collect(Collectors.*groupingBy*(mark -> mark.getLesson().getLessonPlan().getSubject().getSubjectName()));  
 Map<String, Double> averageMarksBySubject = marksBySubject.entrySet().stream()  
 .collect(Collectors.*toMap*(Map.Entry::getKey,  
 entry -> entry.getValue().stream().mapToInt(Mark::getMark).average().orElse(0.0)));  
  
 Double average = markList.stream().mapToInt(Mark::getMark).average().orElse(0.0);  
  
return new StudentReportDto(student.getFirstName() + " " + student.getLastName(), missedLessons, totalLessons - missedLessons, averageMarksBySubject, average);  
 }  
  
*/\*\*  
 \* Deletes a class by its ID.  
 \*  
 \* @param id the ID of the class to be deleted.  
 \*/*@Override  
public void deleteClassById(Long id) {  
try {  
classRepository.deleteById(id);  
 } catch (Exception e) {  
throw new RuntimeException("Failed to delete attendance");  
 }  
 }  
}

ClassController

@RestController  
@RequestMapping("/api/classes")  
public class ClassController {  
@Autowired  
private ClassService classService;  
  
@GetMapping("/getAllClasses")  
public List<ClassDto>getAllClasses() {  
return classService.getAllClasses();  
 }  
@GetMapping("/get/byTeacher/{teacherId}")  
public List<ClassDto>getClassesByTeacher(@PathVariable Long teacherId) {  
return classService.getClassesByTeacher(teacherId);  
 }  
  
  
@GetMapping("/get/{id}")  
public ResponseEntity<ClassDto>getClassById(@PathVariable("id") Long classId) {  
 ClassDto classDto = classService.getClassById(classId);  
return ResponseEntity.*ok*(classDto);  
 }  
  
  
@GetMapping("/get/{id}/studentsInfo")  
public ResponseEntity<List<StudentResponseDto>>getStudentsInfo(@PathVariable("id") Long classId) {  
try {  
 List<StudentResponseDto> students = classService.getStudentsInfo(classId);  
return ResponseEntity.*ok*(students);  
 } catch (EntityNotFoundException e) {  
return ResponseEntity.*status*(HttpStatus.*NOT\_FOUND*).build();  
 }  
  
 }  
  
  
@GetMapping("/get/{classId}/attendance")  
public ResponseEntity<List<AttendanceResponseDto>>getClassAttendance(@PathVariable Long classId) {  
try {  
 List<AttendanceResponseDto> attendanceDtoList = classService.getClassAttendance(classId).stream()  
 .map(AttendanceMapper::*convertToResponseDto*).toList();  
return ResponseEntity.*ok*(attendanceDtoList);  
 } catch (EntityNotFoundException e) {  
return ResponseEntity.*status*(HttpStatus.*NOT\_FOUND*).body(null);  
 } catch (Exception e) {  
return ResponseEntity.*status*(HttpStatus.*INTERNAL\_SERVER\_ERROR*).build();  
 }  
 }  
  
  
@GetMapping("/get/{classId}/marks")  
public ResponseEntity<List<MarkResponseDto>>getClassMarks(@PathVariable Long classId) {  
try {  
 List<MarkResponseDto> markDtoList = classService.getClassMarks(classId).stream()  
 .map(MarkMapper::*convertToResponseDto*).toList();  
return ResponseEntity.*ok*(markDtoList);  
 } catch (EntityNotFoundException e) {  
return ResponseEntity.*status*(HttpStatus.*NOT\_FOUND*).body(null);  
 } catch (Exception e) {  
return ResponseEntity.*status*(HttpStatus.*INTERNAL\_SERVER\_ERROR*).build();  
 }  
 }  
  
@Operation(  
 description = "Видалити клас",  
 summary = "Видаляє клас з системи (тільки для адміністраторів)"  
)  
@DeleteMapping("/delete/{id}")  
public void deleteClassById(@PathVariable Long id) {  
classService.deleteClassById(id);  
 }  
  
  
  
@GetMapping("/get/report/student/{studentId}")  
public ResponseEntity<StudentReportDto>generateStudentReport(@PathVariable Long studentId) {  
 StudentReportDto studentReport = classService.generateStudentReport(studentId);  
return new ResponseEntity<>(studentReport, HttpStatus.*OK*);  
 }  
}

Перевіримо функціонал:

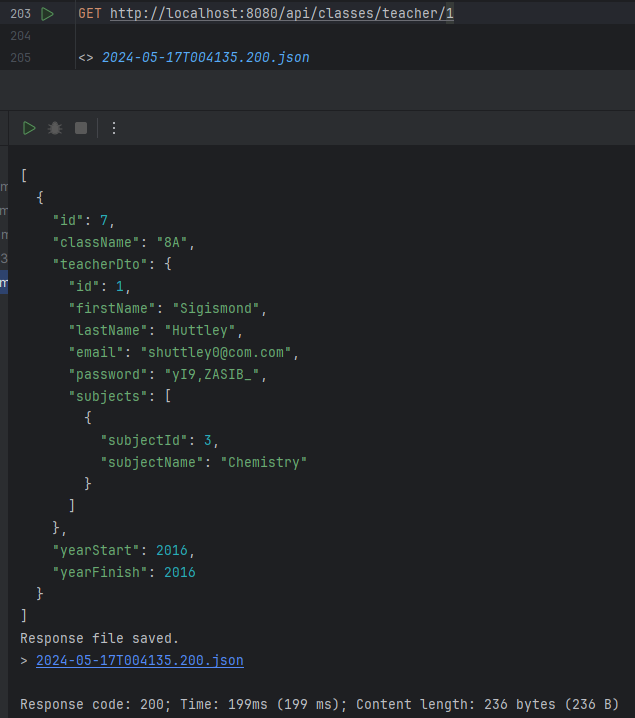


Рисунок 2.27– Переглянути інформацію про свій клас

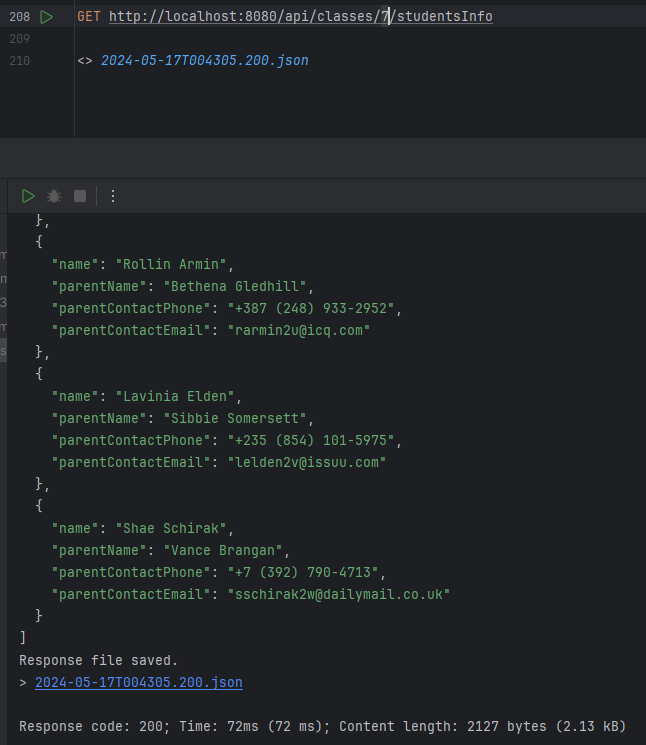


Рисунок 2.28– Переглянути інформацію про всіх учнів класу

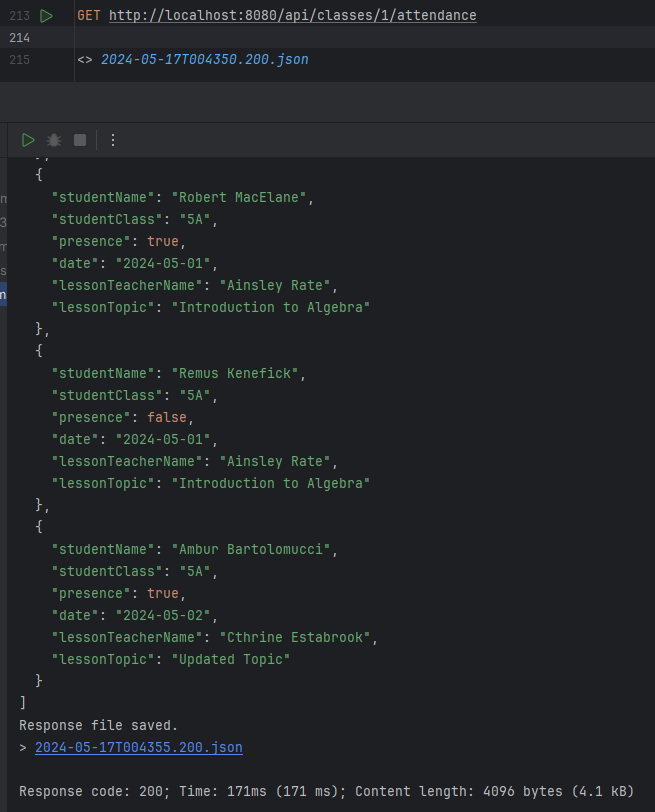


Рисунок 2.29 – Переглянути інформацію про відвідуваність учнів класу

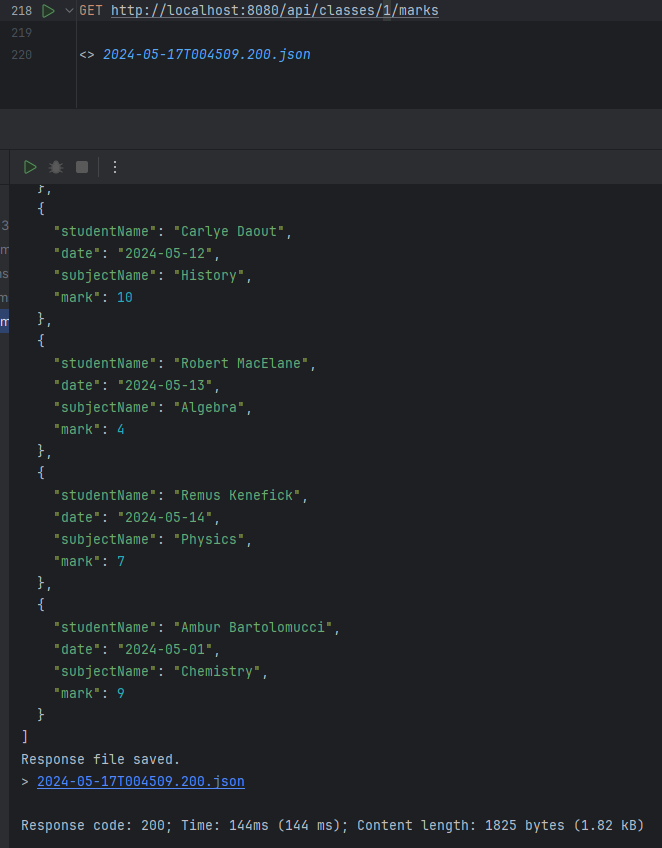


Рисунок 2.30 – Переглянути оцінки учнів класу



Рисунок 2.31 – Переглянути звіт успішності учня класу

# **3 ПІДКЛЮЧЕННЯ SWAGGER, СИСТЕМИ АВТОРИЗАЦІЇ З РОЛЯМИ, DOCKER COMPOSE ДЛЯ РОЗГОРТАННЯ ПРОЕКТУ**

## **3.1 Підключення системи авторизації OAuth2**

OAuth 2.0 - це протокол авторизації, який дозволяє користувачам давати дозвіл на доступ до своїх ресурсів без необхідності передачі свого пароля. У зв'язку з цим, цей протокол широко використовується для делегування доступу до ресурсів стороннім додаткам, таким як мобільні застосунки чи веб-сайти.

OAuth 2.0 базується на видачі "токенів доступу", які дозволяють стороннім додаткам отримувати доступ до обмеженого набору ресурсів від ім'я користувача. Крім того, OAuth 2.0 включає механізми для захисту цих токенів від несанкціонованого доступу та забезпечує можливість оновлення або скасування доступу в будь-який момент.

Для підключення системи авторизації OAuth2 скористаємось Auth0. Auth0- це платформа для управління ідентифікацією та автентифікацією користувачів у додатках. Auth0 пропонує рішення для реалізації безпеки та управління доступом, що дозволяє додаткам швидко та безпечно впроваджувати механізми авторизації та аутентифікації. Auth0 імплементує OAuth 2.0 як один із способів авторизації користувачів в додатках. Ось як Auth0 використовує OAuth 2.0:

* Auth0 дозволяє додаткам отримувати доступ до ресурсів в ім'я користувача, використовуючи різні типи Authorization Grants, такі як Authorization Code, Implicit, Password та інші. Ці гранти дозволяють додаткам отримувати доступ до захищених ресурсів після успішної аутентифікації користувача;
* після успішної аутентифікації Auth0 видає токени доступу (Access Token) та, за необхідності, токени оновлення (Refresh Token). Access Token представляє собою джерело авторизації для доступу до захищених ресурсів, тоді як Refresh Token дозволяє поновлювати токен доступу без повторної аутентифікації користувача;
* додатки можуть перевірити валідність отриманих токенів, використовуючи механізми перевірки підпису та підтвердження аутентичності токенів. Auth0 надає JWKS (JSON Web Key Set), який містить публічні ключі для перевірки підпису JWT токенів;
* Auth0 дозволяє налаштовувати дозволи для користувачів та ролі, які визначають, які ресурси та дії користувач може виконувати. Ці дозволи можуть бути включені в токени доступу, щоб додатки могли здійснювати контроль доступу на рівні ресурсів.

Для роботи з Auth0 необхідно створити на веб-сайті <https://auth0.com/> аккаунт. Потім необхідно створити новий додаток та API. В налаштуваннях додатка нам необхідні такі атрибути як Domain, Client ID та Client Server. Вони знадобляться для інтеграції Auth0 в нашу систему.

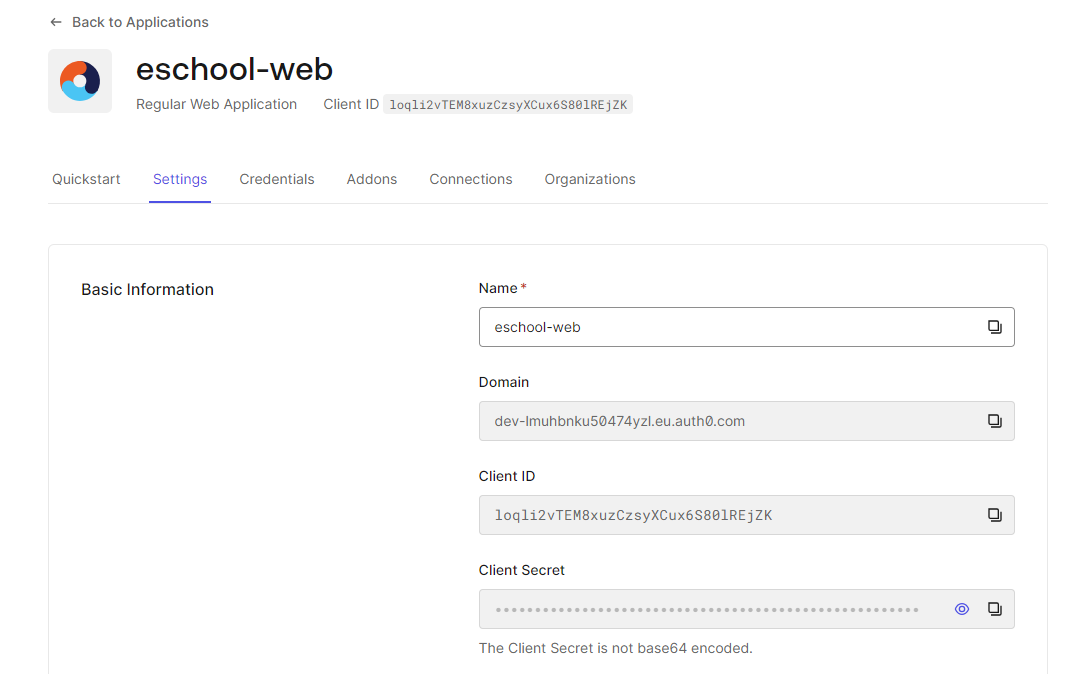


Рисунок 3.1 – Створення нового додатку в Auth0

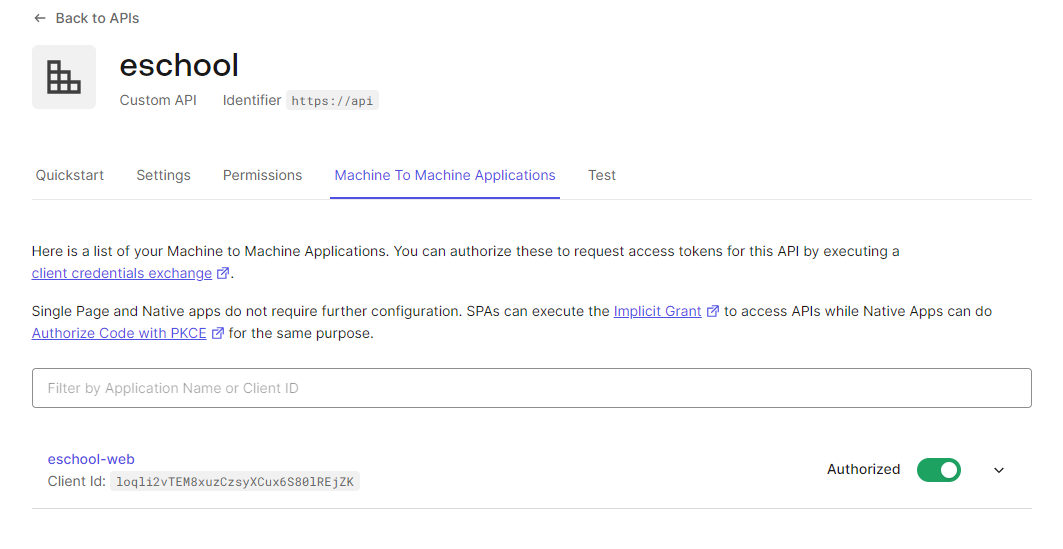


Рисунок 3.2 – Надання доступу до API

В API налаштуємо дозволи (permissons) та створимо ролі для наших користувачів. У системі будуть дві ролі: користувач, який матиме дозвіл для роботи з електронним журналом, та адміністратор, який матиме дозволи користувача, а також доступи до додавання/редагування/видалення інформації про вчителів та учнів.

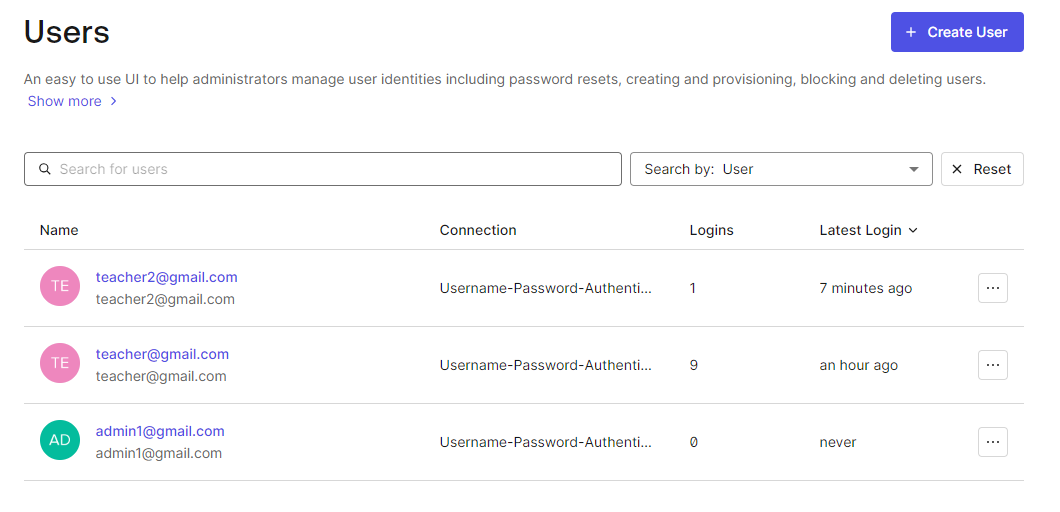


Рисунок 3.3 – Користувачі

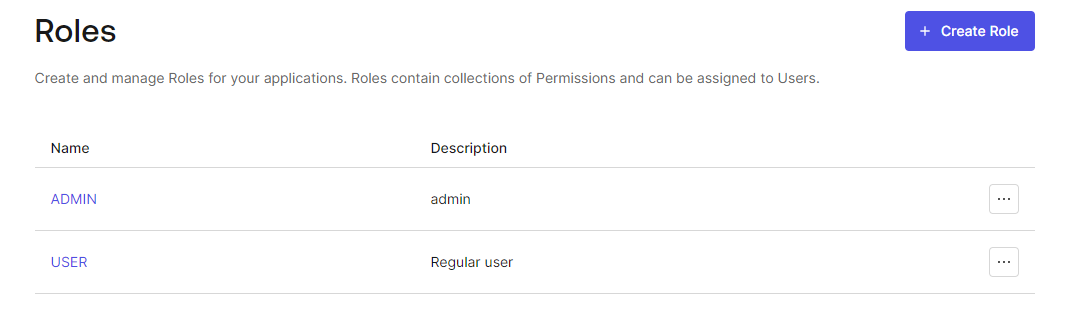


Рисунок 3.4 – Ролі користувачів

Лістинг файлу application.yml:

security:  
oauth2:  
client:  
registration:  
auth0:  
client-id: loqli2vTEM8xuzCzsyXCux6S80lREjZK  
client-secret: CYyZLrBENNIXxKcLembHg-\_8mYxobDy5R2EWhgl1HLSUggKO4iPMGW0OF1fxMg9y  
scope:  
 - openid  
 - profile  
 - email  
provider:  
auth0:  
issuer-uri: https://dev-lmuhbnku50474yzl.eu.auth0.com/  
resourceserver:  
jwt:  
issuer-uri: https://dev-lmuhbnku50474yzl.eu.auth0.com/

Щоб налаштувати додаток як сервер ресурсів і перевірити JWT (JSON Web Token - стандарт для створення компактних та автономних токенів доступу у форматі JSON) створимо клас, який буде надавати екземпляр SecurityFilterChain, і додамо анотацію @Configuration та @EnableWebSecurity.

Лістинг класу SecurityConfig:

@Configuration  
@EnableWebSecurity  
public class SecurityConfig {  
  
@Value("${spring.security.oauth2.resourceserver.jwt.issuer-uri}")  
private String issuerUri;  
  
@Bean  
public SecurityFilterChain filterChain(HttpSecurity http) throws Exception {  
 http  
 .authorizeHttpRequests(authorize -> authorize  
 .anyRequest().authenticated()  
 )  
 .oauth2Login(*withDefaults*())  
 .cors(*withDefaults*())  
 .oauth2ResourceServer(oauth2 -> oauth2  
 .jwt(*withDefaults*())  
 );  
return http.build();  
 }  
  
@Bean  
public JwtDecoder jwtDecoder() {  
return JwtDecoders.*fromIssuerLocation*(issuerUri);  
 }  
}

Додамо до методів контролерів API анотацію, яка заборонить доступ звичайним користувачам до цих методів.

@PreAuthorize("hasAuthority('ROLE\_ADMIN')")

Також додамо контролер, який буде перенаправляти користувачів після авторизації на сторінку з документацією. Запустимо проект і перевіримо роботу. При запуску проекта відкривається сторінка з логіном.

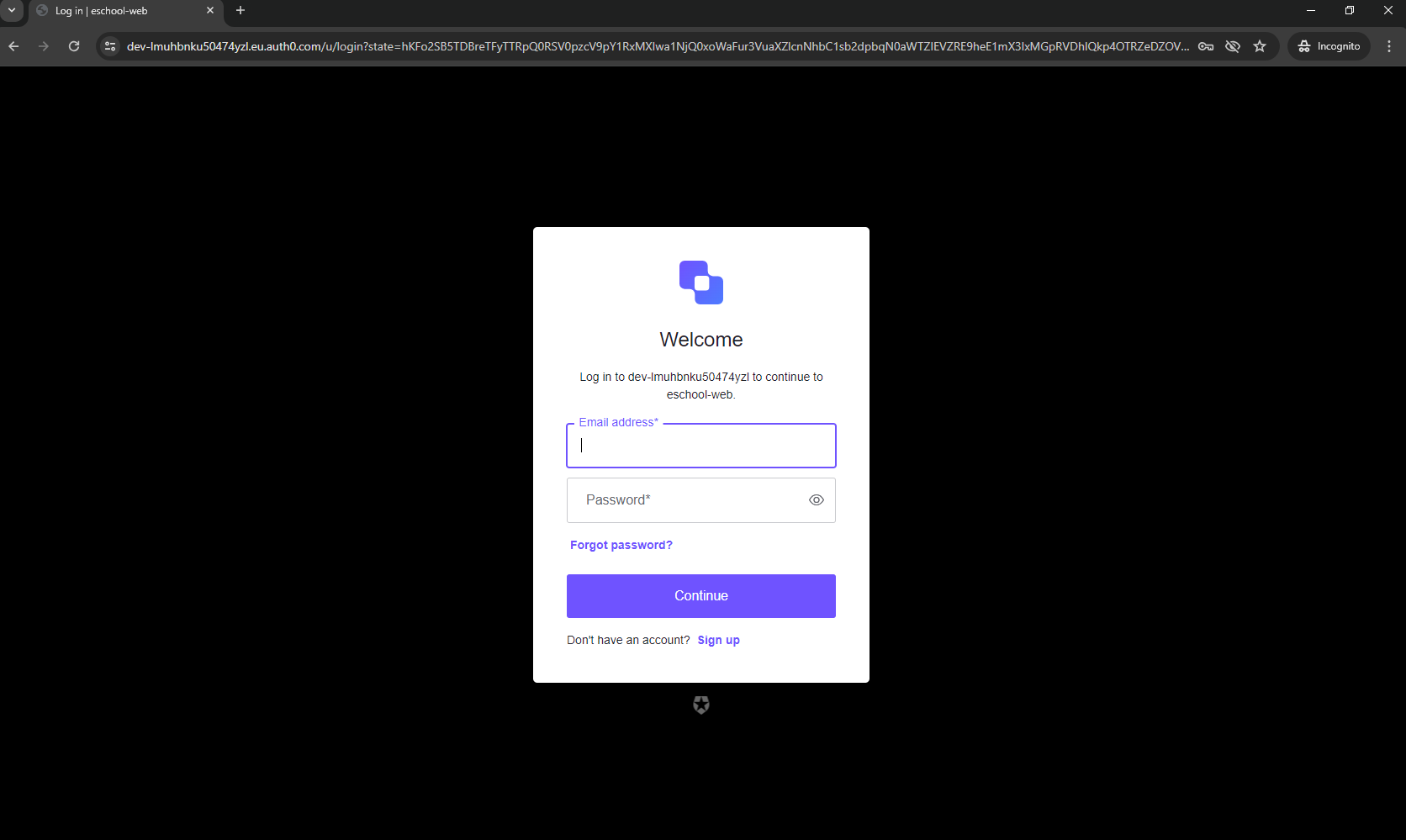


Рисунок 3.5 – Екран входу в систему

Якщо ввести неправильні дані (що не відповідають коректним даним користувача), ввійти в систему не вийде.

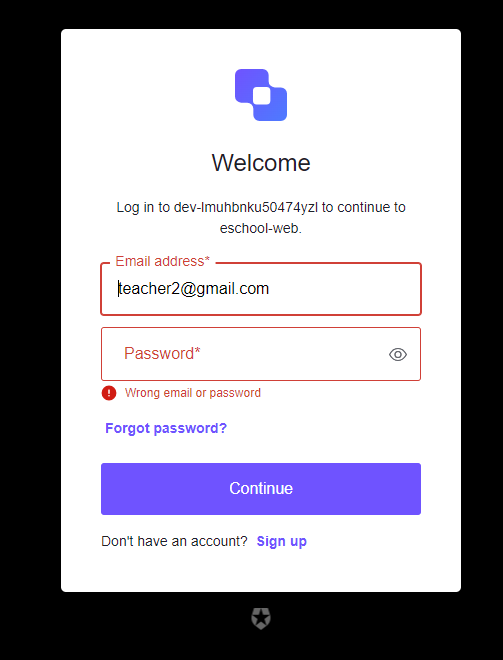


Рисунок 3.6 – Неправильні дані для входу

Якщо дані правильні, Auth0 питає у користувача, чи точно він хоче надати свої дані додатку. Якщо так, відбувається авторизація і користувача перенаправляє на сторінку з документацією.

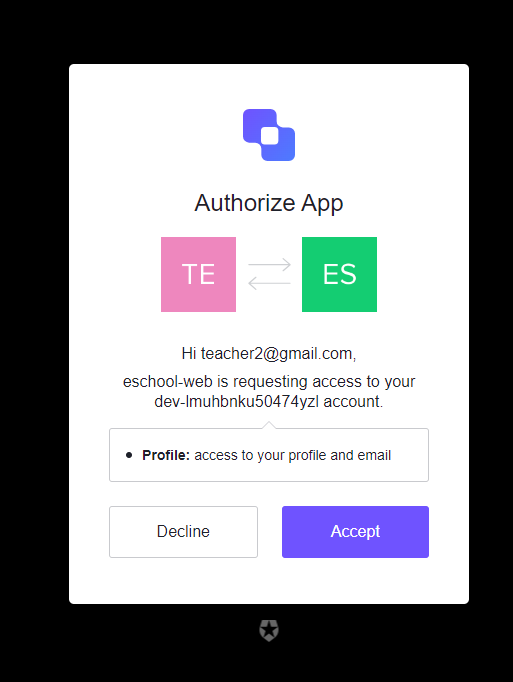


Рисунок 3.7 – Уточнення щодо надання даних

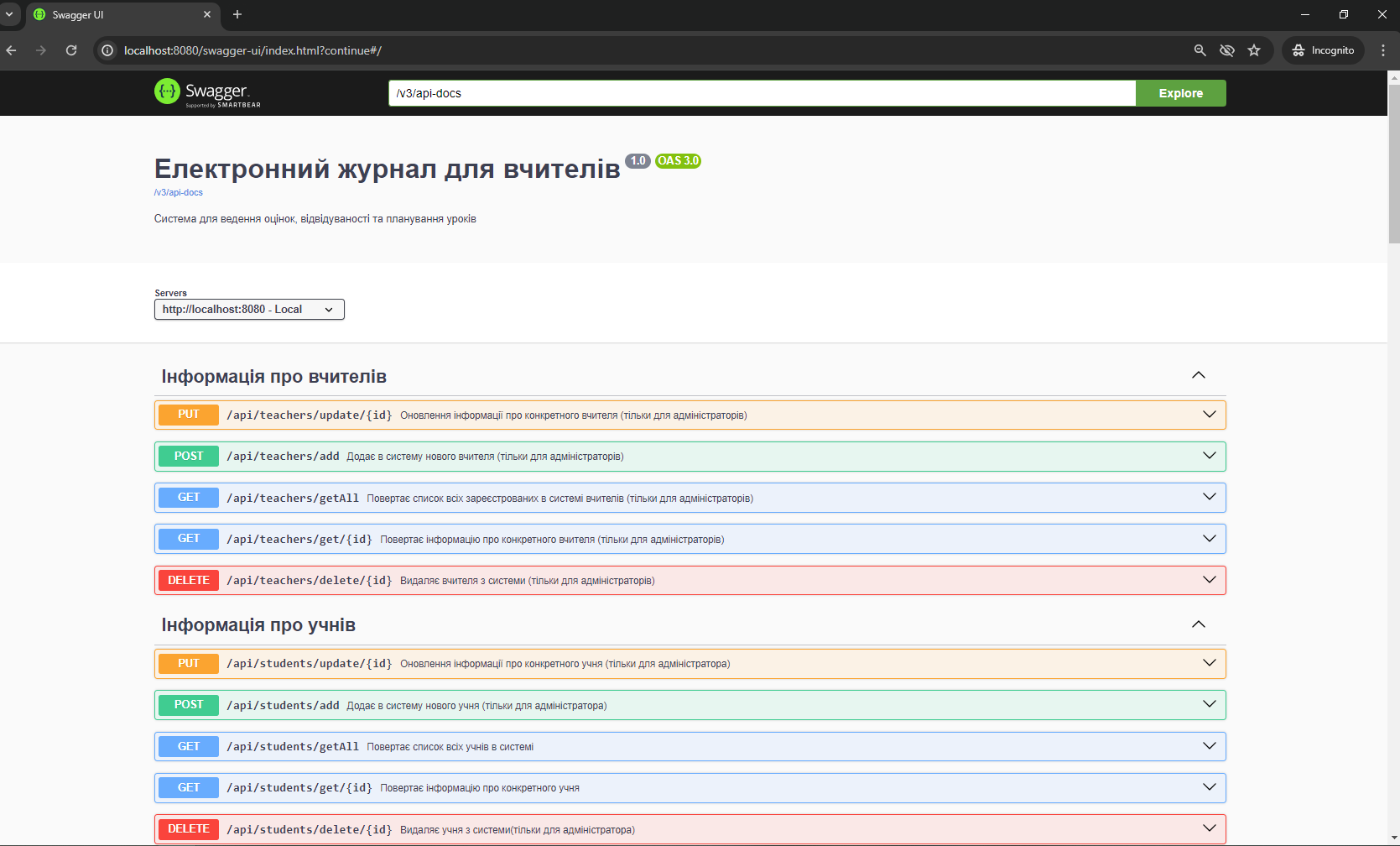


Рисунок 3.8 – Екран після авторизації користувача

**3.2 Підключення Swagger UI з детальним описом REST API**

Swagger - це набір інструментів для створення, документування та використання API. Основна мета Swagger - полегшити розробку та спілкування між розробниками та іншими учасниками проекту щодо використання API.

Swagger UI – це інтерактивний веб-інтерфейс, що автоматично генерує документацію API на основі специфікації OpenAPI. З Swagger UI можна переглядати, тестувати та спілкуватися з вашим API безпосередньо в браузері.

Для підключення Swagger UI в наш проект використаємо OpenApi: додамо відповідну залежність та створимо конфігураційний клас.

Лістинг класу SwaggerConfig:

@OpenAPIDefinition (  
 info = @Info(  
 description = "Система для ведення оцінок, відвідуваності та планування уроків",  
 title = "Електронний журнал для вчителів",  
 version = "1.0"  
),  
 servers = {  
@Server(  
 description = "Local",  
 url = "http://localhost:8080"  
)  
 }  
)  
public class SwaggerConfig {  
@Bean  
public GroupedOpenApi publicApi() {  
return GroupedOpenApi.*builder*()  
 .group("api")  
 .pathsToMatch("/api/\*\*")  
 .build();  
 }  
}

Додамо до всіх контролерів анотації OpenApi, щоб надати детальний опис.

Наведемо приклад анотації. Всі інші контролери проанатуємо за аналогією.

Лістинг коду.

@Operation(  
 description = "Оновити інформацію про вчителя",  
 summary = "Оновлення інформації про конкретного вчителя (тільки для адміністраторів)"  
)  
@PreAuthorize("hasAuthority('ROLE\_ADMIN')")  
@PutMapping("/update/{id}")  
public ResponseEntity<TeacherDto>updateTeacher(@PathVariable("id") Long teacherId,  
@RequestBody TeacherDto updatedTeacher) {  
 TeacherDto teacherDto = teacherService.updateTeacher(teacherId, updatedTeacher);  
return ResponseEntity.*ok*(teacherDto);  
}

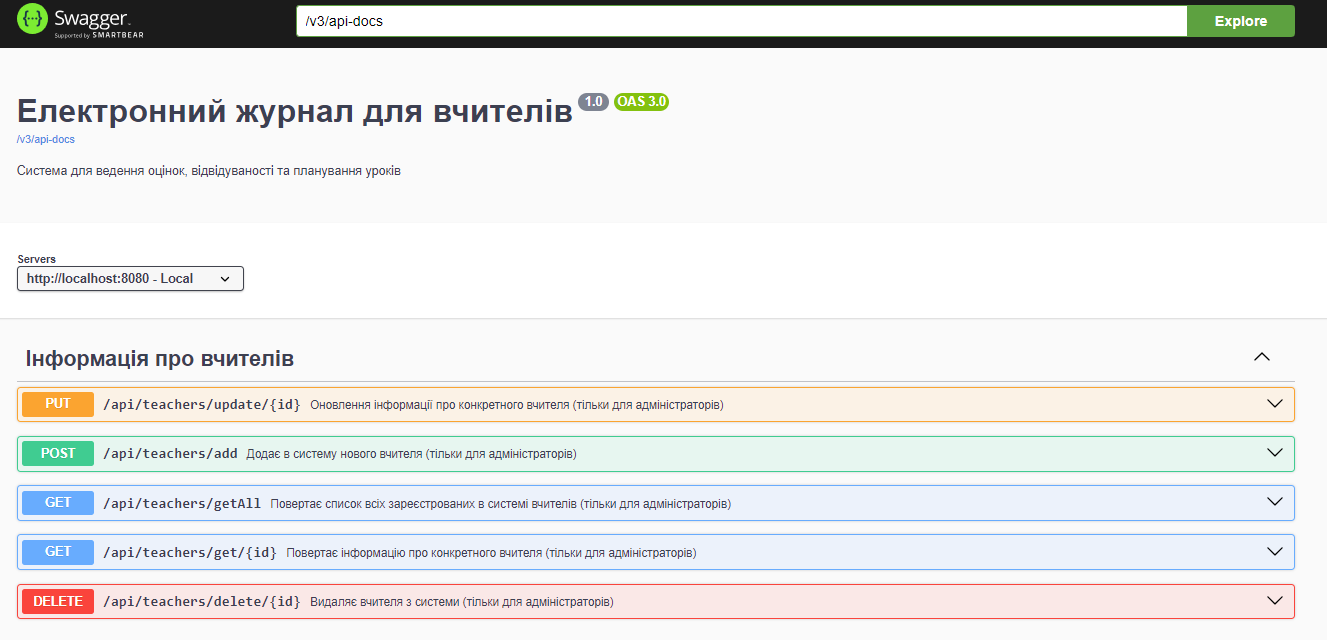


Рисунок 3.9 – Документація блоку інформації про вчителів

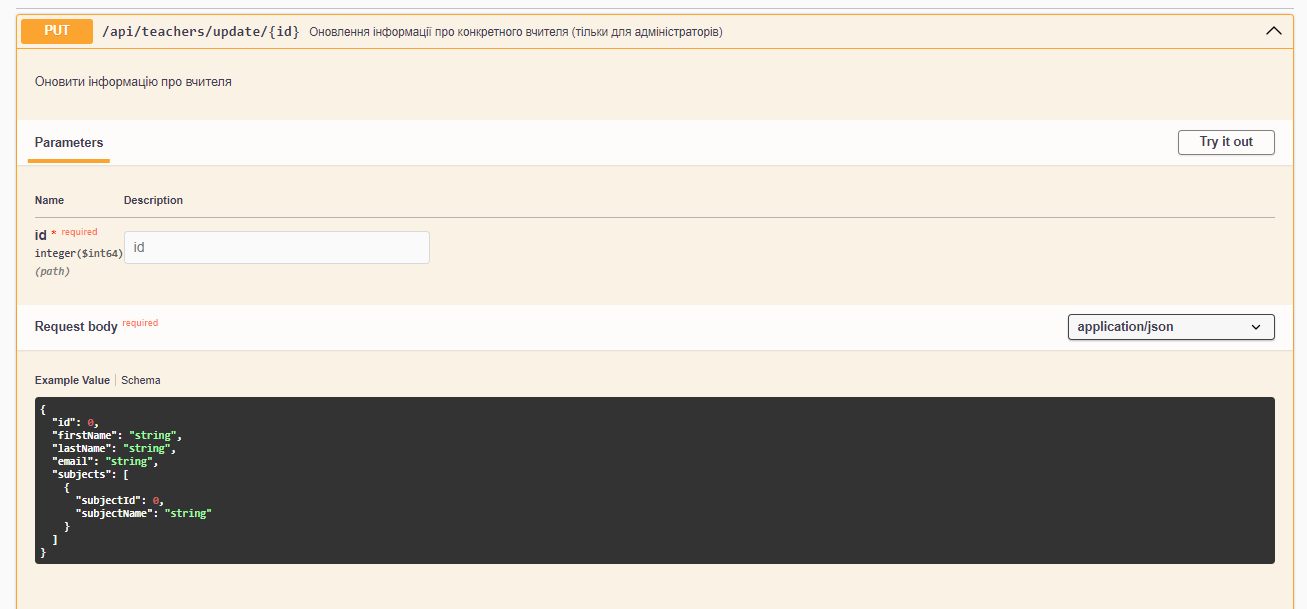
****

Рисунок 3.10 – Документація/api/teachers/update/{id}

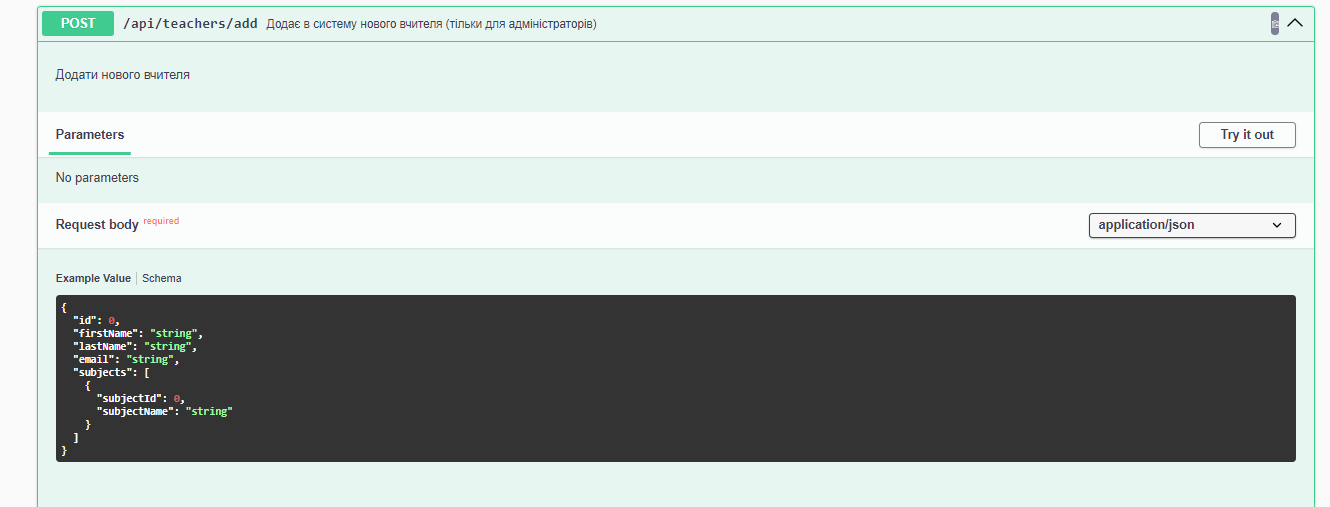
****

Рисунок 3.11 – Документація/api/teachers/add

****

Рисунок 3.12 – Документація/api/teachers/getAll

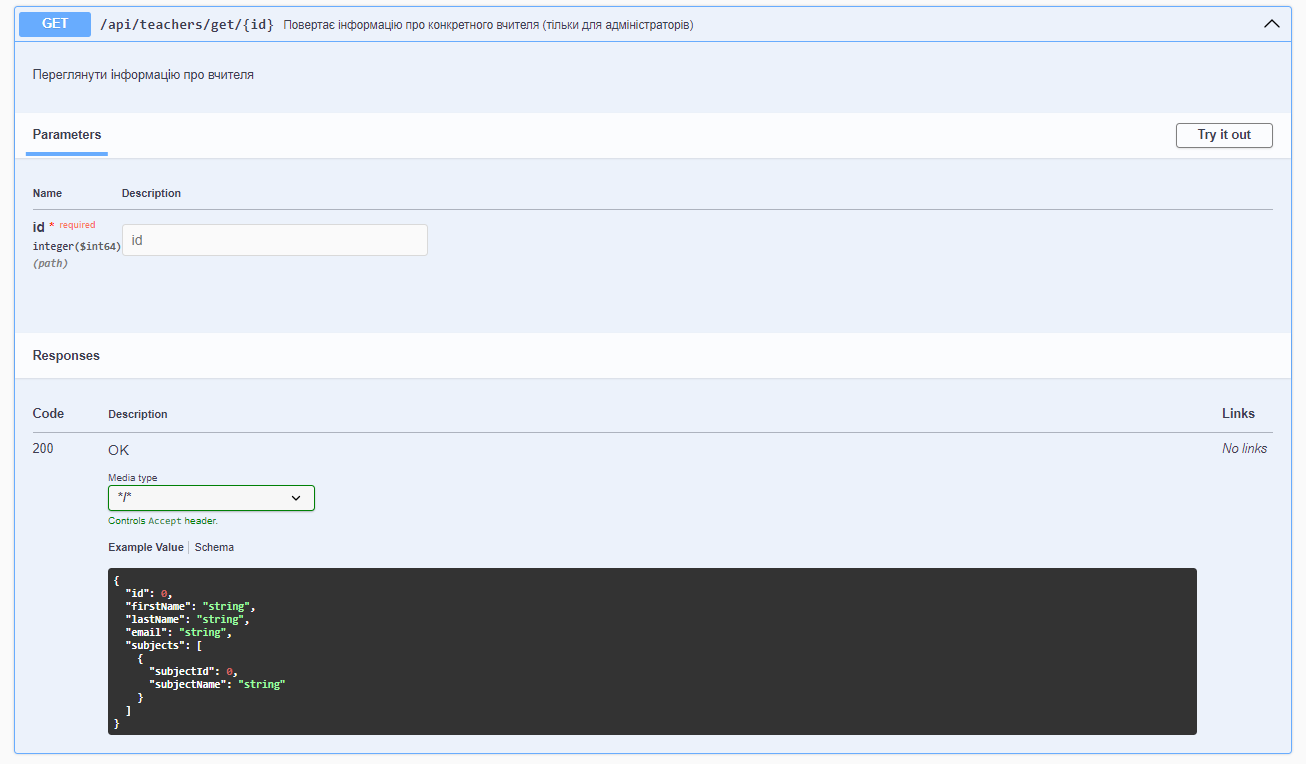
****

Рисунок 3.13 – Документація/api/teachers/get/{id}

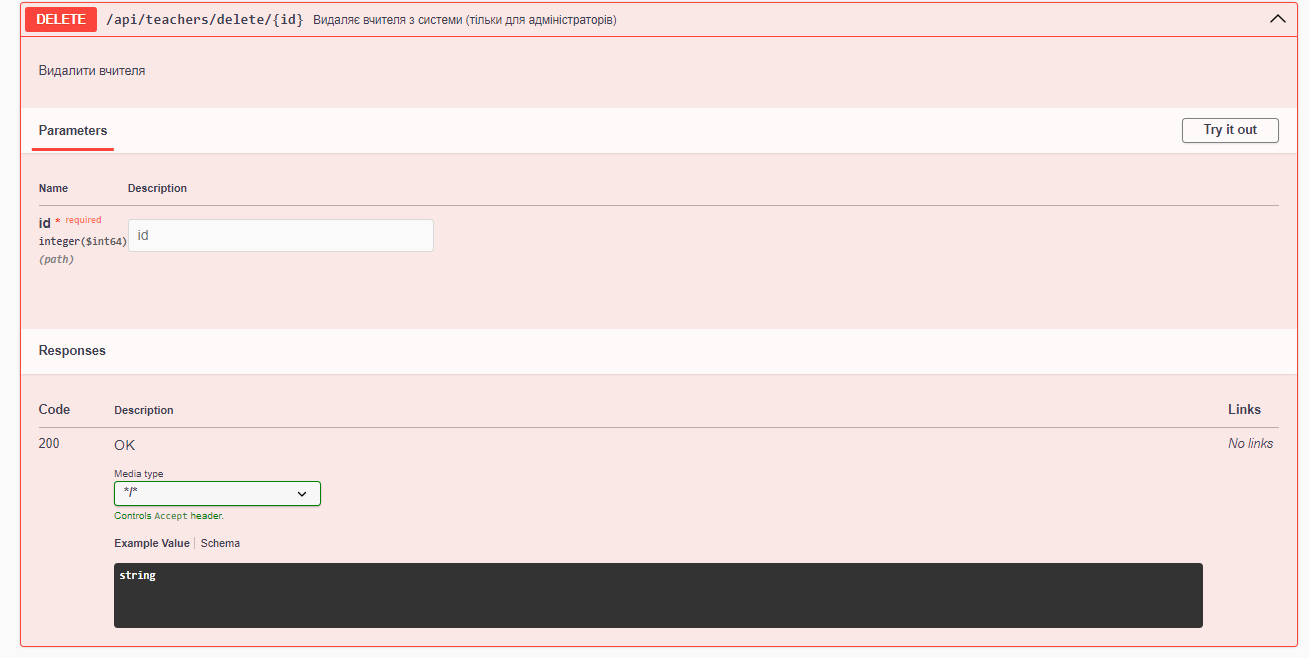
****

Рисунок 3.14 – Документація/api/teachers/delete/{id}

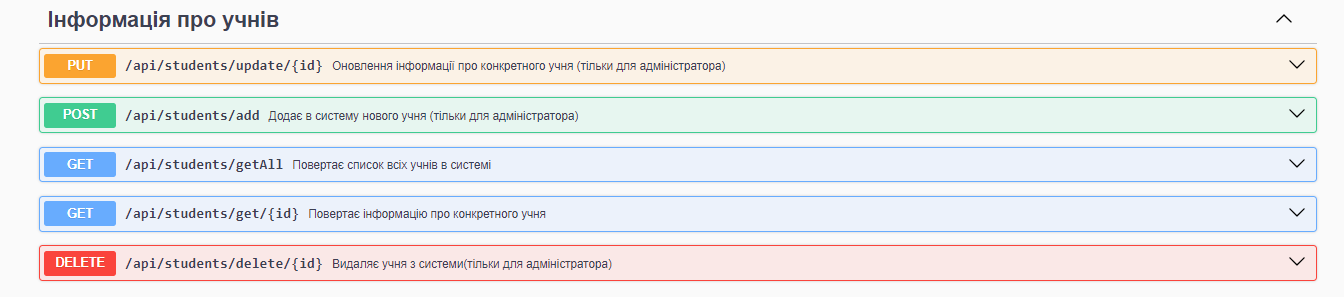
****

Рисунок 3.15 – Документація блоку інформації про учнів



Рисунок 3.16 – Документація/api/students/update/{id}

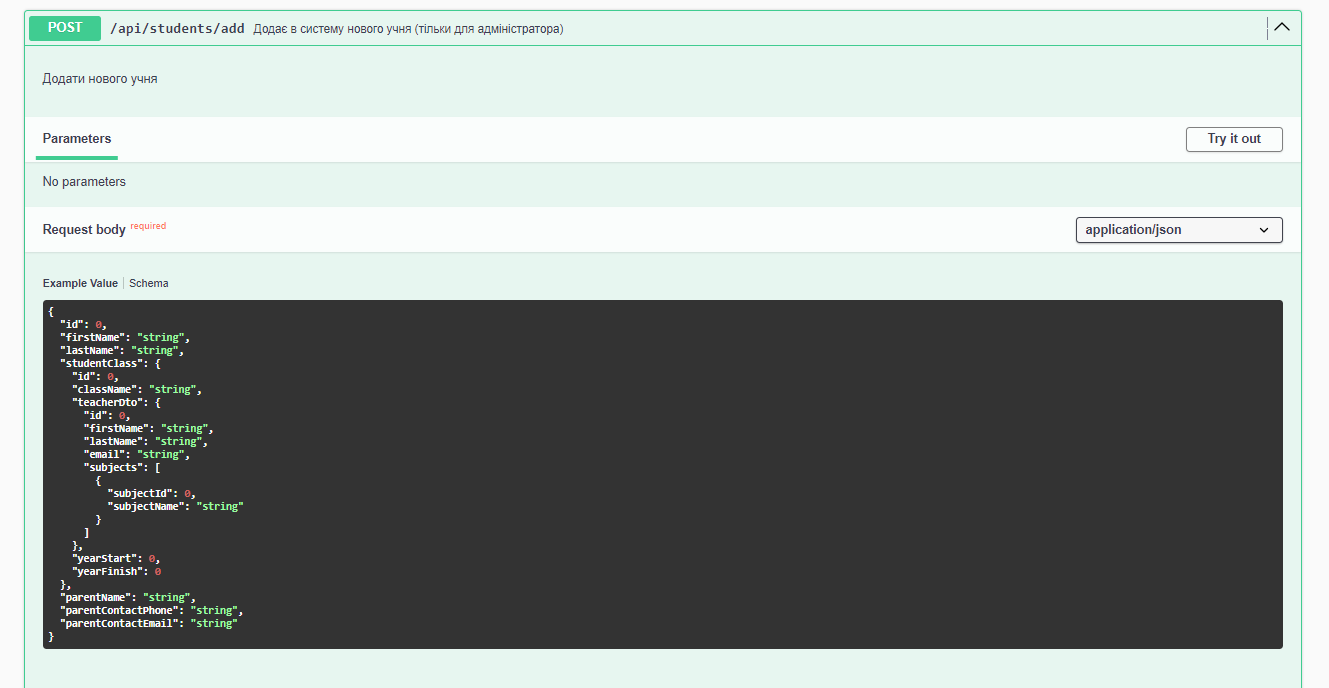


Рисунок 3.17 – Документація/api/students/add

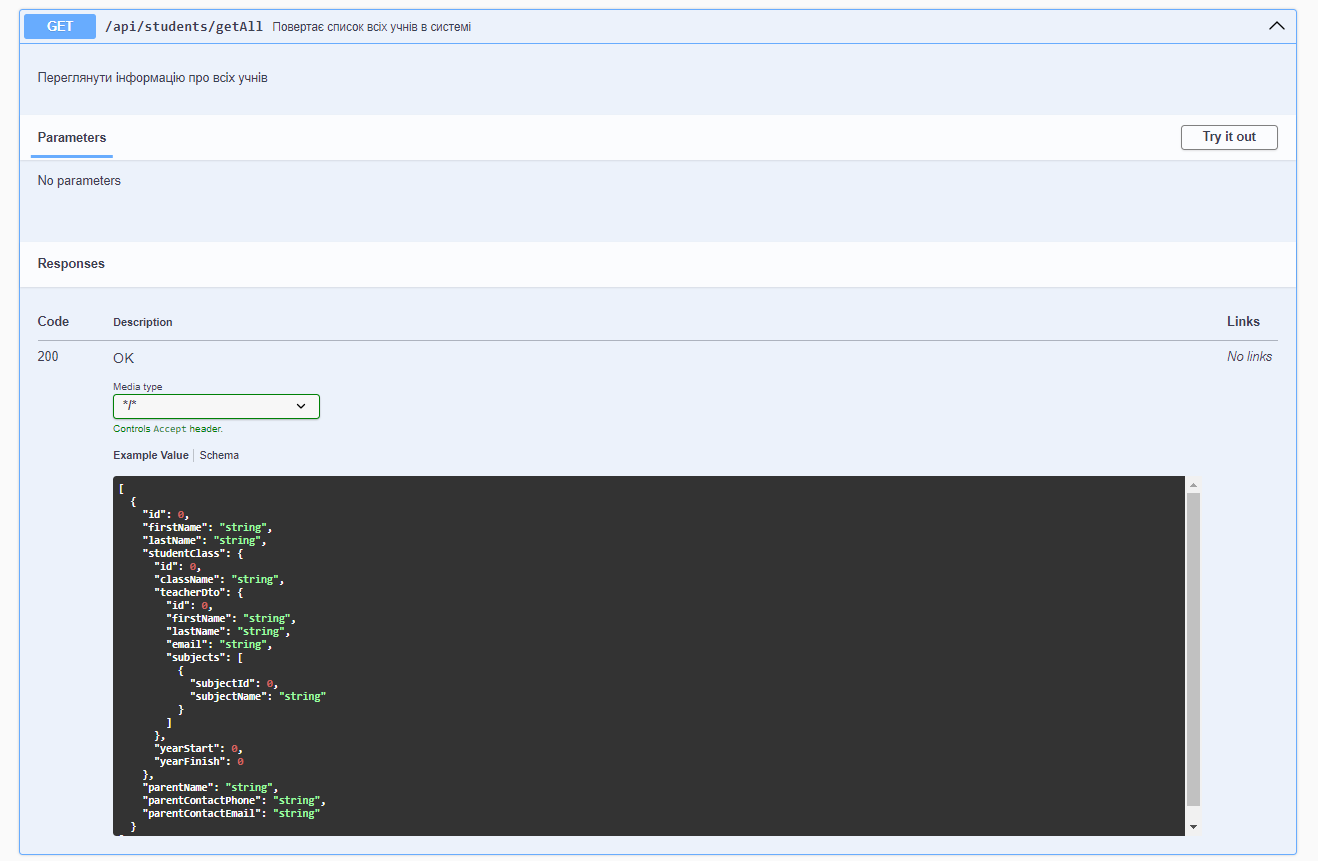


Рисунок 3.18 – Документація/api/students/add

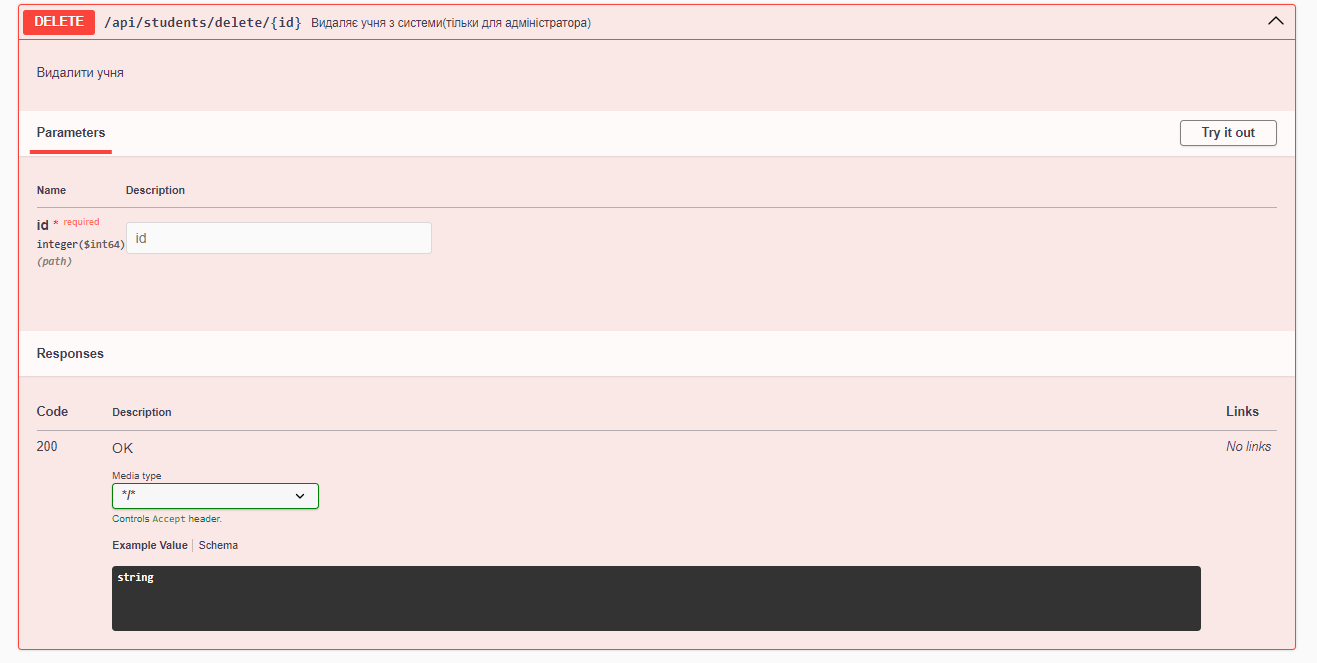


Рисунок 3.19 – Документація/api/students/delete/{id}

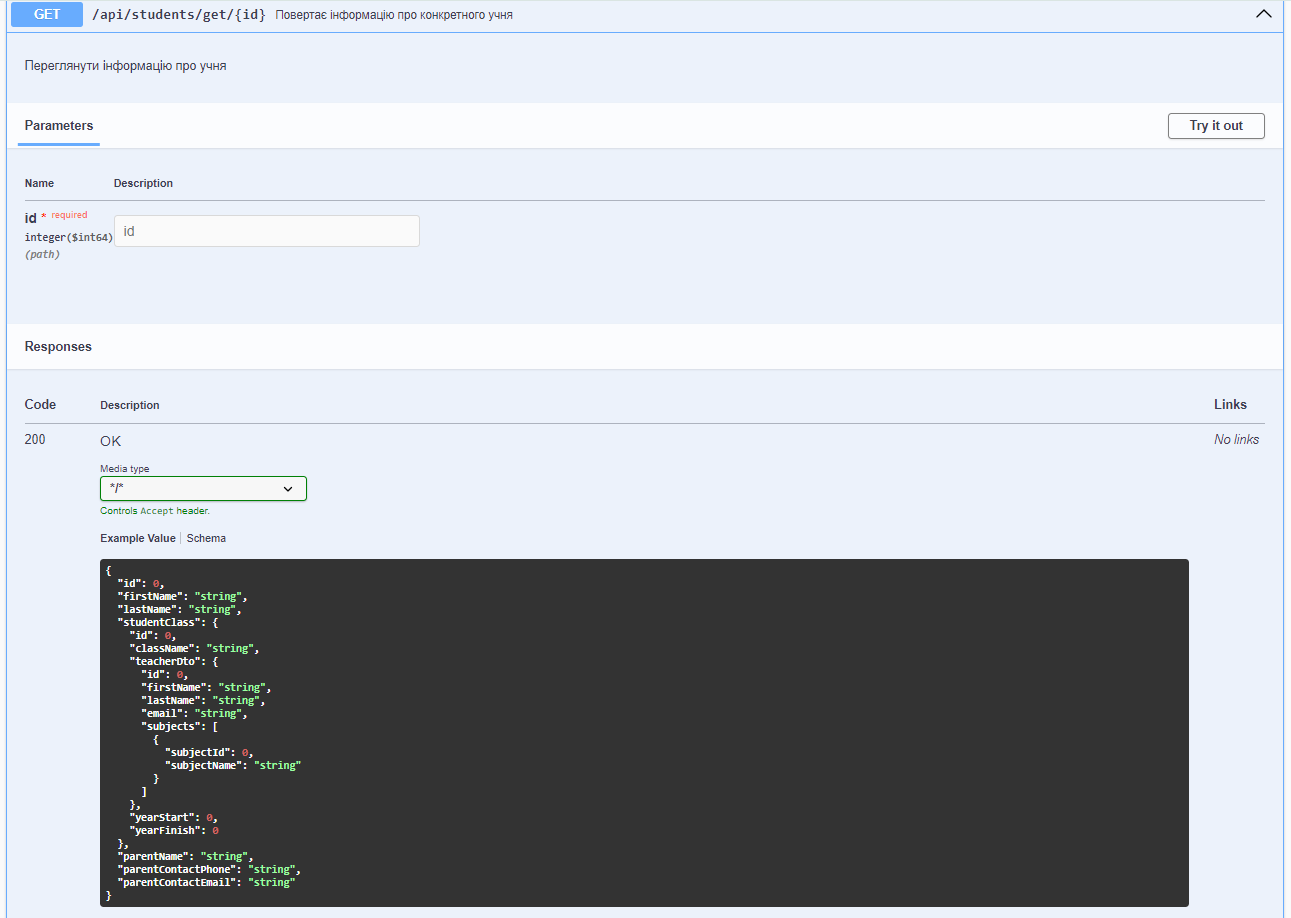


Рисунок 3.20 – Документація/api/students/get/{id}

****

Рисунок 3.21 – Документація блоку керування матеріалами для уроків

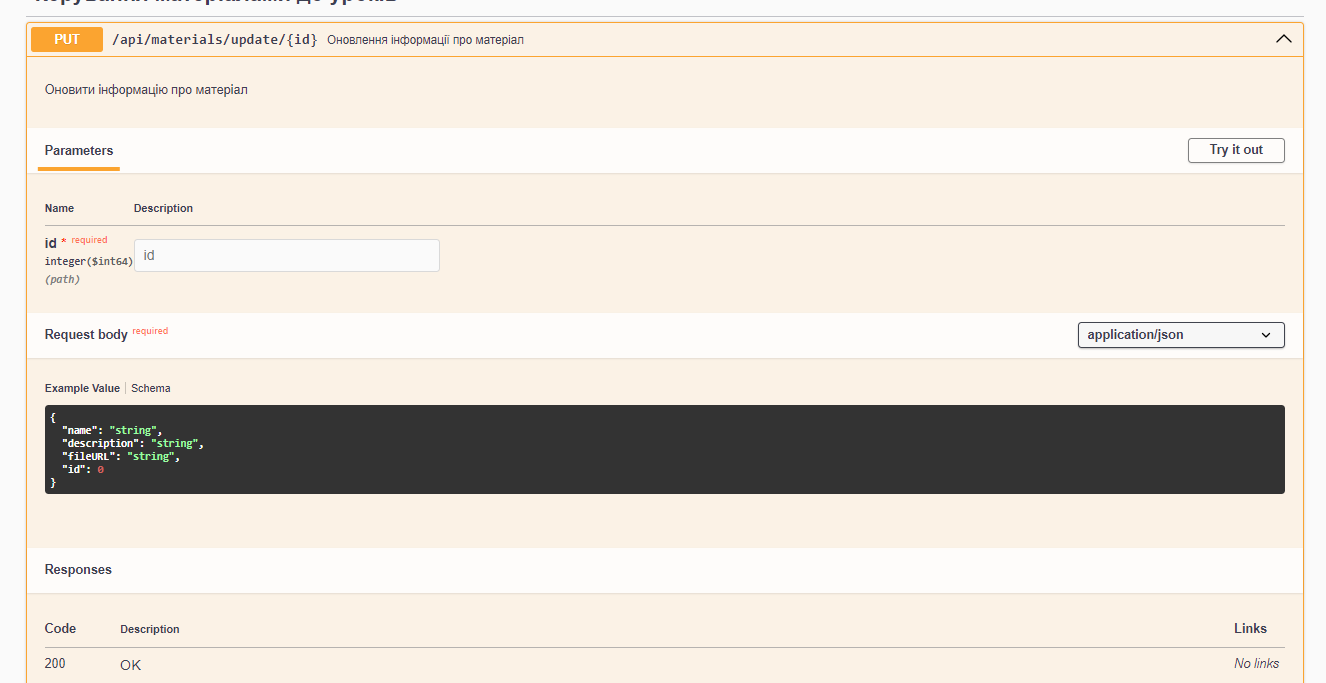
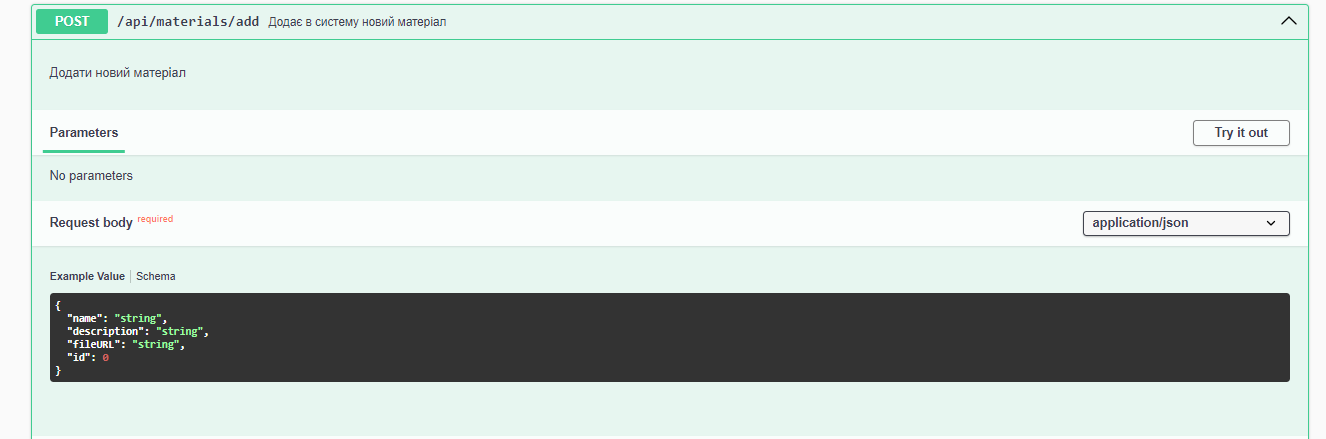


Рисунок 3.22 – Документація/api/materials/update/{id}

 Рисунок 3.23 – Документація/api/ materials/add

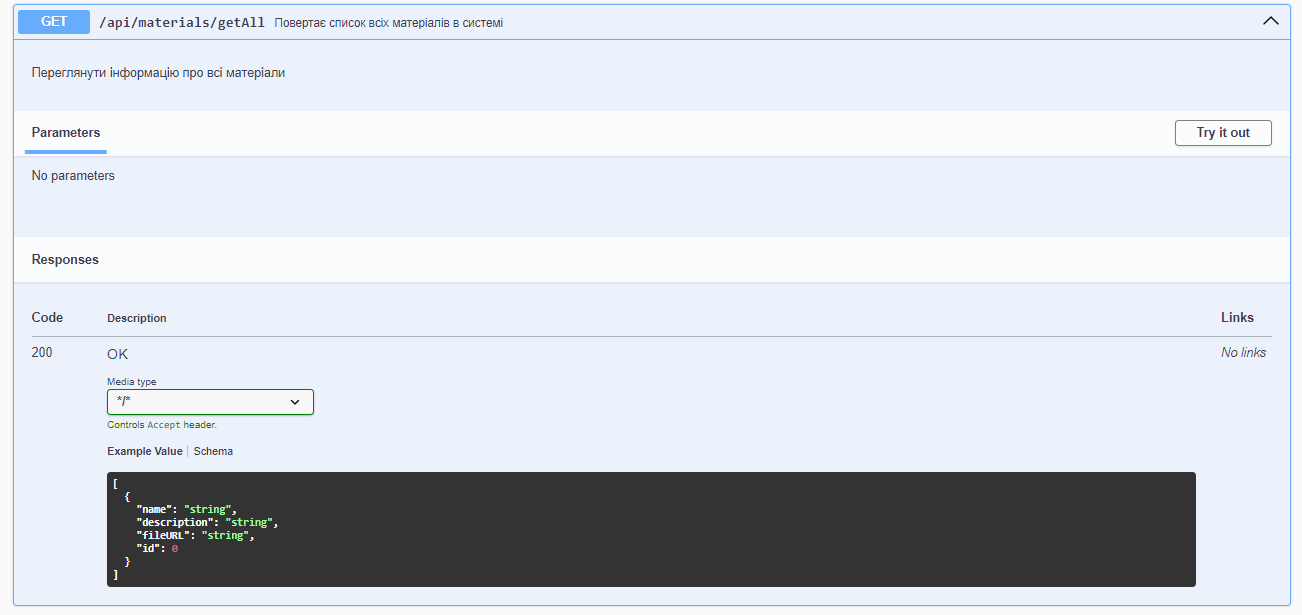


Рисунок 3.24 – Документація/api/ materials/getAll

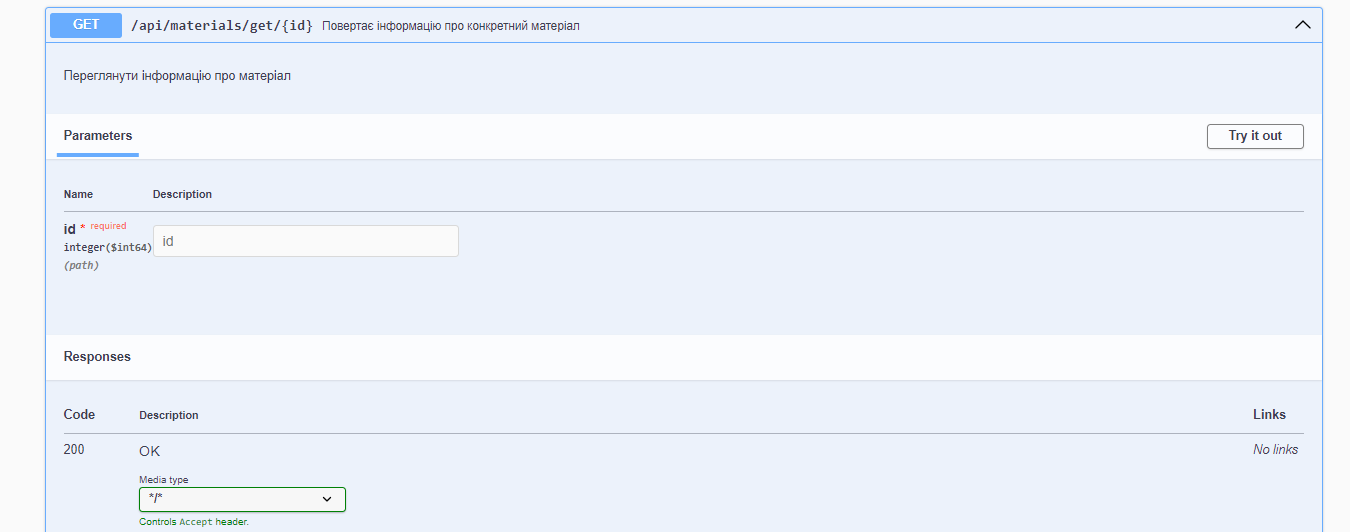


Рисунок 3.25 – Документація/api/ materials/get/{id}

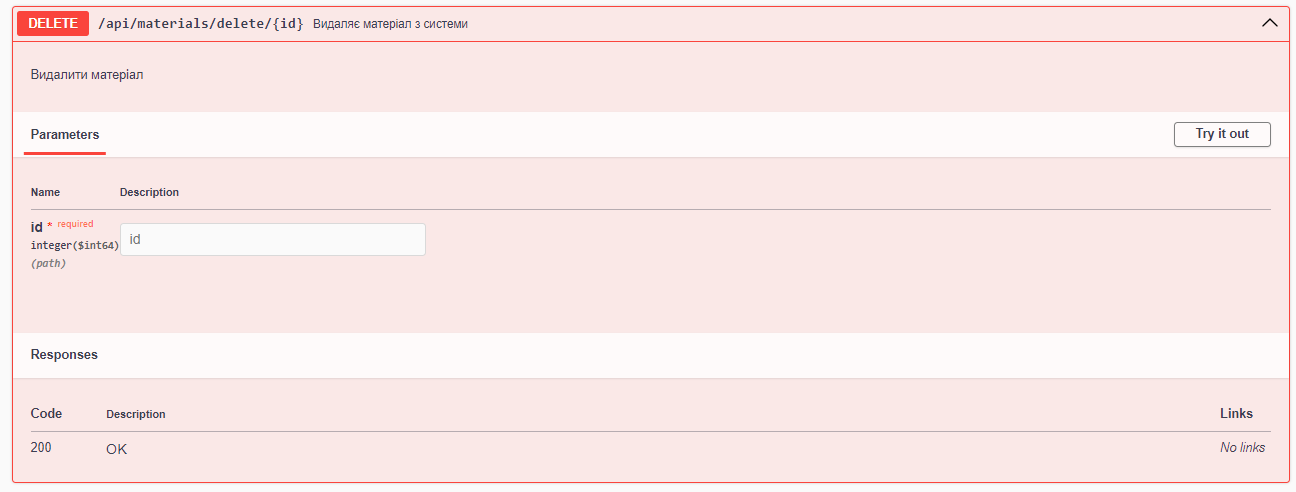


Рисунок 3.26 – Документація/api/ materials/delete/{id}

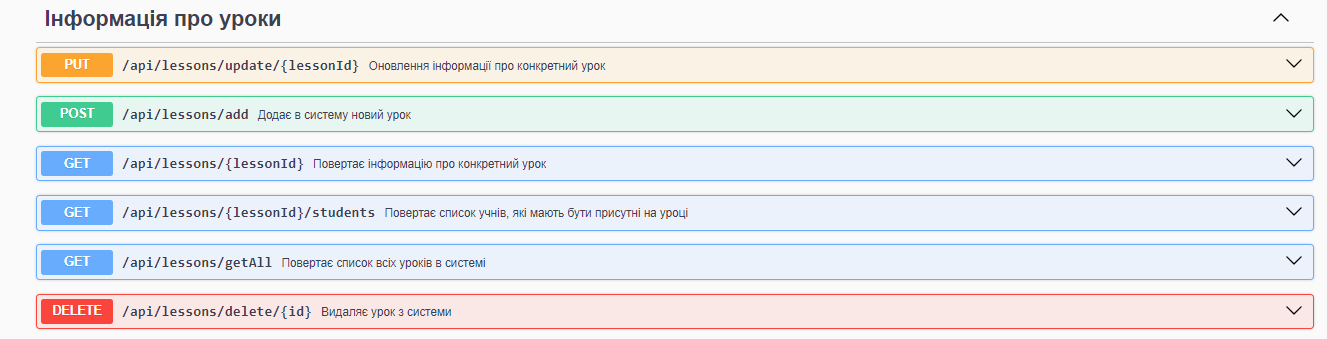
****

Рисунок 3.27 – Документація блоку інформації про уроки

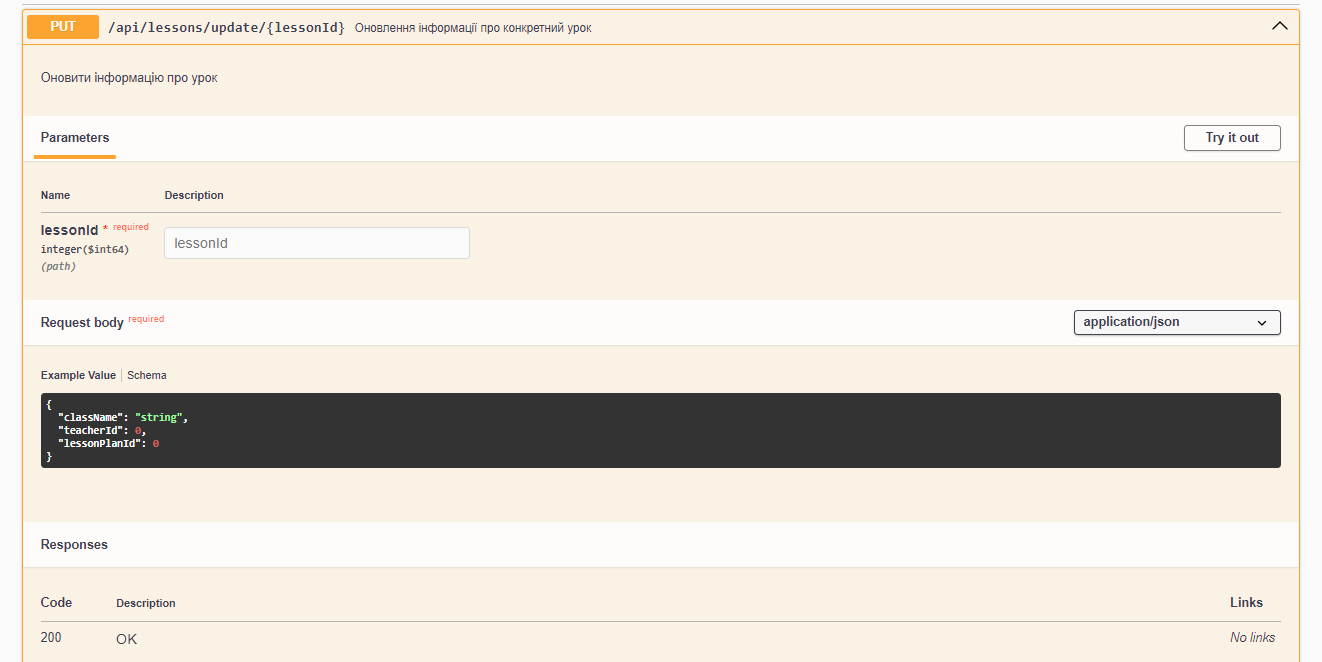


Рисунок 3.28 – Документація/api/lessons/update/{lessonId}



Рисунок 3.29 – Документація/api/ lessons/add

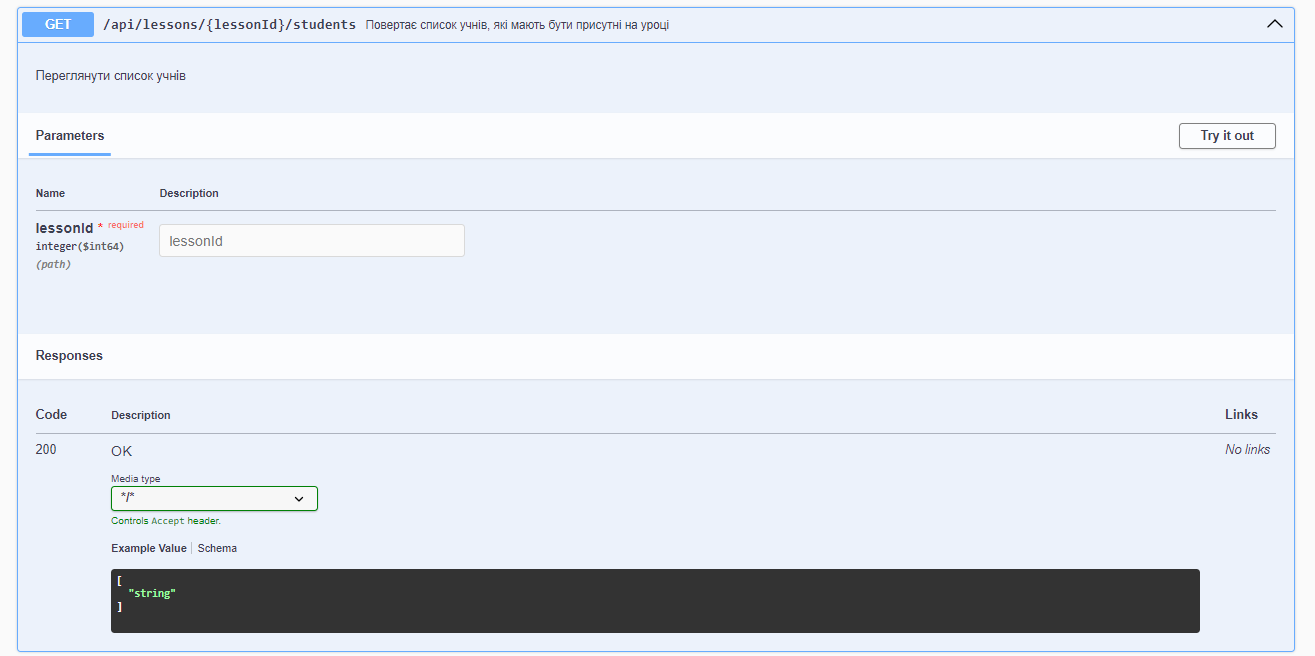


Рисунок 3.30 – Документація/api/ lessons/{lessonId}/students

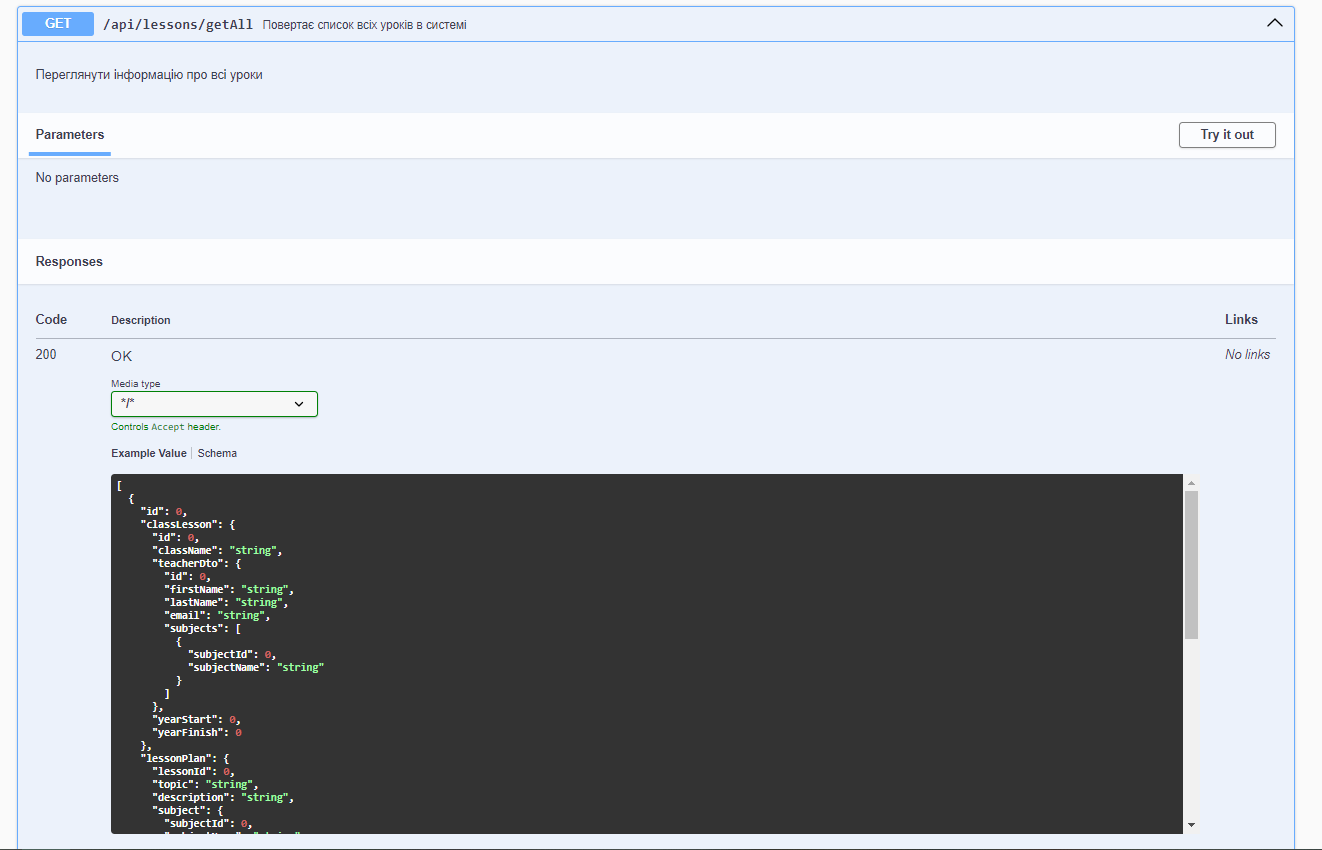


Рисунок 3.31 – Документація/api/ lessons/getAll



Рисунок 3.32 – Документація/api/ lessons/delete/{id}

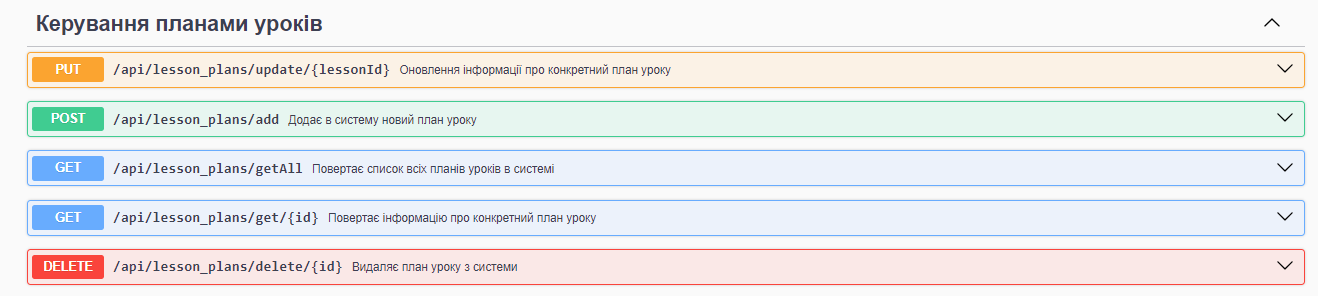
****

Рисунок 3.33 – Документація блоку керування планами уроків

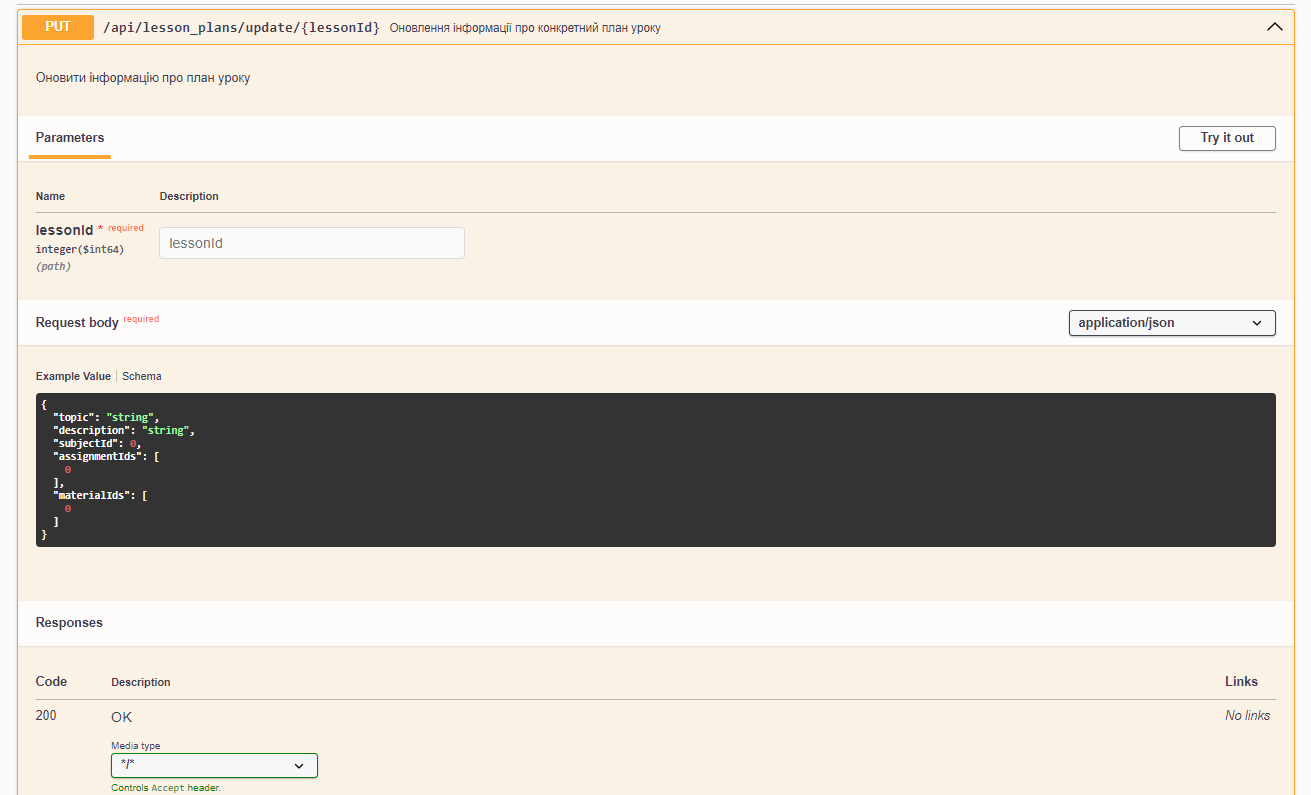
****

Рисунок 3.34 – Документація/api/ lessons\_plans/update/{id}

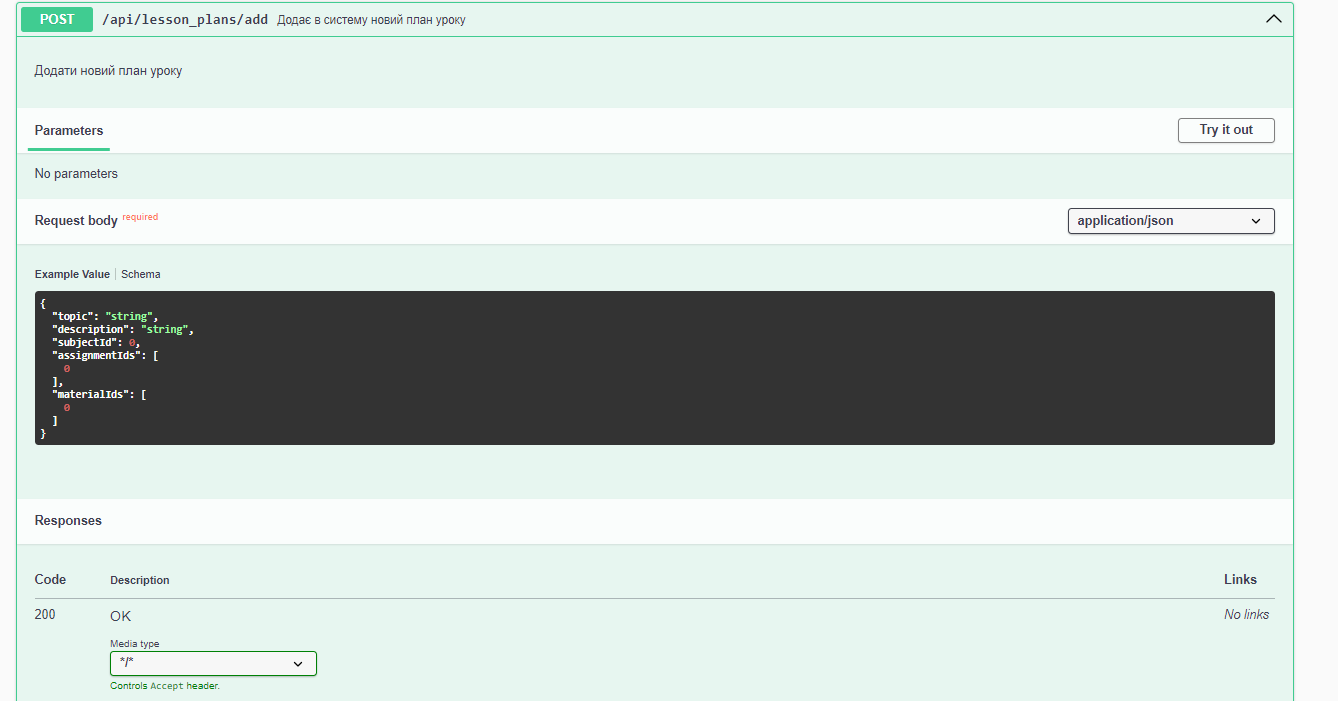
****

Рисунок 3.35 – Документація/api/ lessons\_plans/add

****

Рисунок 3.36 – Документація/api/ lessons\_plans/getAll

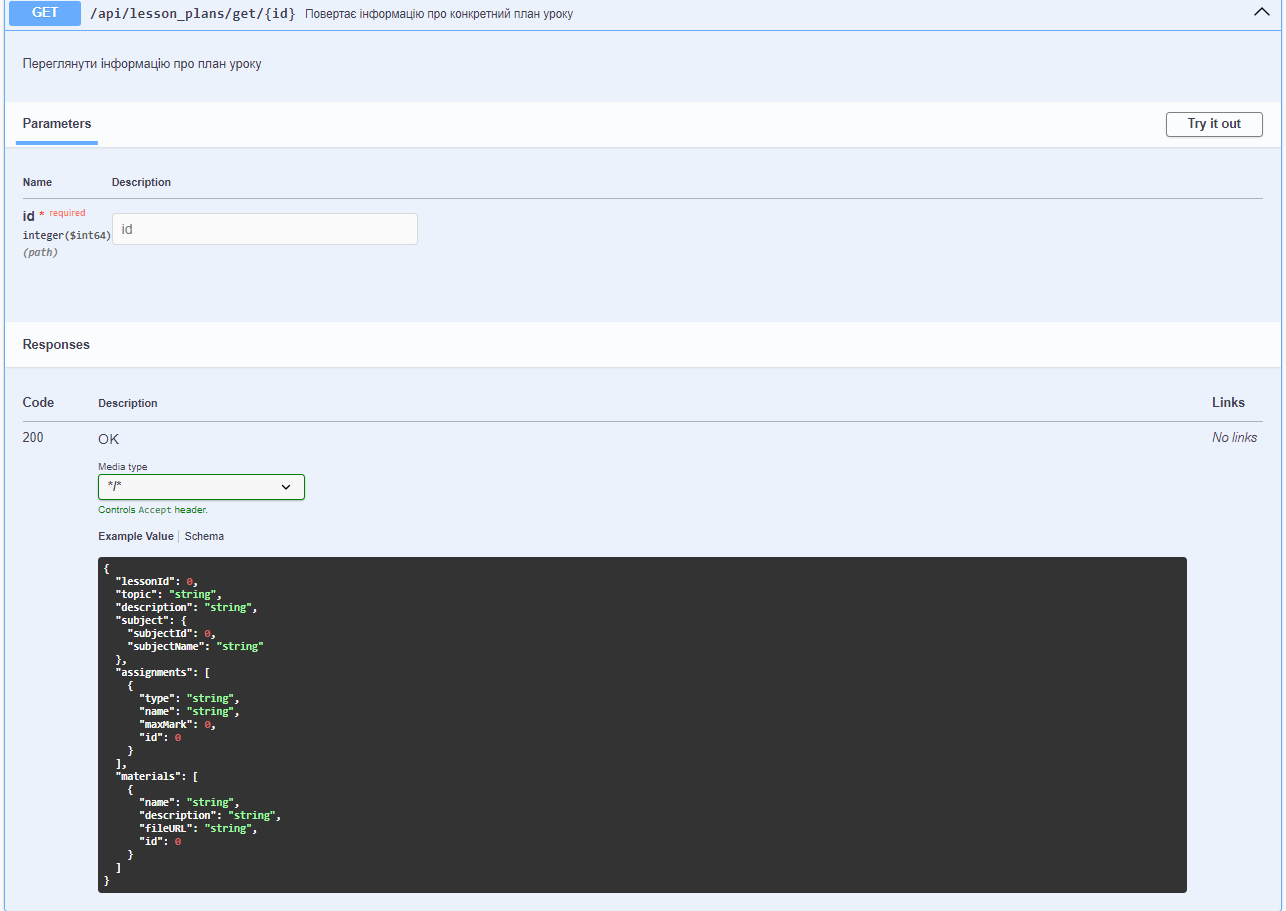
****

Рисунок 3.37 – Документація/api/ lessons\_plans/get/{id}

****

Рисунок 3.38 – Документація/api/ lessons\_plans/delete/{id}

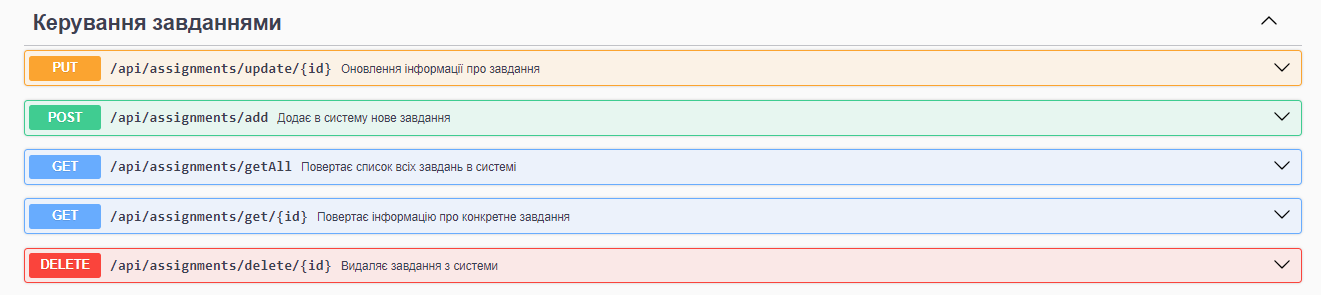
****

Рисунок 3.39 – Документація блоку керування завданнями

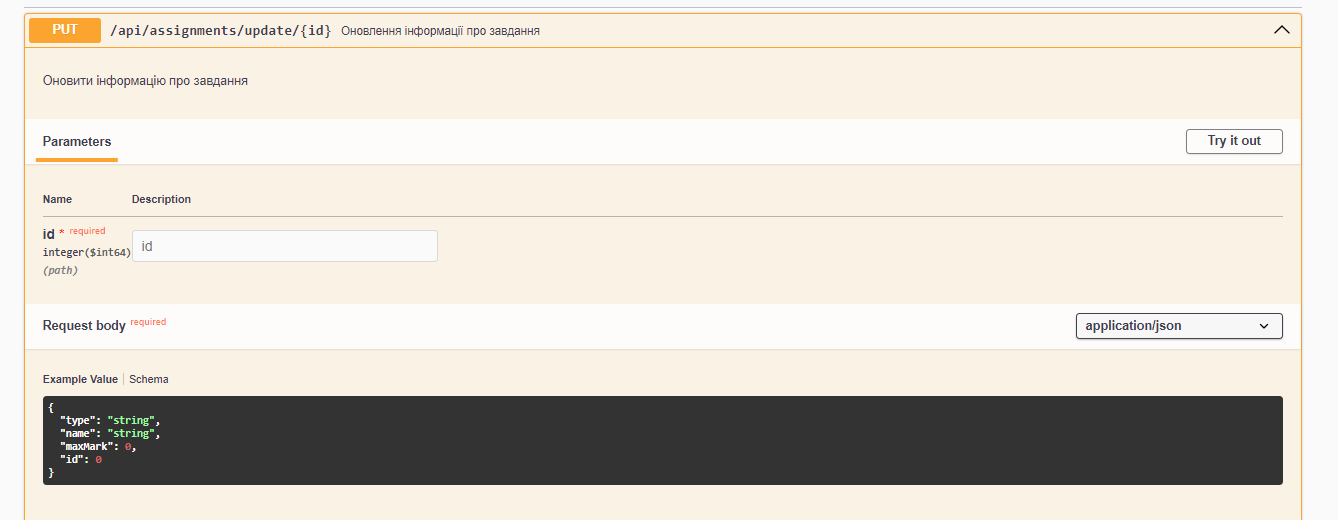
****

Рисунок 3.40 – Документація/api/ assignments/update/{id}

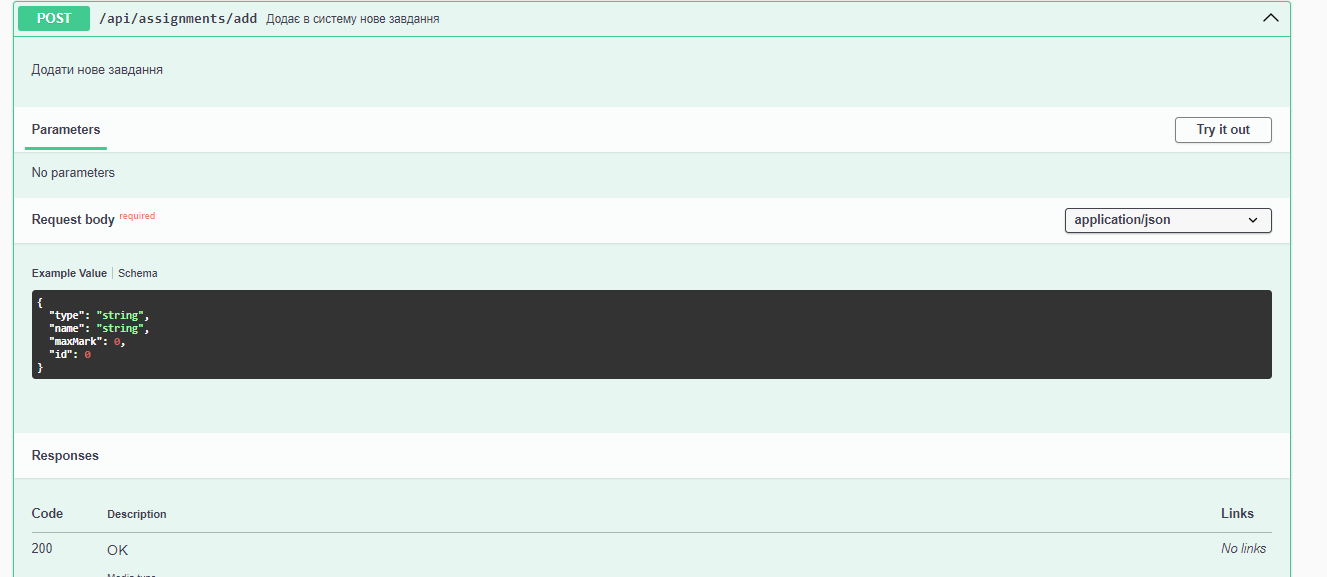
****

Рисунок 3.41 – Документація/api/ assignments/add

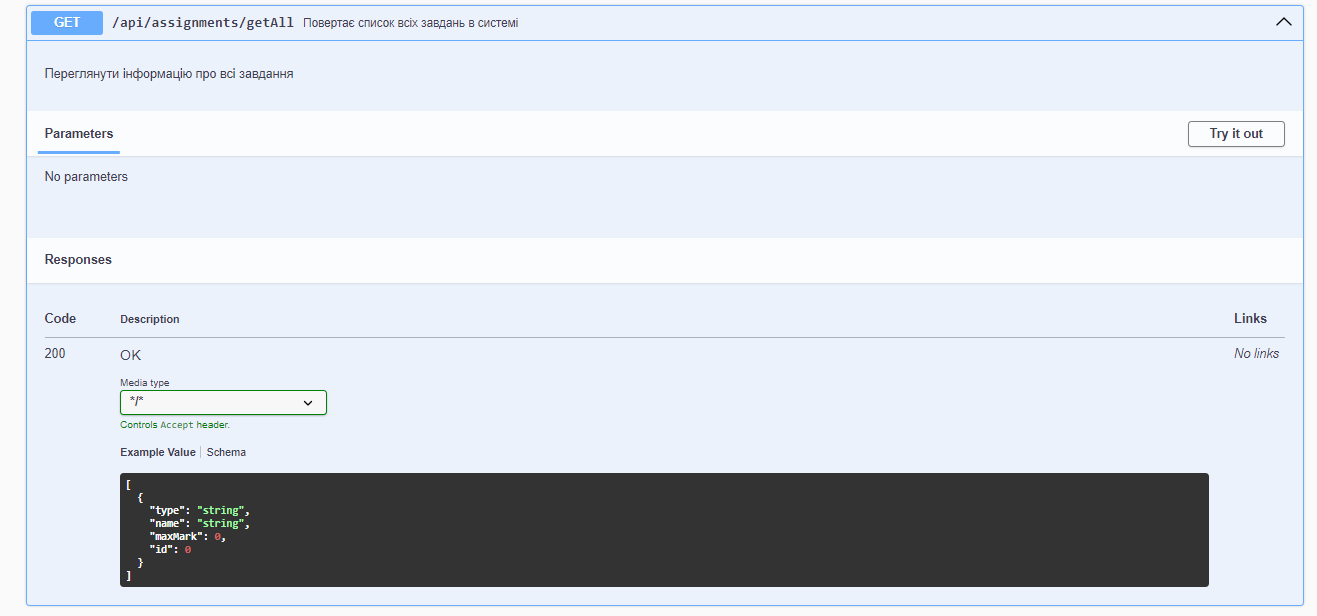
****

Рисунок 3.42 – Документація/api/ assignments/getAll

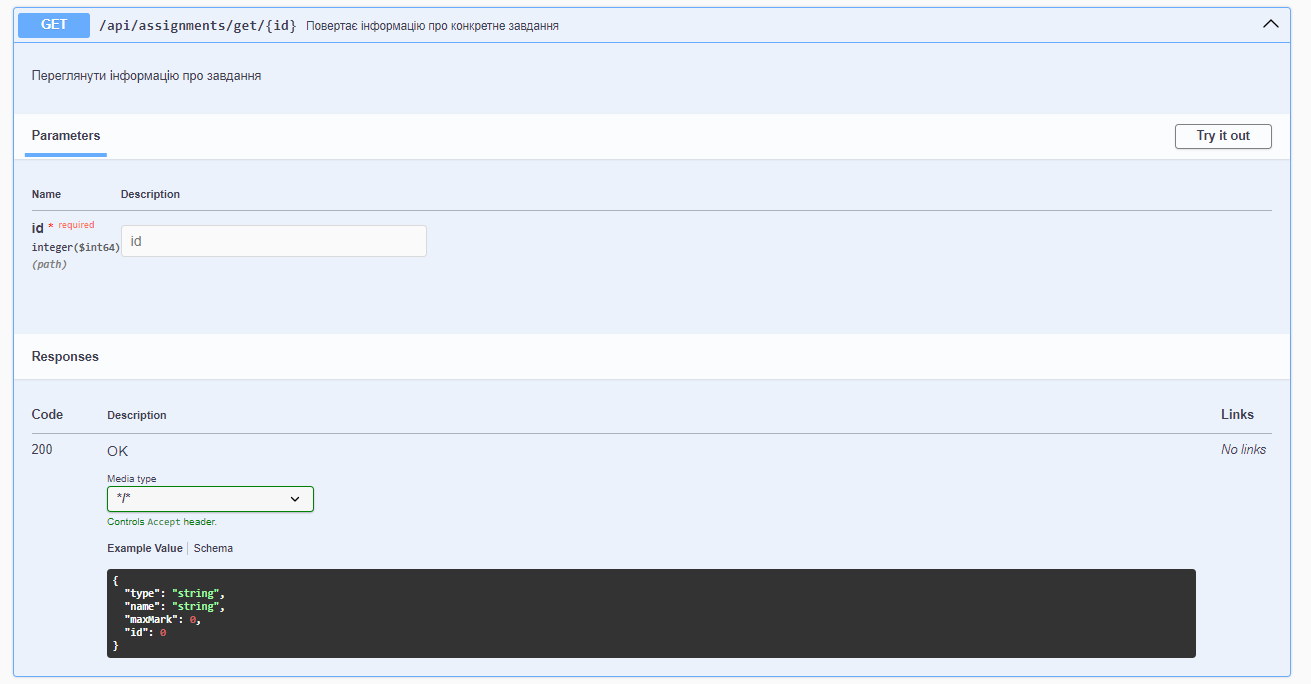
****

Рисунок 3.43 – Документація/api/ assignments/get/{id}

****

Рисунок 3.44 – Документація/api/ assignments/delete/{id}

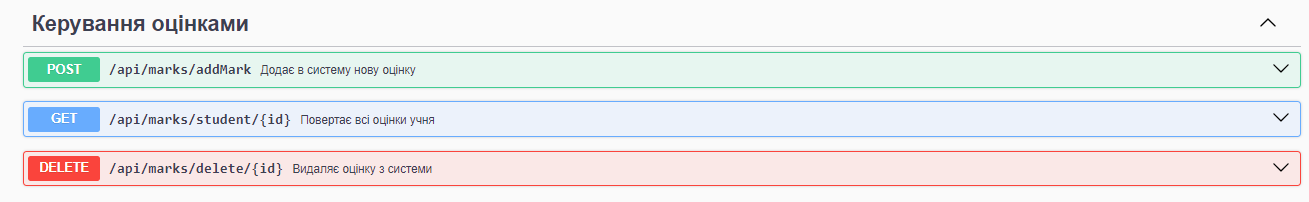
****

Рисунок 3.45 – Документація блоку керування оцінками

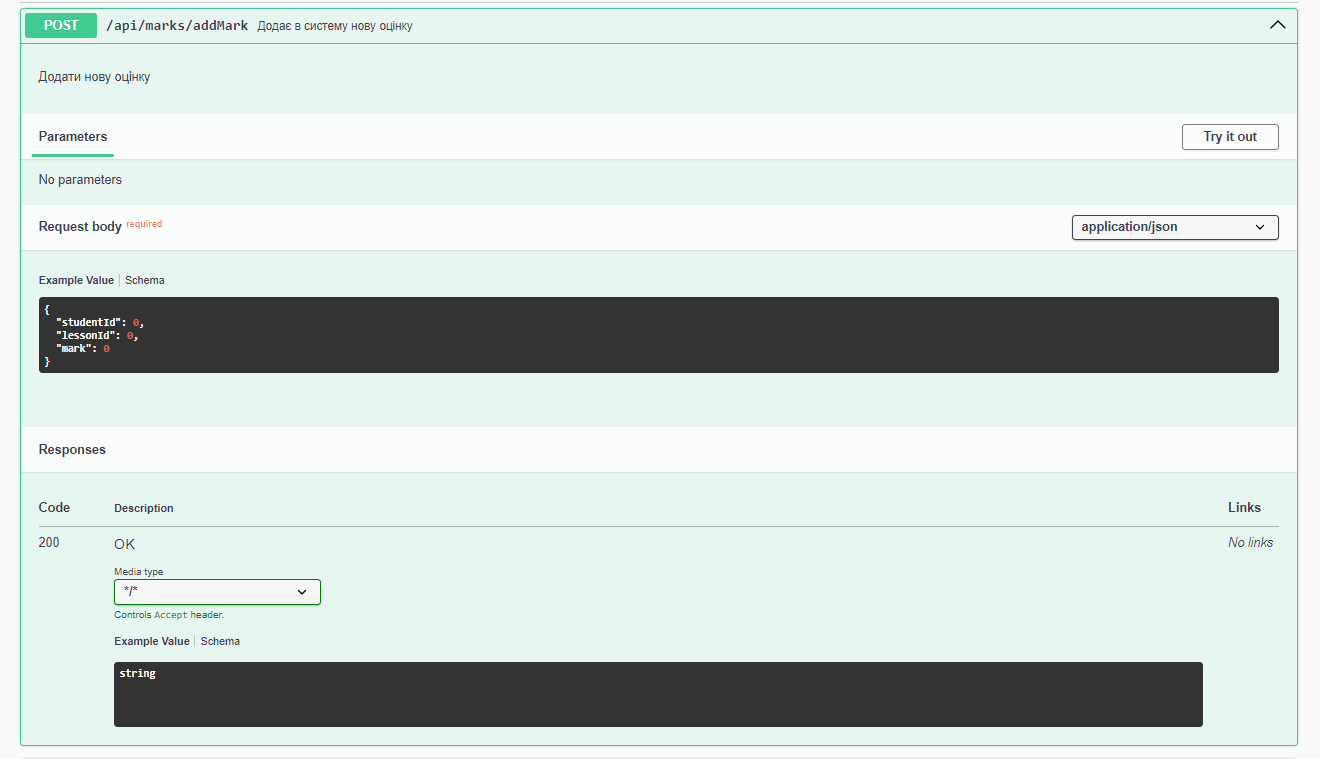
****

Рисунок 3.46 – Документація/api/ marks/addMark

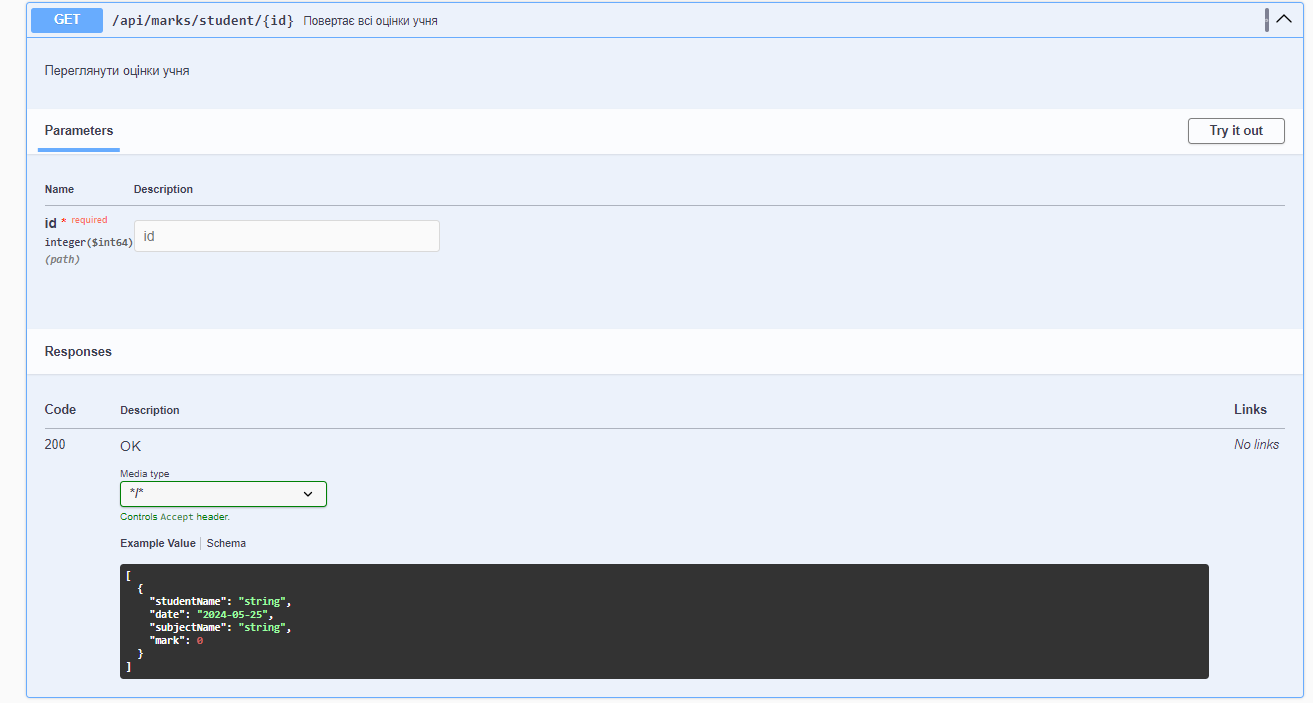
****

Рисунок 3.47 – Документація/api/ marks/student/{id}

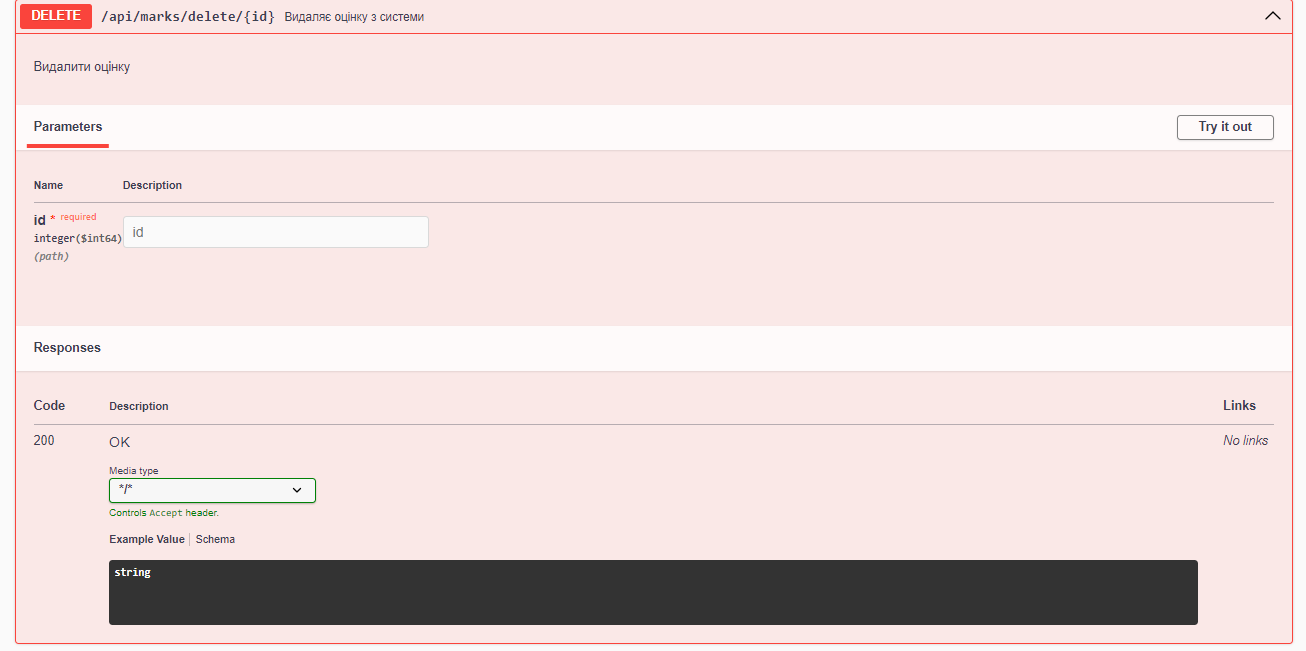
****

Рисунок 3.48 – Документація/api/ marks/delete/{id}

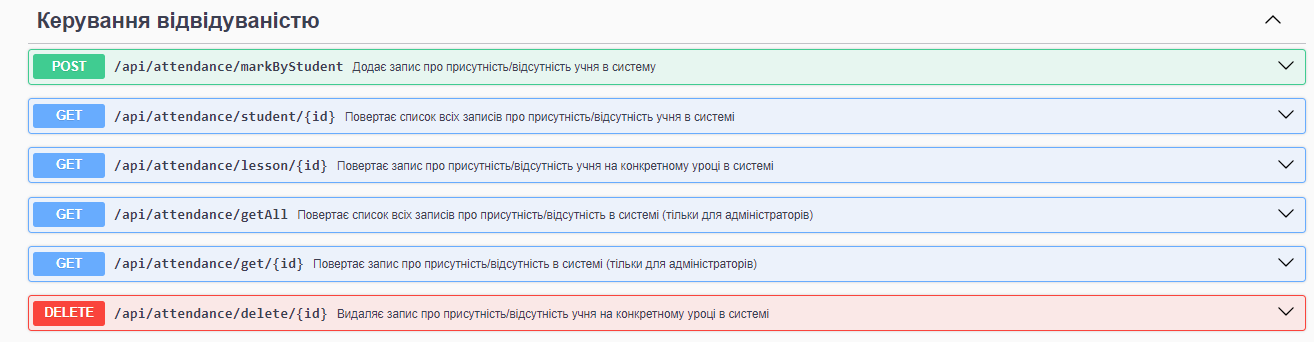
****

Рисунок 3.49 – Документація блоку керування відвідуваністю

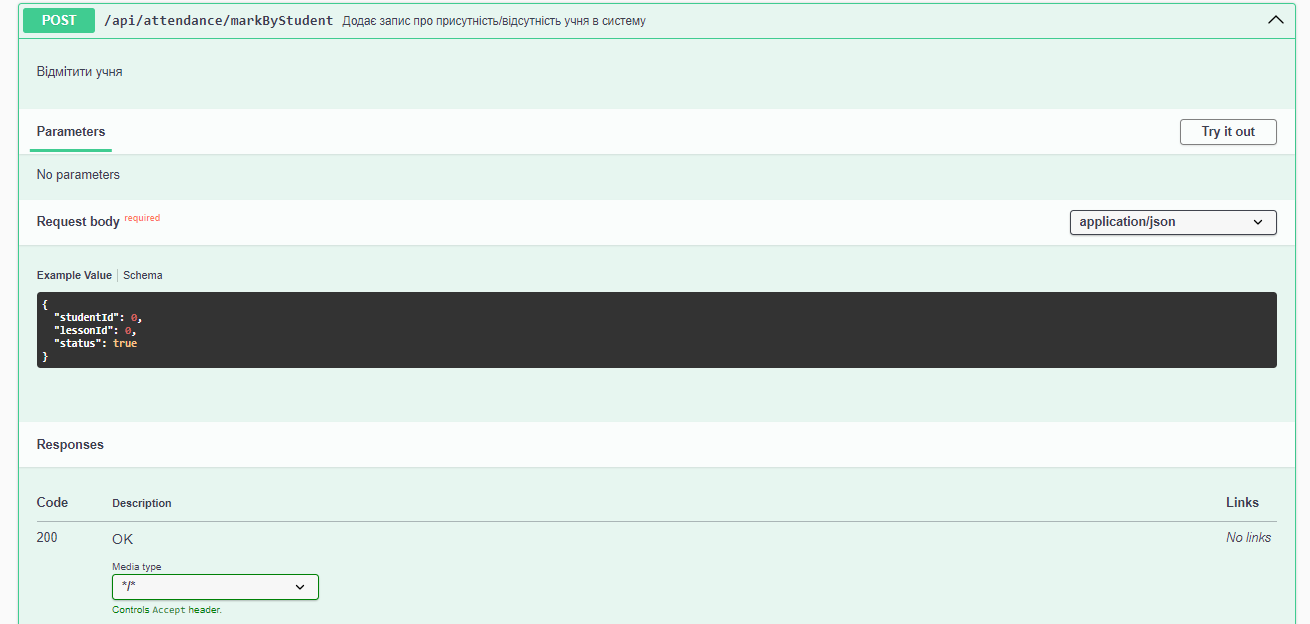
****

Рисунок 3.50 – Документація/api/ attendance/markByStudent

****

Рисунок 3.51 – Документація/api/ attendance/student/{id}

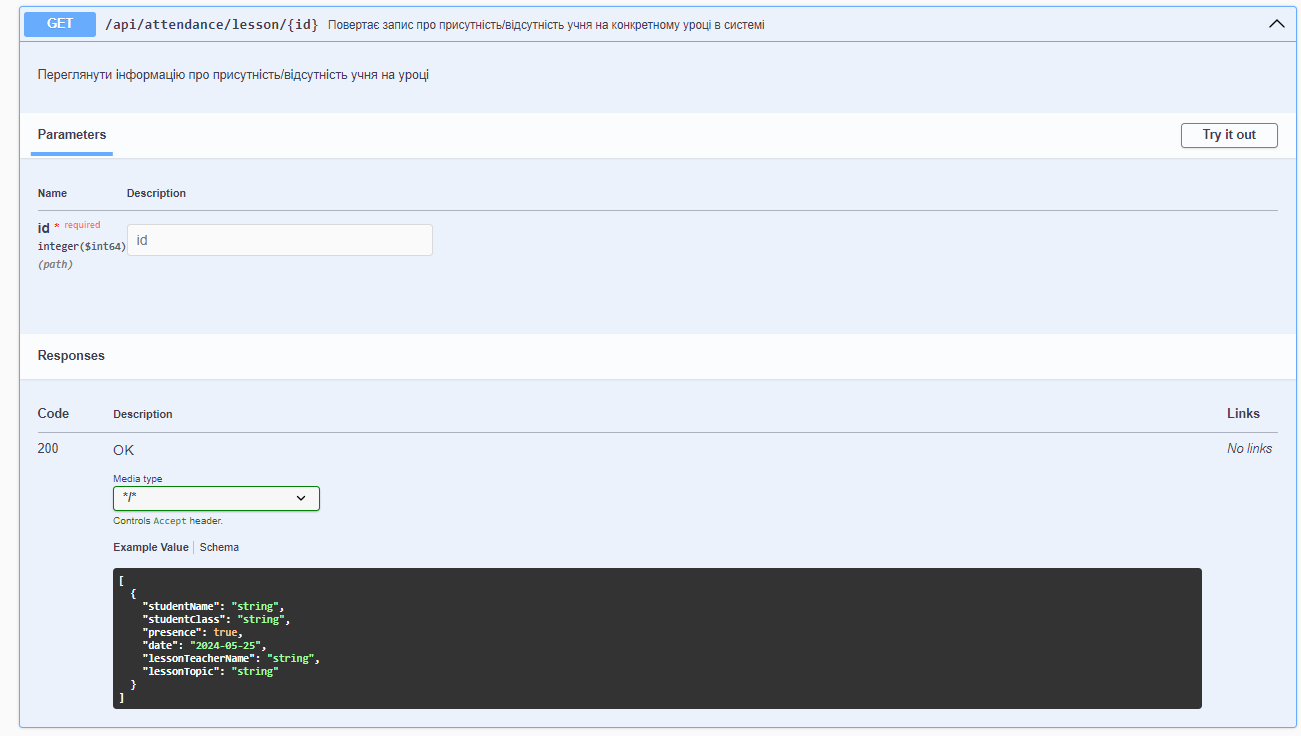
****

Рисунок 3.52 – Документація/api/ attendance/lesson/{id}

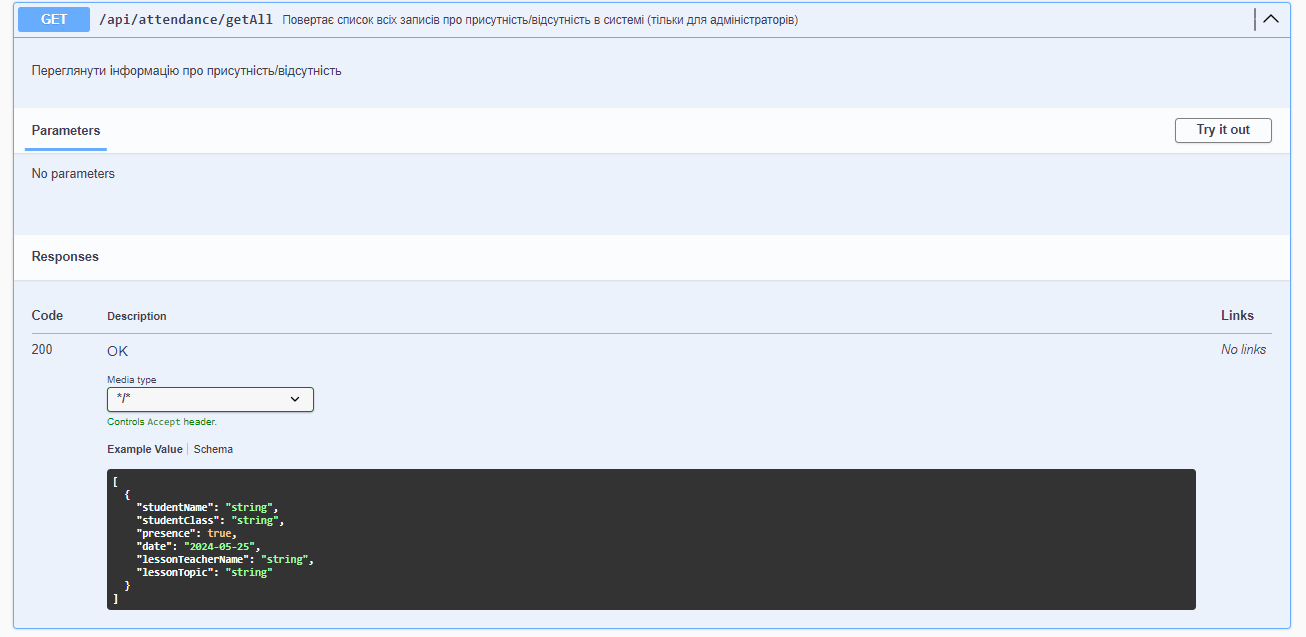
****

Рисунок 3.53 – Документація/api/ attendance/getAll

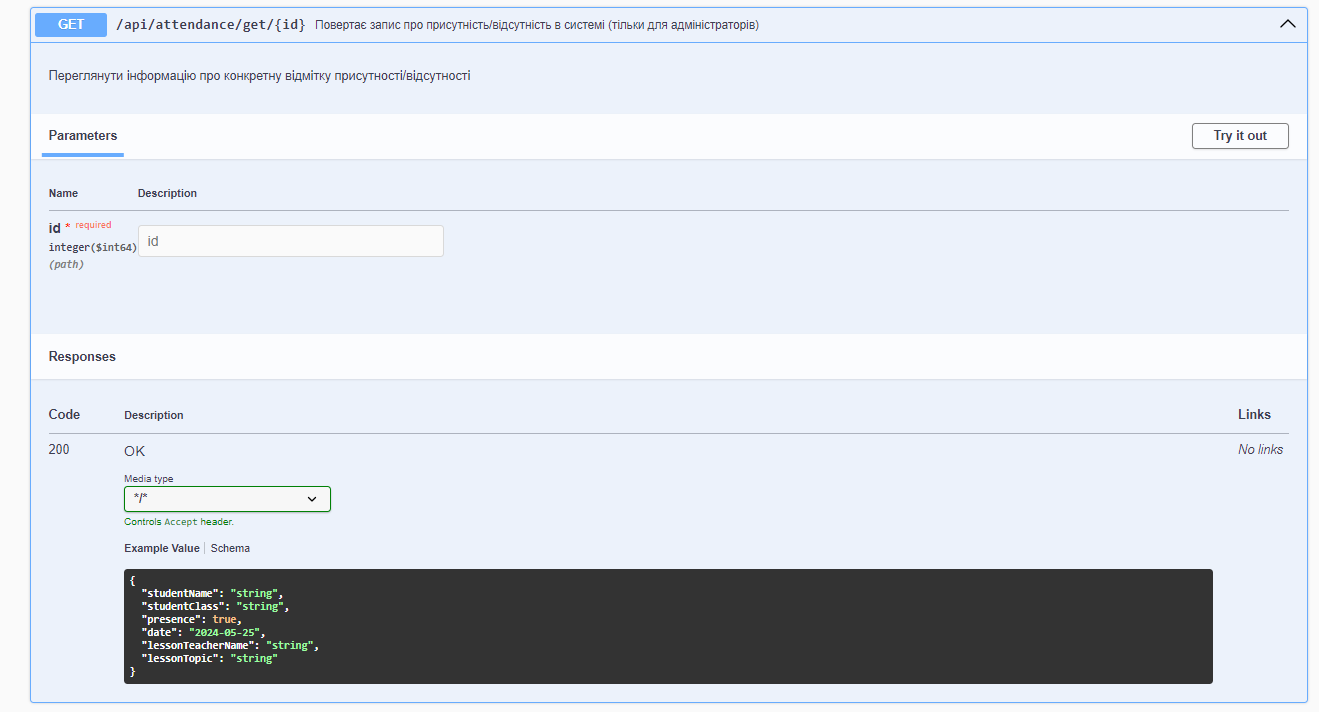
****

Рисунок 3.54 – Документація/api/ attendance/get/{id}

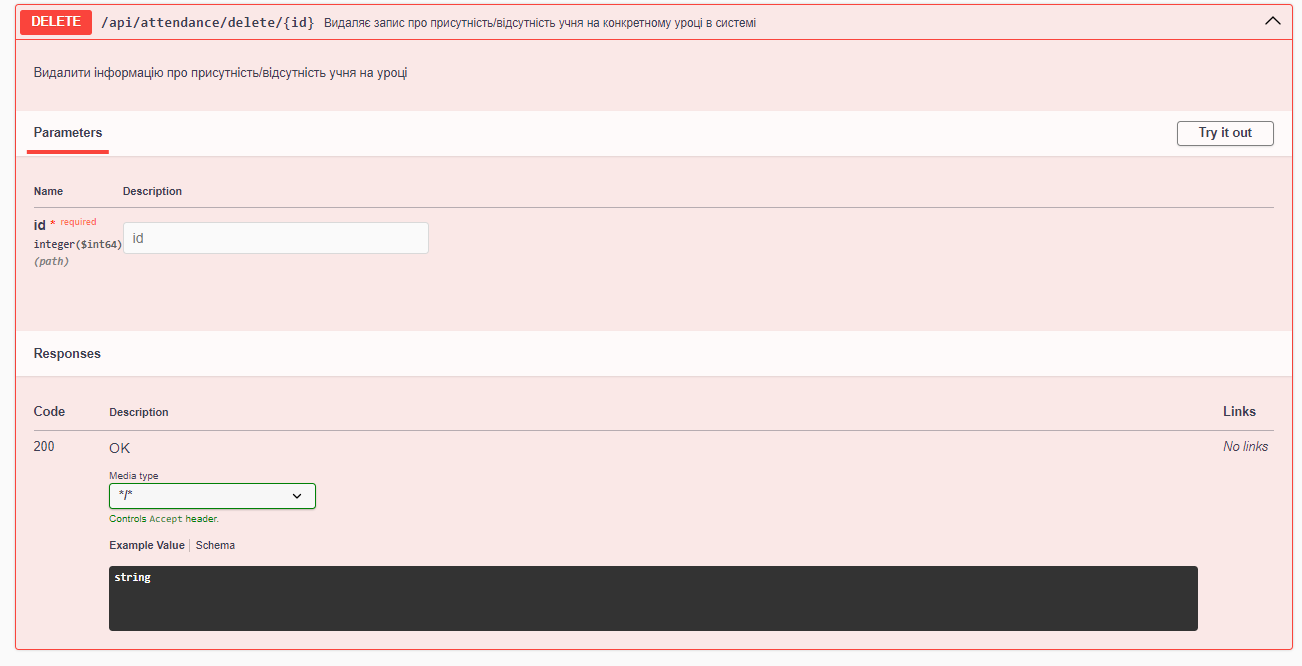
****

Рисунок 3.55 – Документація/api/ attendance/delete/{id}

****

Рисунок 3.56 – Документація блоку інформації про класи

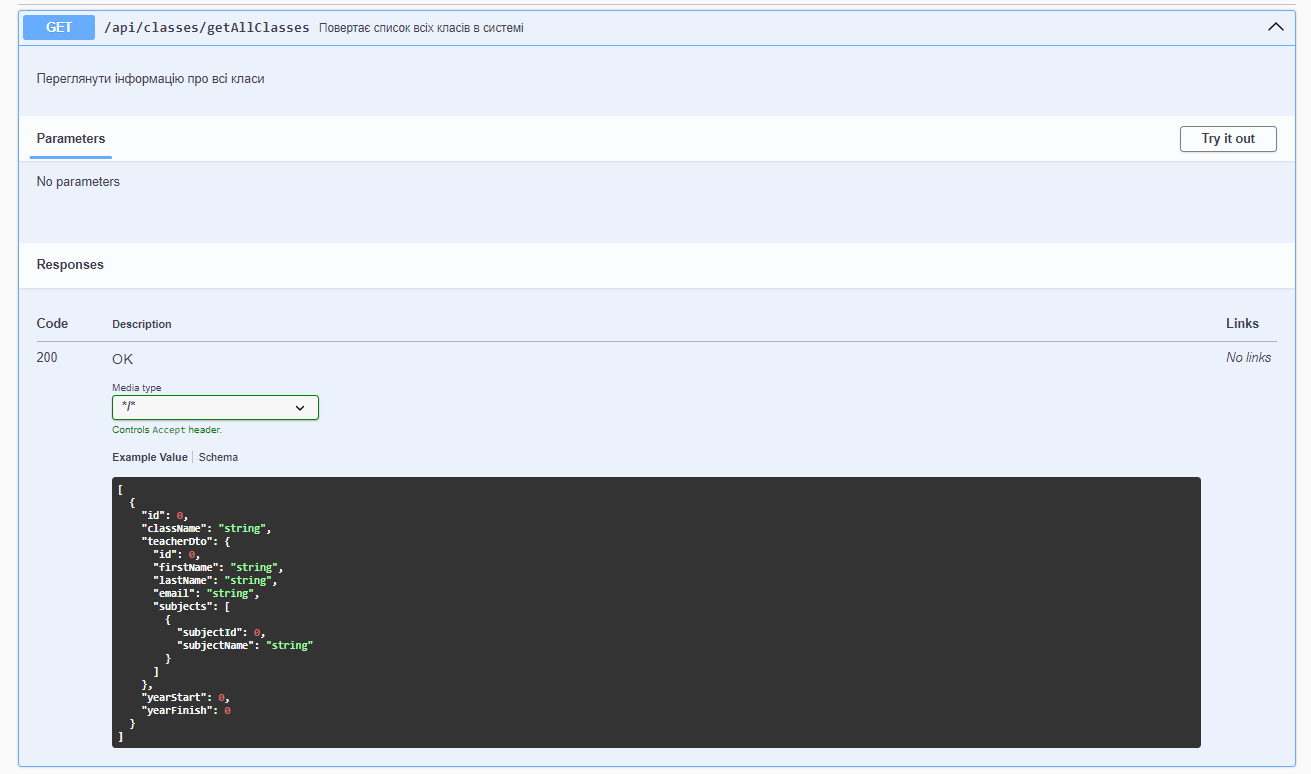
****

Рисунок 3.57 – Документація/api/ classes/getAll

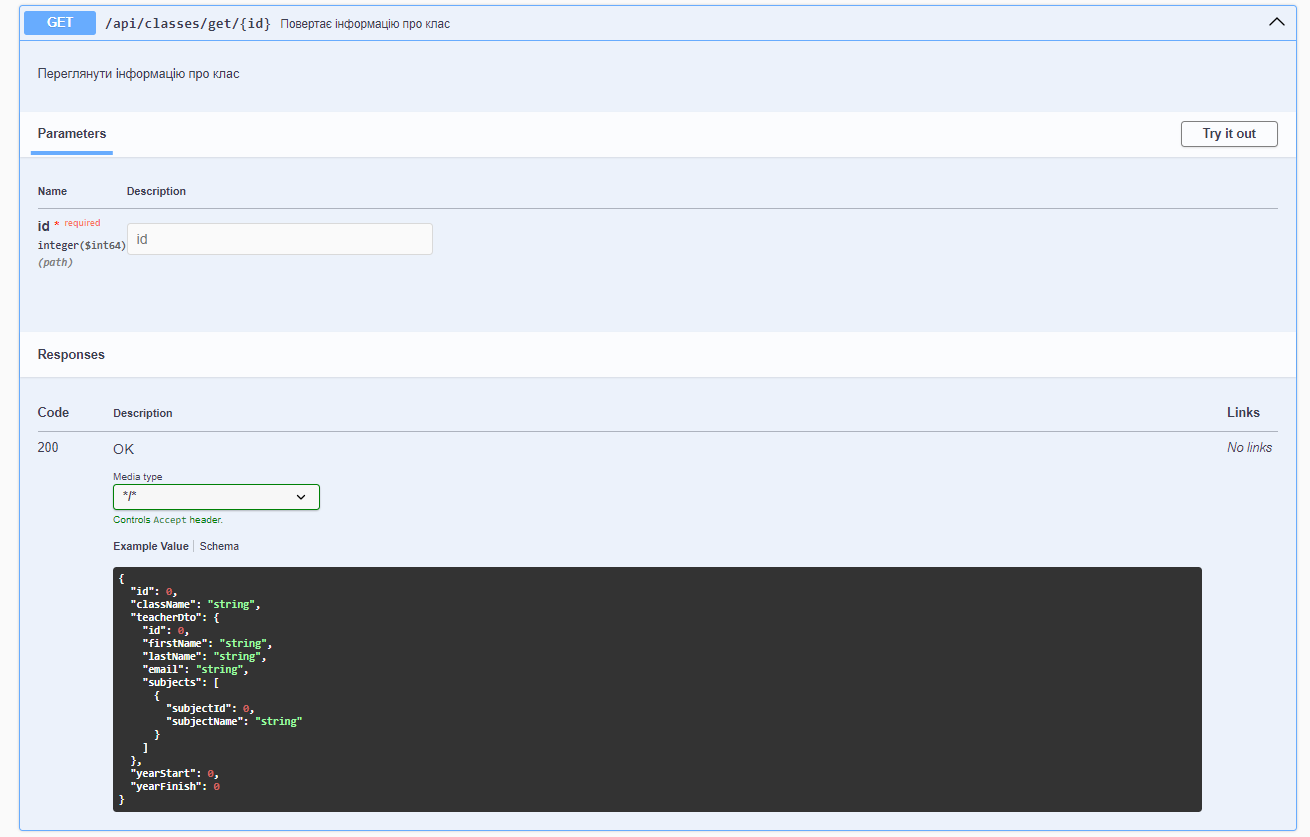
****

Рисунок 3.58 – Документація/api/ classes/get/{id}

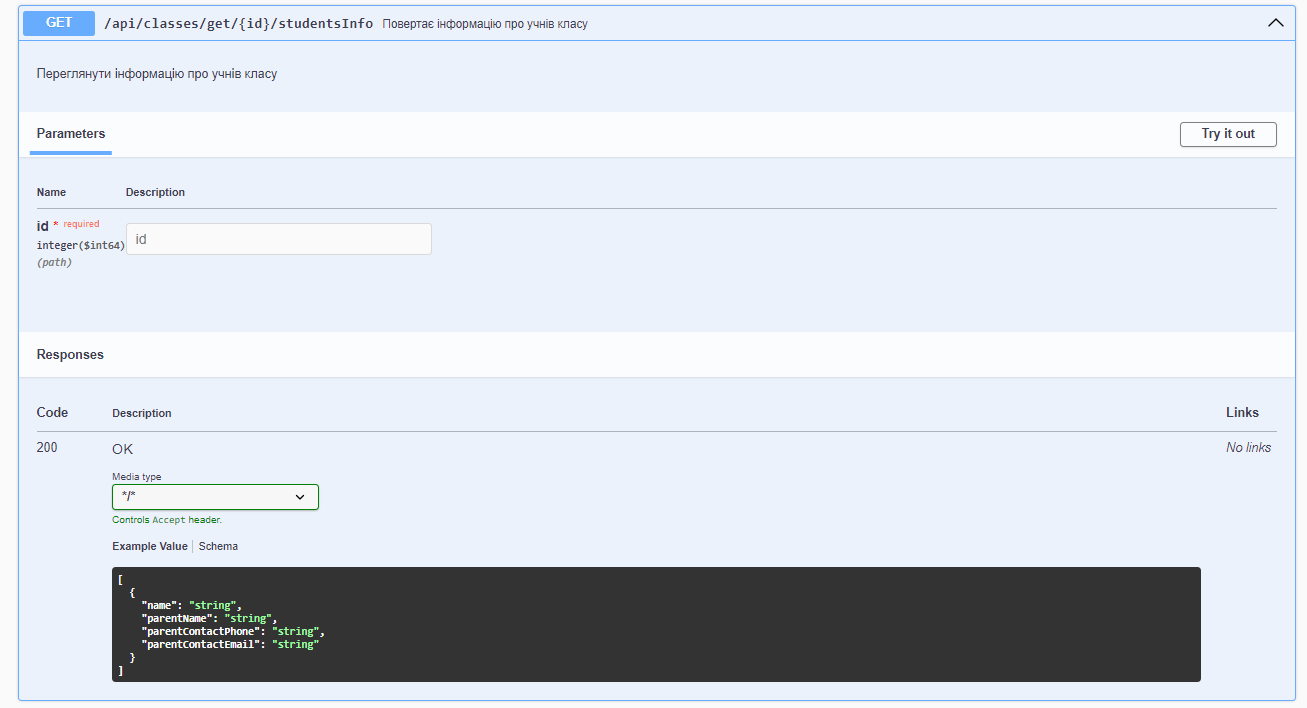
****

Рисунок 3.59 – Документація/api/classes/get/{id}/studentsInfo

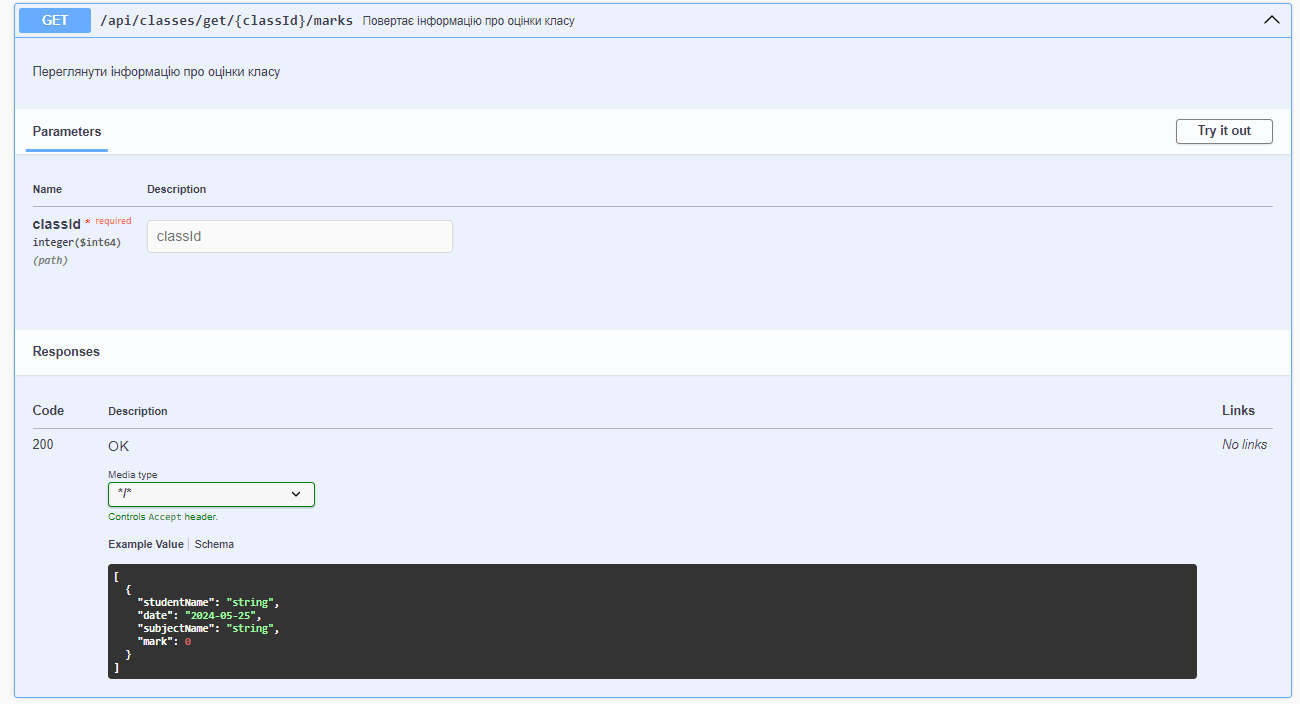
****

Рисунок 3.60 – Документація/api/classes/get/{ classId}/marks

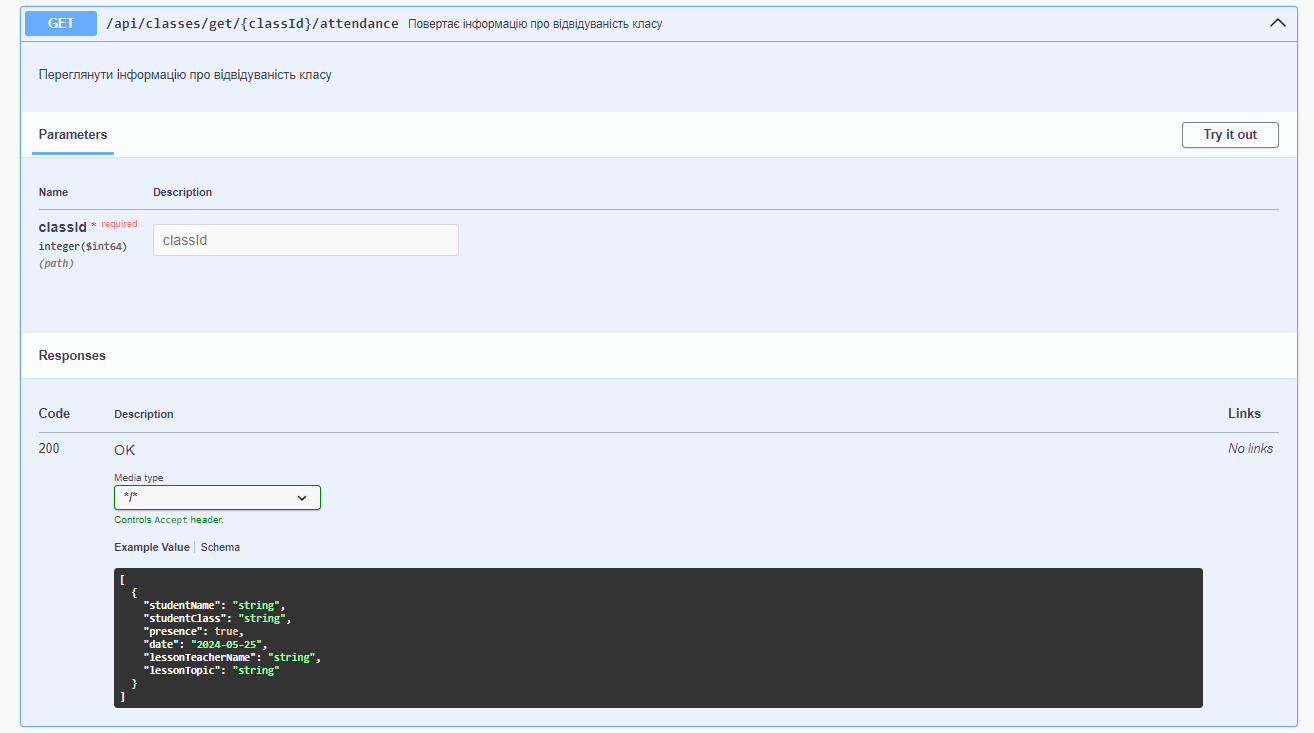
****

Рисунок 3.61 – Документація /api/classes/get/{ classId}/attendance

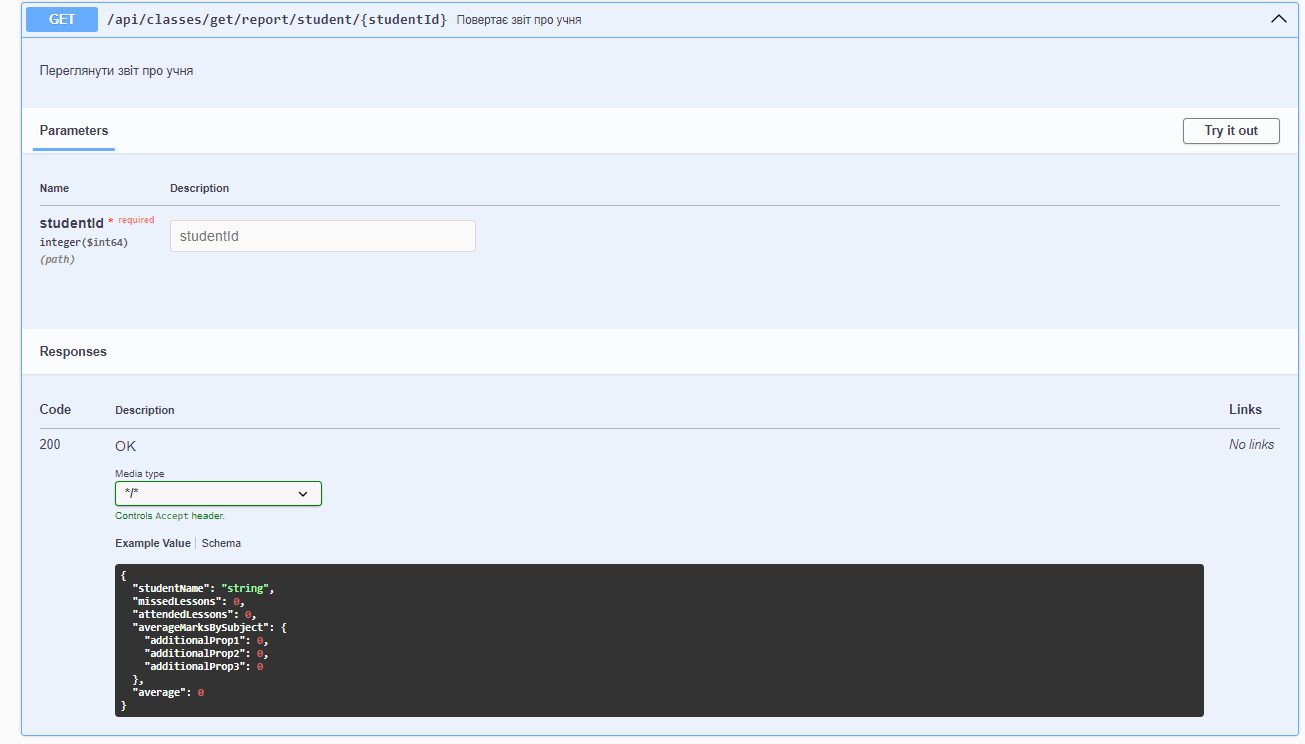
****

Рисунок 3. 62 – Документація /api/classes/get/report/student/{ studentId}

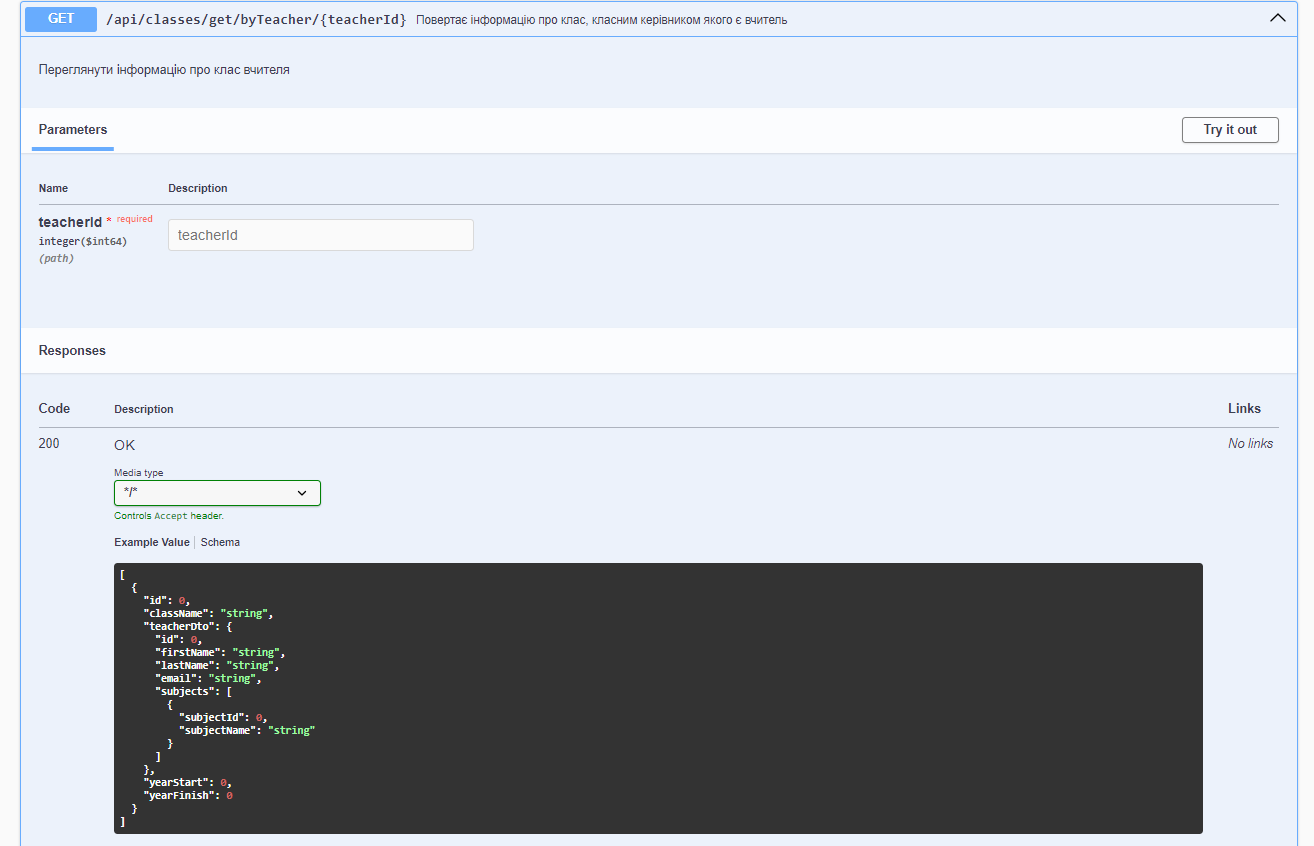
****

Рисунок 3.63 – Документація /api/classes/get/byTeacher/{teacherId}

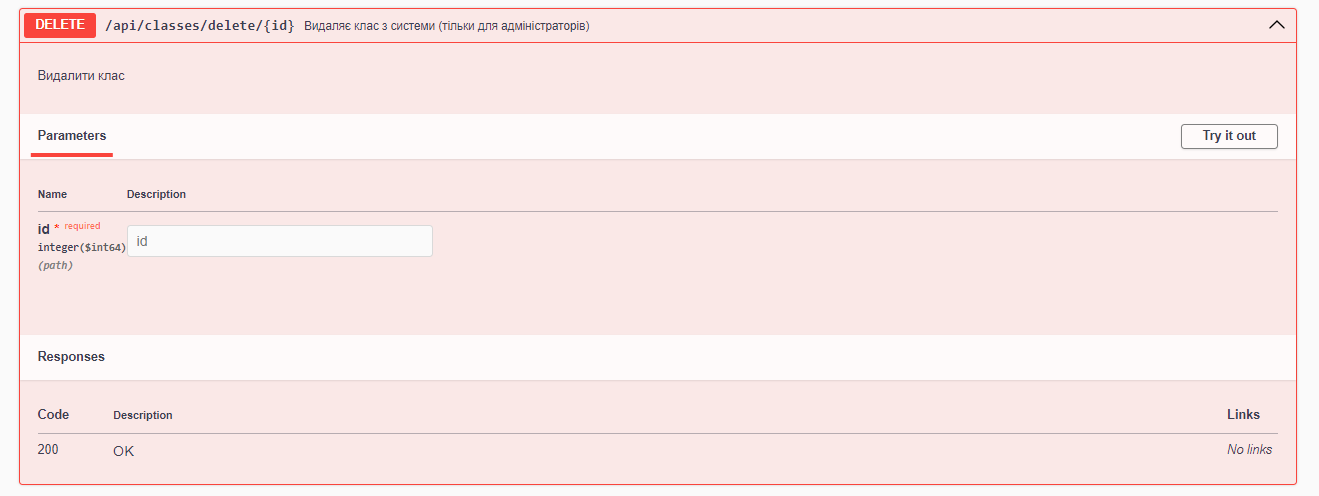
****

Рисунок 3.64 – Документація /api/classes/delete/{id}

**3.3 Збірка проекту в контейнери Docker**

Docker - це платформа для розробки, розгортання та управління контейнеризованими додатками. Вона дозволяє розробникам запускати додатки у віртуальних контейнерах, які мають усі необхідні залежності та середовище для їх виконання.

Щоб зібрати Spring Boot додаток у контейнер Docker, спочатку потрібно створити Dockerfile, який описує середовище та інструкції для створення образу Docker. За допомогою Gradle в середовищі розробки InteliJIdea створимо .jar файл.

Лістинг Dockerfile

FROM openjdk:17-jdk-alpine  
ARG *JAR\_FILE*=build/libs/\*.jar  
COPY ${*JAR\_FILE*} eschool-app.jar  
ENTRYPOINT ["java","-jar","/eschool-app.jar"]

Також треба врахувати те, що маємо в проекті підключення до бази даних.

Лістинг docker-compose.yml

services:  
backend:  
build: .  
container\_name: eschool\_container  
environment:  
 - SPRING\_DATASOURCE\_URL=jdbc:postgresql://db:5432/eschool  
 - SPRING\_DATASOURCE\_USERNAME=postgres  
 - SPRING\_DATASOURCE\_PASSWORD=12345  
depends\_on:  
 - db  
db:  
image: postgres  
container\_name: db  
environment:  
 - POSTGRES\_DB=eschool  
 - POSTGRES\_USER=postgres  
 - POSTGRES\_PASSWORD=12345

За допомогою команди docker-compose up почнемо збірку проекту.

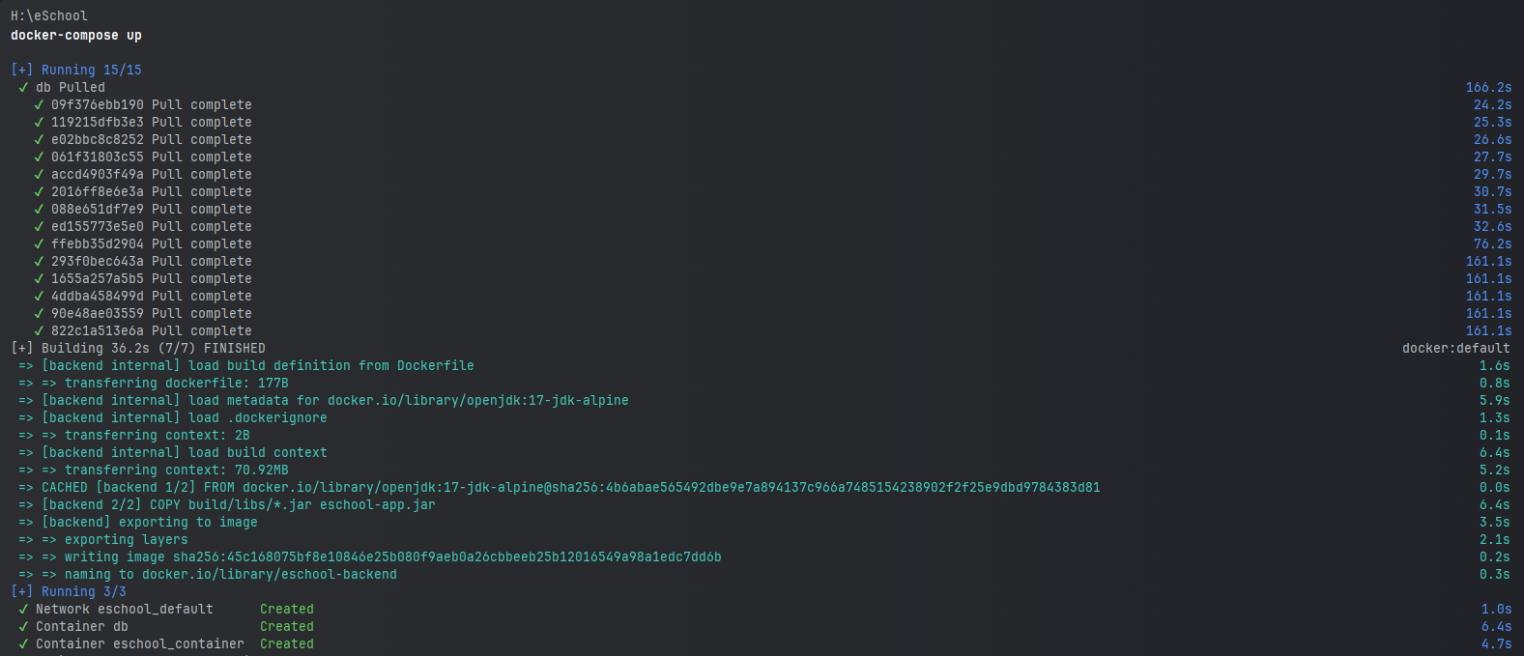


Рисунок 3.65 – Збірка проекту

Бачимо, шо в Docker Desktop з’явилися images, які використовуються в контейнері:

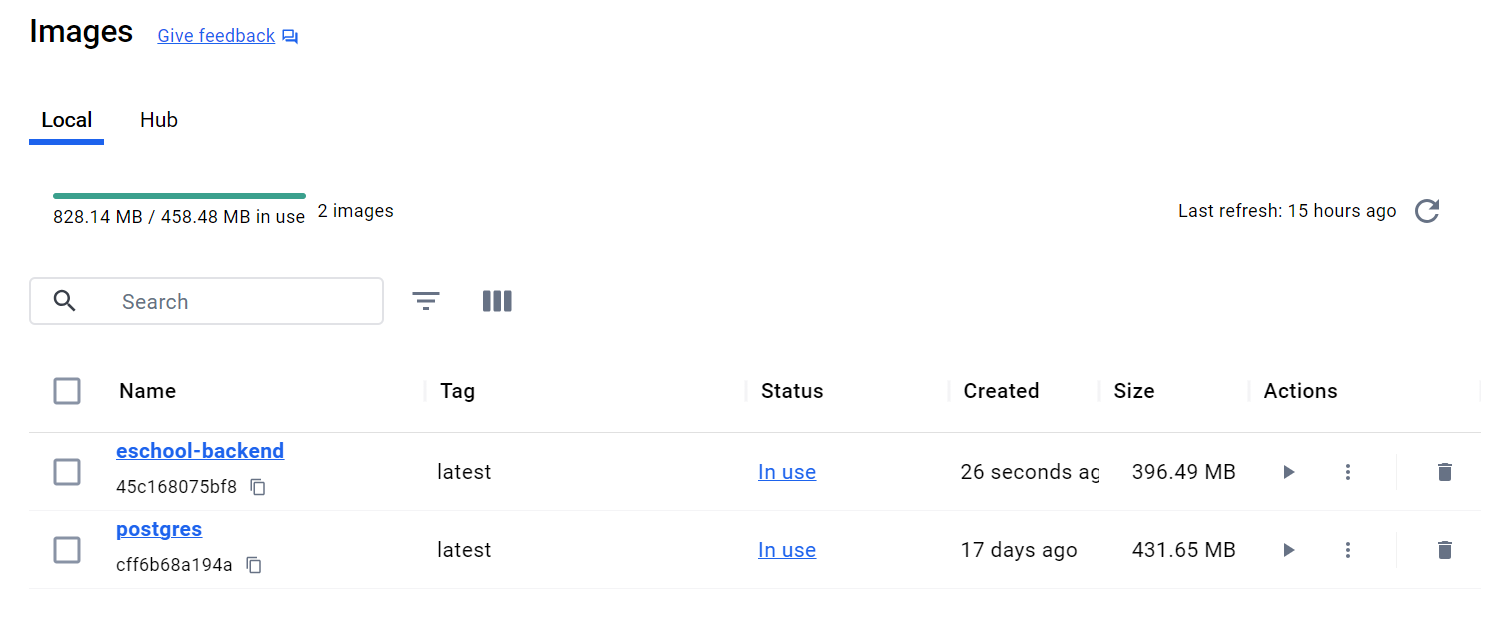


Рисунок 3.66 – Images в Docker Desktop

В терміналі бачимо, що проект запустився.

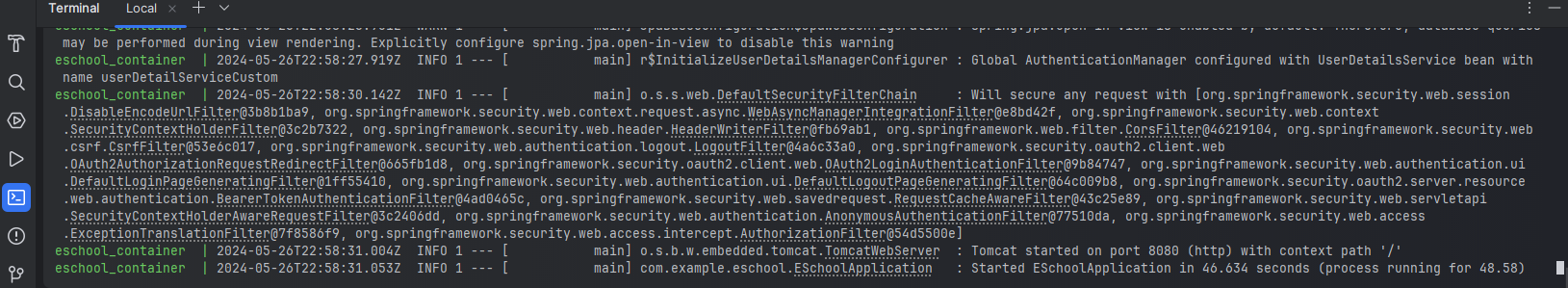


Рисунок 3.67 – Повідомлення про старт ESchoolApplication

Перевіримо його роботу, відкривши браузер:

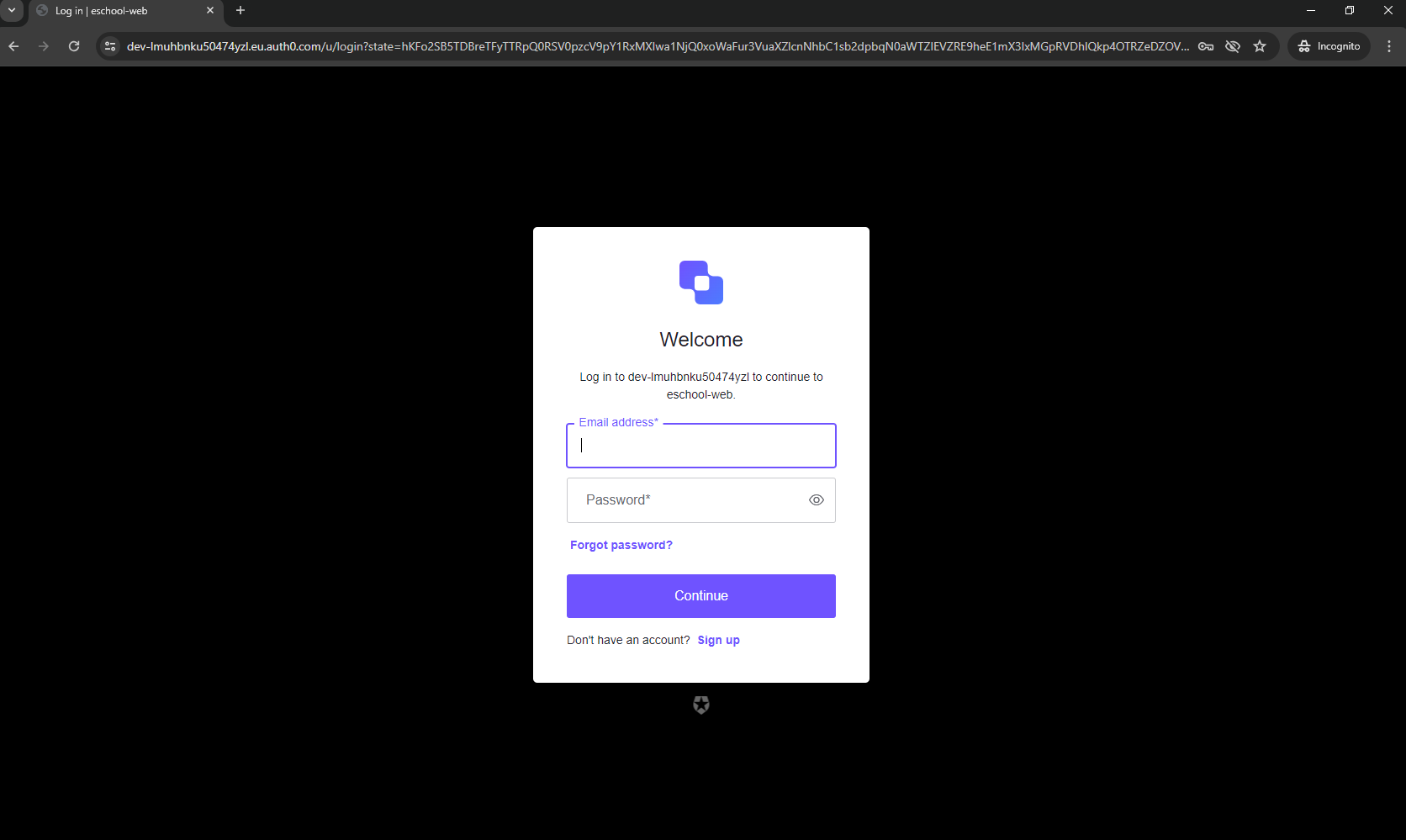


Рисунок 3.68 – Проект у браузері

# **4 НАПИСАННЯ ІНТЕГРАЦІЙНИХ АВТОТЕСТІВ ДЛЯ ВСІХ USER STORIES З ПОЗИТИВНИМИ І НЕГАТИВНИМИ СЦЕНАРІЯМИ**

Інтеграційні тести (інтеграційне тестування) — це тип тестування програмного забезпечення, що спрямований на перевірку взаємодії між різними модулями або компонентами системи. Мета інтеграційного тестування полягає в тому, щоб виявити дефекти в інтерфейсах і взаємодіях між модулями.

Згадаємо описані в першій частині User Stories, які представлені в таблиці 1.

Таблиця 1

|  |  |  |
| --- | --- | --- |
| User Story | Сценарії | |
| Позитивні | Негативні |
| Відслідковування відвідуваності на кожному уроці. | Вчитель успішно відмічає відвідуваність кожного учня на уроці.  Вчитель перевіряє список відвідуваності після відмітки та бачить оновлені дані. | Вчитель намагається відмітити відвідуваність для неіснуючого учня або уроку.  Вчитель вводить недійсний статус відвідуваності (наприклад, не "присутній" або "відсутній").  Вчитель вводить недійсний ідентифікатор учня або уроку. |
| Виставлення оцінок учням. | Вчитель успішно виставляє оцінку учню.  Вчитель перевіряє журнал оцінок та бачить оновлені дані. | Вчитель намагається виставити оцінку для неіснуючого учня або предмету.  Вчитель вводить недійсну оцінку (наприклад, більше ніж максимально допустима оцінка). |
| Планування уроків. | Вчитель успішно планує урок, додаючи матеріали, завдання та інші ресурси.  Вчитель переглядає список запланованих уроків та бачить оновлені дані. | Вчитель намагається планувати урок без обов'язкових матеріалів або завдань.  Вчитель вводить недійсні дані при створенні уроку. |
| Доступ до електронного журналу класу свого класу. | Вчитель успішно отримує доступ до електронного журналу класу і переглядає дані.  Вчитель перевіряє журнал та бачить оновлені дані про успішність учнів. | Вчитель намагається отримати доступ до неіснуючого класу або предмету. |
| Генерація звітів про успішність учнів. | Вчитель успішно генерує звіт про успішність учнів і отримує його.  Вчитель переглядає звіт. | Вчитель намагається згенерувати звіт для неіснуючого класу або предмету.  Вчитель намагається згенерувати звіт без належних даних. |

Створимо тести для кожної User Story.

**Відслідковування відвідуваності на кожному уроці.**

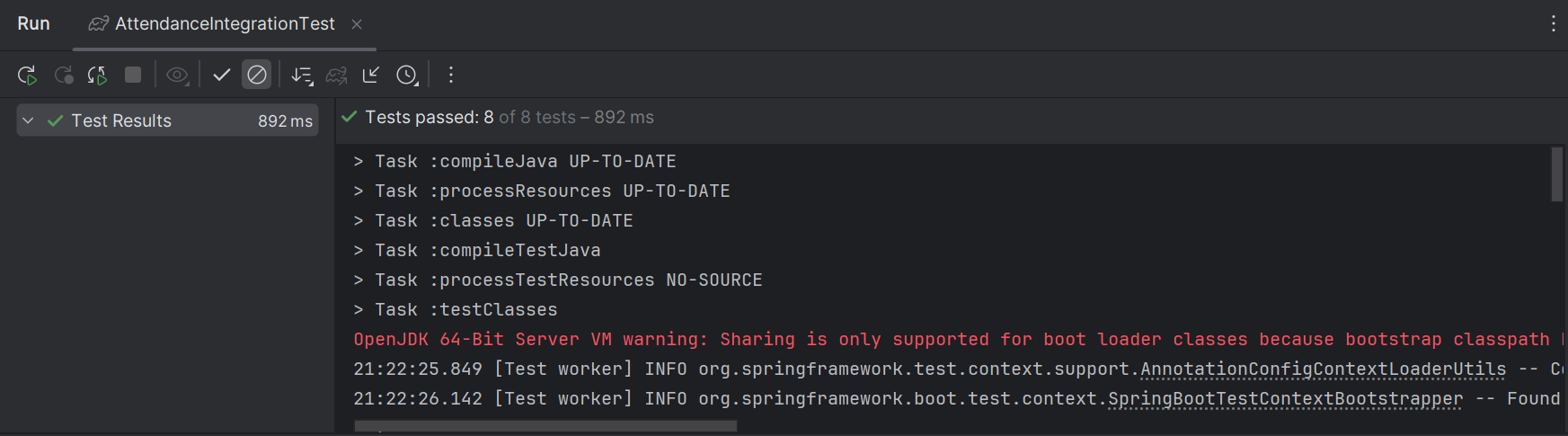


Рисунок 4.1 – Результати проходження тестів

Лістинг AttendanceIntegrationTest:

@SpringBootTest  
@AutoConfigureMockMvc  
public class AttendanceIntegrationTest {  
  
@Autowired  
private MockMvc mockMvc;  
  
@MockBean  
private AttendanceService attendanceService;  
  
@Test  
public void testMarkAttendanceByStudent() throws Exception {  
// Перевірка успішного відмічання відвідуваності для учня  
Mockito.*doNothing*().when(attendanceService).markAttendanceByStudent(1L, 1L, true);  
mockMvc.perform(MockMvcRequestBuilders.*post*("/api/attendance/markByStudent")  
 .header("Authorization", "Bearer " + TestObjectFactory.*getAccessToken*())  
 .contentType(MediaType.*APPLICATION\_JSON*)  
 .content("{\"studentId\":1,\"lessonId\":1,\"status\":true}"))  
 .andExpect(MockMvcResultMatchers.*status*().isOk())  
 .andExpect(MockMvcResultMatchers.*content*().string("Attendance marked successfully"));  
 }  
  
@Test  
public void testMarkAttendanceForNonExistentStudent() throws Exception {  
// Перевірка, як відбувається відмітка відвідуваності для неіснуючого учня  
*doThrow*(EntityNotFoundException.class)  
 .when(attendanceService)  
 .markAttendanceByStudent(*anyLong*(), *anyLong*(), *anyBoolean*());  
  
mockMvc.perform(MockMvcRequestBuilders.*post*("/api/attendance/markByStudent")  
 .header("Authorization", "Bearer " + TestObjectFactory.*getAccessToken*())  
 .contentType(MediaType.*APPLICATION\_JSON*)  
 .content("{\"studentId\":999,\"lessonId\":1,\"status\":true}"))  
 .andExpect(MockMvcResultMatchers.*status*().isNotFound());  
 }  
  
@Test  
public void testMarkAttendanceWithInvalidLessonId() throws Exception {  
// Перевірка, як відбувається відмітка відвідуваності з недійсним ID уроку  
*doThrow*(EntityNotFoundException.class)  
 .when(attendanceService)  
 .markAttendanceByStudent(*anyLong*(), *eq*(999L), *anyBoolean*());  
  
mockMvc.perform(MockMvcRequestBuilders.*post*("/api/attendance/markByStudent")  
 .header("Authorization", "Bearer " + TestObjectFactory.*getAccessToken*())  
 .contentType(MediaType.*APPLICATION\_JSON*)  
 .content("{\"studentId\":1,\"lessonId\":999,\"status\":true}"))  
 .andExpect(MockMvcResultMatchers.*status*().isNotFound());  
 }  
  
@Test  
public void testMarkAttendanceWithInvalidStatus() throws Exception {  
// Перевірка, як відбувається відмітка відвідуваності з недійсним статусом  
*doThrow*(IllegalArgumentException.class)  
 .when(attendanceService)  
 .markAttendanceByStudent(*anyLong*(), *anyLong*(), *anyBoolean*());  
  
mockMvc.perform(MockMvcRequestBuilders.*post*("/api/attendance/markByStudent")  
 .header("Authorization", "Bearer " + TestObjectFactory.*getAccessToken*())  
 .contentType(MediaType.*APPLICATION\_JSON*)  
 .content("{\"studentId\":1,\"lessonId\":1,\"status\":null}"))  
 .andExpect(MockMvcResultMatchers.*status*().isBadRequest())  
 .andExpect(MockMvcResultMatchers.*content*().string("Status must be either true or false"));  
 }  
  
@Test  
public void testGetAllAttendance() throws Exception {  
// Перевірка отримання всіх записів відвідуваності  
TeacherDto teacherDto = TestObjectFactory.*createSampleTeacher*(1L);  
 ClassDto \_class = TestObjectFactory.*createSampleClass*(1L);  
 StudentDto student1 = TestObjectFactory.*createSampleStudent*(1L,"John", "Smith", \_class);  
 StudentDto student2 = TestObjectFactory.*createSampleStudent*(2L, "Jane", "Doe", \_class);  
  
 LessonPlanDto lp = TestObjectFactory.*createSampleLessonPlanDto*(1L);  
  
 LessonDto lesson = TestObjectFactory.*createSampleLesson*(1L, teacherDto, \_class, lp);  
  
 List<AttendanceDto> attendanceList = Arrays.*asList*(  
new AttendanceDto(1L, student1, lesson,true, LocalDate.*of*(2024, 4, 28)),  
new AttendanceDto(2L, student2, lesson,true, LocalDate.*of*(2024, 4, 28))  
 );  
*when*(attendanceService.getAllAttendances()).thenReturn(attendanceList);  
  
mockMvc.perform(MockMvcRequestBuilders.*get*("/api/attendance/getAll")  
 .header("Authorization", "Bearer " + TestObjectFactory.*getAccessToken*()))  
 .andExpect(MockMvcResultMatchers.*status*().isOk())  
 .andExpect(MockMvcResultMatchers.*jsonPath*("$.size()").value(attendanceList.size()));  
 }  
  
@Test  
public void testGetAttendanceByStudentId() throws Exception {  
// Перевірка отримання відвідуваності за ID учня  
TeacherDto teacherDto = TestObjectFactory.*createSampleTeacher*(1L);  
 ClassDto \_class = TestObjectFactory.*createSampleClass*(1L);  
 StudentDto student1 = TestObjectFactory.*createSampleStudent*(1L,"John", "Smith", \_class);  
 StudentDto student2 = TestObjectFactory.*createSampleStudent*(2L, "Jane", "Doe", \_class);  
 LessonPlanDto lp = TestObjectFactory.*createSampleLessonPlanDto*(1L);  
 LessonDto lesson = TestObjectFactory.*createSampleLesson*(1L, teacherDto, \_class, lp);  
  
 List<AttendanceDto> attendanceList = Arrays.*asList*(  
new AttendanceDto(1L, student1, lesson,true, LocalDate.*of*(2024, 4, 28)),  
new AttendanceDto(2L, student2, lesson,true, LocalDate.*of*(2024, 4, 28))  
 );  
*when*(attendanceService.getAttendanceByStudentId(1L)).thenReturn(attendanceList);  
  
mockMvc.perform(MockMvcRequestBuilders.*get*("/api/attendance/student/1")  
 .header("Authorization", "Bearer " + TestObjectFactory.*getAccessToken*()))  
 .andExpect(MockMvcResultMatchers.*status*().isOk())  
 .andExpect(MockMvcResultMatchers.*jsonPath*("$.size()").value(attendanceList.size()));  
 }  
  
@Test  
public void testGetAttendanceByLessonId() throws Exception {  
// Перевірка отримання відвідуваності за ID уроку  
TeacherDto teacherDto = TestObjectFactory.*createSampleTeacher*(1L);  
 ClassDto \_class = TestObjectFactory.*createSampleClass*(1L);  
 StudentDto student1 = TestObjectFactory.*createSampleStudent*(1L,"John", "Smith", \_class);  
 StudentDto student2 = TestObjectFactory.*createSampleStudent*(2L, "Jane", "Doe", \_class);  
  
 LessonPlanDto lp = TestObjectFactory.*createSampleLessonPlanDto*(1L);  
  
 LessonDto lesson = TestObjectFactory.*createSampleLesson*(1L, teacherDto, \_class, lp);  
  
 List<AttendanceDto> attendanceList = Arrays.*asList*(  
new AttendanceDto(1L, student1, lesson,true, LocalDate.*of*(2024, 4, 28)),  
new AttendanceDto(2L, student2, lesson,true, LocalDate.*of*(2024, 4, 28))  
 );  
*when*(attendanceService.getAttendanceByLessonId(1L)).thenReturn(attendanceList);  
  
mockMvc.perform(MockMvcRequestBuilders.*get*("/api/attendance/lesson/1")  
 .header("Authorization", "Bearer " + TestObjectFactory.*getAccessToken*()))  
 .andExpect(MockMvcResultMatchers.*status*().isOk())  
 .andExpect(MockMvcResultMatchers.*jsonPath*("$.size()").value(attendanceList.size()));  
 }  
  
@Test  
public void testDeleteAttendanceById() throws Exception {  
// Перевірка видалення запису відвідуваності за ID  
mockMvc.perform(MockMvcRequestBuilders.*delete*("/api/attendance/delete/1")  
 .header("Authorization", "Bearer " + TestObjectFactory.*getAccessToken*()))  
 .andExpect(MockMvcResultMatchers.*status*().isOk())  
 .andExpect(MockMvcResultMatchers.*content*().string("Attendance deleted successfully"));  
 }  
}

**Виставлення оцінок учням.**

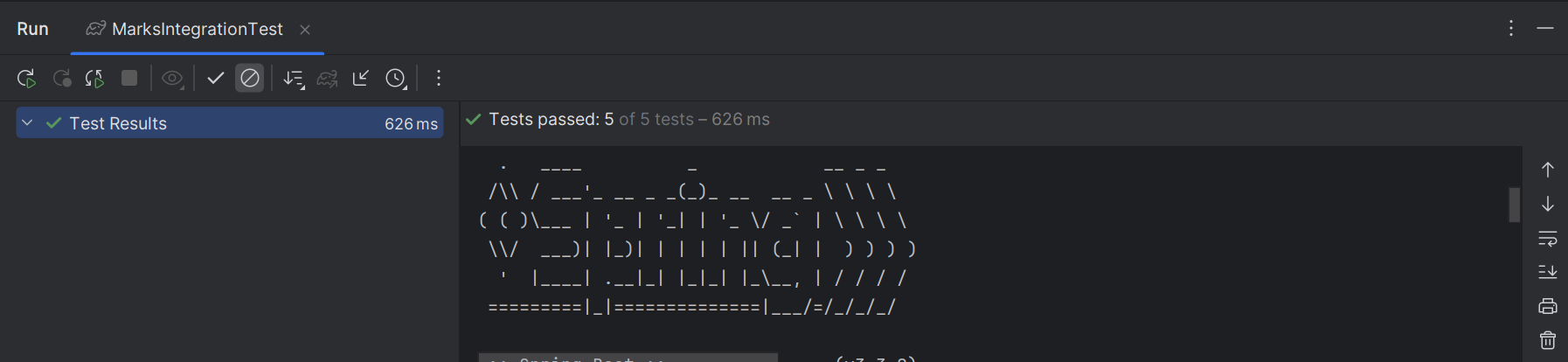


Рисунок 4.2 – Результати проходження тестів

Лістинг MarksIntegrationTest:

@SpringBootTest  
@AutoConfigureMockMvc  
public class MarksIntegrationTest {  
@Autowired  
private MockMvc mockMvc;  
  
@MockBean  
private MarkService markService;  
  
@Test  
public void testTeacherAddsMarkForStudent() throws Exception {  
// Перевірка додавання оцінки для учня  
Mockito.*doNothing*().when(markService).markStudent(*anyLong*(), *anyLong*(), *anyInt*());  
  
mockMvc.perform(MockMvcRequestBuilders.*post*("/api/marks/addMark")  
 .header("Authorization", "Bearer " + TestObjectFactory.*getAccessToken*())  
 .contentType(MediaType.*APPLICATION\_JSON*)  
 .content("{\"studentId\":1,\"lessonId\":1,\"mark\":10}"))  
 .andExpect(MockMvcResultMatchers.*status*().isOk())  
 .andExpect(MockMvcResultMatchers.*content*().string("Mark added successfully"));  
 }  
  
@Test  
public void testAddMarkForNonExistentStudent() throws Exception {  
// Перевірка додавання оцінки для неіснуючого учня  
Mockito.*doThrow*(EntityNotFoundException.class)  
 .when(markService)  
 .markStudent(999L, 1L, 9);  
  
mockMvc.perform(MockMvcRequestBuilders.*post*("/api/marks/addMark")  
 .header("Authorization", "Bearer " + TestObjectFactory.*getAccessToken*())  
 .contentType(MediaType.*APPLICATION\_JSON*)  
 .content("{\"studentId\":999,\"lessonId\":1,\"mark\":9}"))  
 .andExpect(MockMvcResultMatchers.*status*().isNotFound());  
 }  
  
@Test  
public void testAddMarkForNonExistentLesson() throws Exception {  
// Перевірка додавання оцінки для неіснуючого уроку  
Mockito.*doThrow*(EntityNotFoundException.class)  
 .when(markService)  
 .markStudent(1L, 999L, 8);  
  
mockMvc.perform(MockMvcRequestBuilders.*post*("/api/marks/addMark")  
 .header("Authorization", "Bearer " + TestObjectFactory.*getAccessToken*())  
 .contentType(MediaType.*APPLICATION\_JSON*)  
 .content("{\"studentId\":1,\"lessonId\":999,\"mark\":8}"))  
 .andExpect(MockMvcResultMatchers.*status*().isNotFound());  
 }  
  
@Test  
public void testAddInvalidMarkValue() throws Exception {  
// Перевірка додавання недійсної оцінки  
Mockito.*doThrow*(IllegalArgumentException.class)  
 .when(markService)  
 .markStudent(1L, 1L, 150);  
  
mockMvc.perform(MockMvcRequestBuilders.*post*("/api/marks/addMark")  
 .header("Authorization", "Bearer " + TestObjectFactory.*getAccessToken*())  
 .contentType(MediaType.*APPLICATION\_JSON*)  
 .content("{\"studentId\":1,\"lessonId\":1,\"mark\":150}"))  
 .andExpect(MockMvcResultMatchers.*status*().isBadRequest());  
 }  
  
@Test  
public void testGetMarkByStudentId() throws Exception {  
// Перевірка отримання всіх оцінок учня  
TeacherDto teacherDto = TestObjectFactory.*createSampleTeacher*(1L);  
 ClassDto \_class = TestObjectFactory.*createSampleClass*(1L);  
 StudentDto student = TestObjectFactory.*createSampleStudent*(2L, "Jane", "Doe", \_class);  
 LessonPlanDto lp = TestObjectFactory.*createSampleLessonPlanDto*(1L);  
 LessonDto lesson = TestObjectFactory.*createSampleLesson*(1L, teacherDto, \_class, lp);  
 MarkDto markDto = TestObjectFactory.*createSampleMark*(1L, 9, lesson, student);  
  
 List<MarkDto> markList = *singletonList*(markDto);  
  
 Mockito.*when*(markService.getMarksByStudentId(1L)).thenReturn(markList);  
  
mockMvc.perform(MockMvcRequestBuilders.*get*("/api/marks/student/2")  
 .header("Authorization", "Bearer " + TestObjectFactory.*getAccessToken*()))  
 .andExpect(MockMvcResultMatchers.*status*().isOk());  
 }  
}

**Планування уроків**

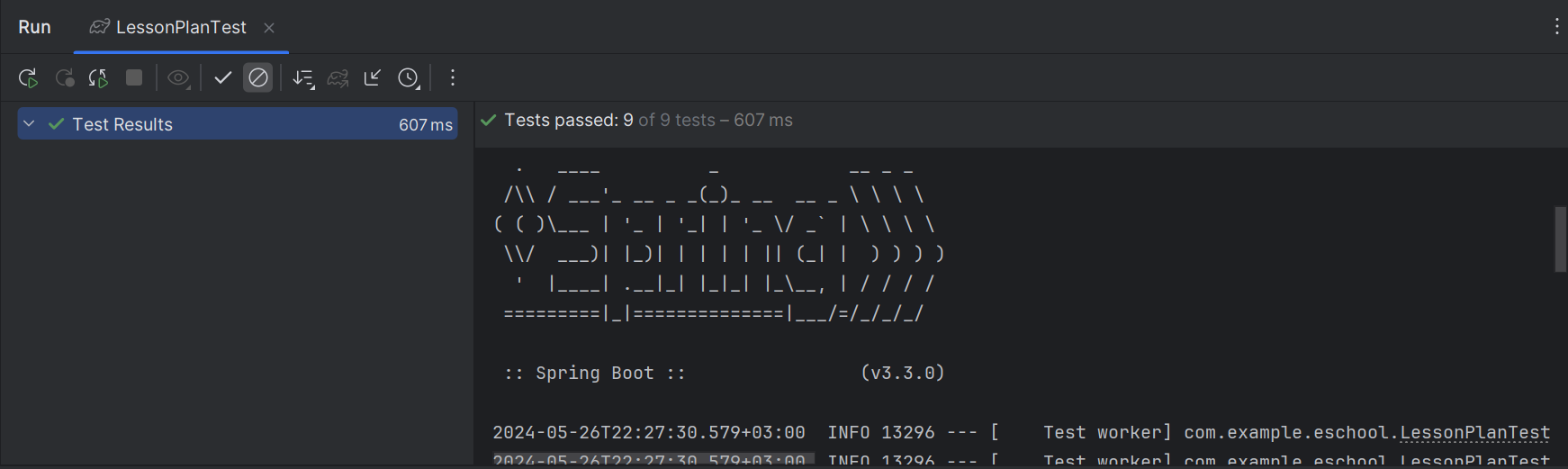


Рисунок 4.3 – Результати проходження тестів

Лістинг LessonPlanTest:

@SpringBootTest  
@AutoConfigureMockMvc  
public class LessonPlanTest {  
  
@Autowired  
private MockMvc mockMvc;  
  
@MockBean  
private LessonPlanService lessonPlanService;  
  
@Test  
public void testCreateLessonPlan() throws Exception {  
// Перевірка успішного додавання плану уроку  
LessonPlanDto lessonPlanDto = new LessonPlanDto();  
 lessonPlanDto.setTopic("Mathematics");  
 lessonPlanDto.setDescription("Algebra lesson");  
  
 Mockito.*when*(lessonPlanService.createLessonPlan(*any*(), *any*(), *anyLong*(), *any*(), *any*()))  
 .thenReturn(lessonPlanDto);  
  
mockMvc.perform(MockMvcRequestBuilders.*post*("/api/lesson\_plans/add")  
 .header("Authorization", "Bearer " + TestObjectFactory.*getAccessToken*())  
 .contentType(MediaType.*APPLICATION\_JSON*)  
 .content("{\"topic\":\"Mathematics\",\"description\":\"Algebra lesson\",\"subjectId\":1,\"assignmentIds\":[1],\"materialIds\":[1]}"))  
 .andExpect(MockMvcResultMatchers.*status*().isCreated())  
 .andExpect(MockMvcResultMatchers.*jsonPath*("$.topic").value("Mathematics"))  
 .andExpect(MockMvcResultMatchers.*jsonPath*("$.description").value("Algebra lesson"));  
 }  
  
  
@Test  
public void testGetAllLessonPlans() throws Exception {  
// Перевірка отримання списку всіх планів уроків  
LessonPlanDto lessonPlanDto = new LessonPlanDto();  
 lessonPlanDto.setTopic("Mathematics");  
  
 Mockito.*when*(lessonPlanService.getAllLessonPlans())  
 .thenReturn(Collections.*singletonList*(lessonPlanDto));  
  
mockMvc.perform(MockMvcRequestBuilders.*get*("/api/lesson\_plans/getAll")  
 .header("Authorization", "Bearer " + TestObjectFactory.*getAccessToken*()))  
 .andExpect(MockMvcResultMatchers.*status*().isOk())  
 .andExpect(MockMvcResultMatchers.*jsonPath*("$[0].topic").value("Mathematics"));  
 }  
  
  
@Test  
public void testCreateLessonPlanWithInvalidData() throws Exception {  
// Перевірка додавання плану уроку з недійсними даними  
Mockito.*doThrow*(IllegalArgumentException.class)  
 .when(lessonPlanService)  
 .createLessonPlan(*any*(), *any*(), *anyLong*(), *any*(), *any*());  
  
mockMvc.perform(MockMvcRequestBuilders.*post*("/api/lesson\_plans/add")  
 .header("Authorization", "Bearer " + TestObjectFactory.*getAccessToken*())  
 .contentType(MediaType.*APPLICATION\_JSON*)  
 .content("{\"topic\":\"\",\"description\":\"Algebra lesson\",\"subjectId\":1,\"assignmentIds\":[1],\"materialIds\":[1]}"))  
 .andExpect(MockMvcResultMatchers.*status*().isBadRequest());  
 }  
  
@Test  
public void testGetLessonPlanById() throws Exception {  
// Перевірка отримання плану уроку за ID  
LessonPlanDto lessonPlanDto = new LessonPlanDto();  
 lessonPlanDto.setTopic("Mathematics");  
  
 Mockito.*when*(lessonPlanService.getLessonPlanById(*anyLong*()))  
 .thenReturn(lessonPlanDto);  
  
mockMvc.perform(MockMvcRequestBuilders.*get*("/api/lesson\_plans/get/1")  
 .header("Authorization", "Bearer " + TestObjectFactory.*getAccessToken*()))  
 .andExpect(MockMvcResultMatchers.*status*().isOk())  
 .andExpect(MockMvcResultMatchers.*jsonPath*("$.topic").value("Mathematics"));  
 }  
  
@Test  
public void testGetLessonPlanByNonExistentId() throws Exception {  
// Перевірка отримання плану уроку за неіснуючим ID  
Mockito.*doThrow*(EntityNotFoundException.class)  
 .when(lessonPlanService)  
 .getLessonPlanById(*anyLong*());  
  
mockMvc.perform(MockMvcRequestBuilders.*get*("/api/lesson\_plans/get/999")  
 .header("Authorization", "Bearer " + TestObjectFactory.*getAccessToken*()))  
 .andExpect(MockMvcResultMatchers.*status*().isNotFound());  
 }  
  
@Test  
public void testUpdateLessonPlan() throws Exception {  
// Перевірка успішного оновлення плану уроку  
LessonPlanDto lessonPlanDto = new LessonPlanDto();  
 lessonPlanDto.setTopic("Mathematics Updated");  
 lessonPlanDto.setDescription("Updated Algebra lesson");  
  
 Mockito.*when*(lessonPlanService.updateLessonPlan(*anyLong*(), *any*(), *any*(), *anyLong*(), *any*(), *any*()))  
 .thenReturn(lessonPlanDto);  
  
mockMvc.perform(MockMvcRequestBuilders.*put*("/api/lesson\_plans/update/1")  
 .header("Authorization", "Bearer " + TestObjectFactory.*getAccessToken*())  
 .contentType(MediaType.*APPLICATION\_JSON*)  
 .content("{\"topic\":\"Mathematics Updated\",\"description\":\"Updated Algebra lesson\",\"subjectId\":1,\"assignmentIds\":[1],\"materialIds\":[1]}"))  
 .andExpect(MockMvcResultMatchers.*status*().isOk())  
 .andExpect(MockMvcResultMatchers.*jsonPath*("$.topic").value("Mathematics Updated"))  
 .andExpect(MockMvcResultMatchers.*jsonPath*("$.description").value("Updated Algebra lesson"));  
 }  
  
@Test  
public void testUpdateLessonPlanWithInvalidData() throws Exception {  
// Перевірка оновлення плану уроку з недійсними даними  
Mockito.*doThrow*(IllegalArgumentException.class)  
 .when(lessonPlanService)  
 .updateLessonPlan(*anyLong*(), *any*(), *any*(), *anyLong*(), *any*(), *any*());  
  
mockMvc.perform(MockMvcRequestBuilders.*put*("/api/lesson\_plans/update/1")  
 .header("Authorization", "Bearer " + TestObjectFactory.*getAccessToken*())  
 .contentType(MediaType.*APPLICATION\_JSON*)  
 .content("{\"topic\":\"\",\"description\":\"Updated Algebra lesson\",\"subjectId\":1,\"assignmentIds\":[1],\"materialIds\":[1]}"))  
 .andExpect(MockMvcResultMatchers.*status*().isBadRequest());  
 }  
  
@Test  
public void testDeleteLessonPlan() throws Exception {  
// Перевірка успішного видалення плану уроку  
Mockito.*doNothing*().when(lessonPlanService).deleteLessonPlan(*anyLong*());  
  
mockMvc.perform(MockMvcRequestBuilders.*delete*("/api/lesson\_plans/delete/1")  
 .header("Authorization", "Bearer " + TestObjectFactory.*getAccessToken*()))  
 .andExpect(MockMvcResultMatchers.*status*().isNoContent());  
 }  
  
@Test  
public void testDeleteNonExistentLessonPlan() throws Exception {  
// Перевірка видалення неіснуючого плану уроку  
Mockito.*doThrow*(EntityNotFoundException.class)  
 .when(lessonPlanService)  
 .deleteLessonPlan(*anyLong*());  
  
mockMvc.perform(MockMvcRequestBuilders.*delete*("/api/lesson\_plans/delete/999")  
 .header("Authorization", "Bearer " + TestObjectFactory.*getAccessToken*()))  
 .andExpect(MockMvcResultMatchers.*status*().isNotFound());  
 }  
}

**Доступ до електронного журналу класу свого класу та генерація звітів про успішність учнів.**

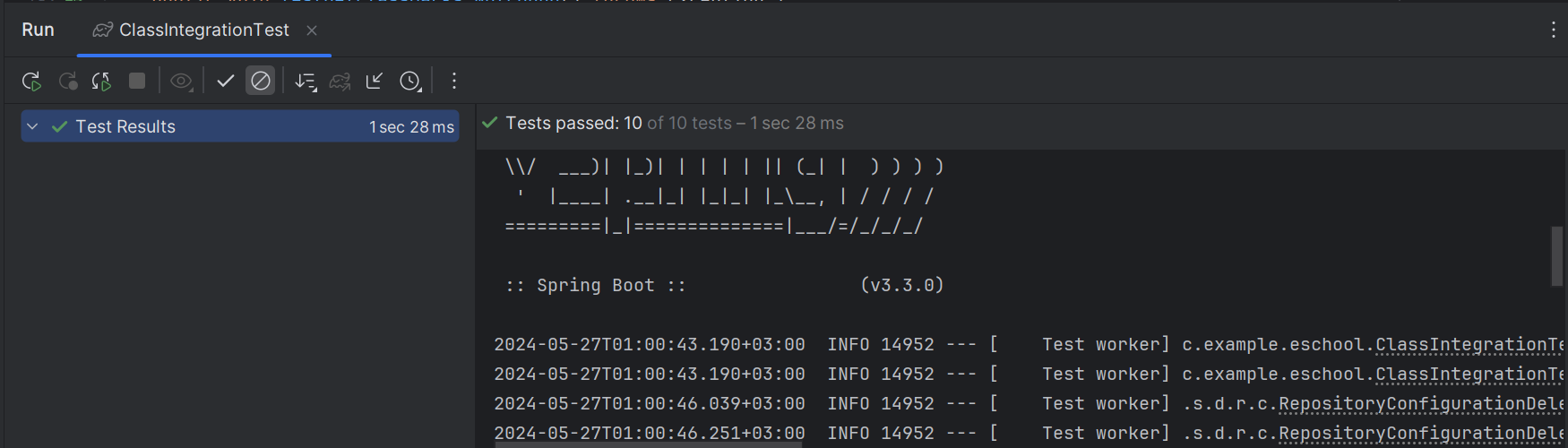


Рисунок 4.4 – Результати проходження тестів

Лістинг ClassIntegrationTest:

@SpringBootTest  
@AutoConfigureMockMvc  
@Transactional  
public class ClassIntegrationTest {  
  
@Autowired  
private MockMvc mockMvc;  
@MockBean  
private ClassService classService;  
  
@Test  
public void testGetClassById\_Success() throws Exception {  
// Перевірка успішного отримання класу за ідентифікатором.  
ClassDto classDto = TestObjectFactory.*createSampleClass*(1L);  
 classDto.setClassName("5A");  
  
*when*(classService.getClassById(1L)).thenReturn(classDto);  
mockMvc.perform(*get*("/api/classes/get/{id}", 1L)  
 .header("Authorization", "Bearer " + TestObjectFactory.*getAccessToken*())  
 .contentType(MediaType.*APPLICATION\_JSON*))  
 .andExpect(*status*().isOk())  
 .andExpect(*content*().contentType(MediaType.*APPLICATION\_JSON*))  
 .andExpect(*jsonPath*("$.className").value("5A"));  
 }  
  
@Test  
public void testGetClassById\_NotFound() throws Exception {  
// Перевірка випадку, коли клас не знайдено.  
*when*(classService.getClassById(999L)).thenThrow(EntityNotFoundException.class);  
  
mockMvc.perform(*get*("/api/classes/get/{id}", 999L)  
 .header("Authorization", "Bearer " + TestObjectFactory.*getAccessToken*())  
 .contentType(MediaType.*APPLICATION\_JSON*))  
 .andExpect(*status*().isNotFound());  
 }  
  
@Test  
public void testGenerateStudentReport\_Success() throws Exception {  
// Перевірка успішного генерування звіту про учня.  
StudentReportDto studentReportDto = new StudentReportDto();  
 studentReportDto.setStudentName("John Doe");  
 studentReportDto.setMissedLessons(2);  
  
*when*(classService.generateStudentReport(1L)).thenReturn(studentReportDto);  
mockMvc.perform(*get*("/api/classes/get/report/student/{studentId}", 1L)  
 .header("Authorization", "Bearer " + TestObjectFactory.*getAccessToken*())  
 .contentType(MediaType.*APPLICATION\_JSON*))  
 .andExpect(*status*().isOk())  
 .andExpect(*content*().contentType(MediaType.*APPLICATION\_JSON*))  
 .andExpect(*jsonPath*("$.studentName").value("John Doe"))  
 .andExpect(*jsonPath*("$.missedLessons").value(2));  
 }  
  
@Test  
public void testGenerateStudentReport\_NotFound() throws Exception {  
// Перевірка випадку, коли звіт про учня не знайдено.  
*when*(classService.generateStudentReport(999L)).thenThrow(EntityNotFoundException.class);  
  
mockMvc.perform(*get*("/api/classes/get/report/student/{studentId}", 999L)  
 .header("Authorization", "Bearer " + TestObjectFactory.*getAccessToken*())  
 .contentType(MediaType.*APPLICATION\_JSON*))  
 .andExpect(*status*().isNotFound());  
 }  
  
@Test  
public void testGetStudentsInfo\_Success() throws Exception {  
// Перевірка успішного отримання інформації про студентів.  
ClassDto \_class = TestObjectFactory.*createSampleClass*(1L);  
 StudentDto studentDto = TestObjectFactory.*createSampleStudent*(1L, "Jane", "Doe", \_class);  
 studentDto.setParentName("John Doe");  
 studentDto.setParentContactPhone("123-456-7890");  
 studentDto.setParentContactEmail("john.doe@example.com");  
  
 StudentResponseDto responseDto = new StudentResponseDto(studentDto.getFirstName() + " " + studentDto.getLastName(),  
 studentDto.getParentName(),  
 studentDto.getParentContactPhone(),  
 studentDto.getParentContactEmail());  
  
*when*(classService.getStudentsInfo(1L)).thenReturn(Collections.*singletonList*(responseDto));  
  
mockMvc.perform(*get*("/api/classes/get/{id}/studentsInfo", 1L)  
 .header("Authorization", "Bearer " + TestObjectFactory.*getAccessToken*())  
 .contentType(MediaType.*APPLICATION\_JSON*))  
 .andExpect(*status*().isOk())  
 .andExpect(*content*().contentType(MediaType.*APPLICATION\_JSON*))  
 .andExpect(*jsonPath*("$[0].parentName").value("John Doe"))  
 .andExpect(*jsonPath*("$[0].parentContactPhone").value("123-456-7890"))  
 .andExpect(*jsonPath*("$[0].parentContactEmail").value("john.doe@example.com"));  
 }  
  
@Test  
public void testGetStudentsInfo\_NotFound() throws Exception {  
// Перевірка випадку, коли інформацію про студентів не знайдено.  
*when*(classService.getStudentsInfo(999L)).thenThrow(EntityNotFoundException.class);  
  
mockMvc.perform(*get*("/api/classes/get/{id}/studentsInfo", 999L)  
 .header("Authorization", "Bearer " + TestObjectFactory.*getAccessToken*())  
 .contentType(MediaType.*APPLICATION\_JSON*))  
 .andExpect(*status*().isNotFound());  
 }  
  
@Test  
public void testGetAllClasses\_Success() throws Exception {  
// Перевірка успішного отримання всіх класів.  
ClassDto class1 = TestObjectFactory.*createSampleClass*(1L);  
 ClassDto class2 = TestObjectFactory.*createSampleClass*(2L);  
 ClassDto class3 = TestObjectFactory.*createSampleClass*(3L);  
  
 List<ClassDto> classList = Arrays.*asList*(class1, class2, class3);  
  
*when*(classService.getAllClasses()).thenReturn(classList);  
mockMvc.perform(*get*("/api/classes/getAllClasses")  
 .header("Authorization", "Bearer " + TestObjectFactory.*getAccessToken*())  
 .contentType(MediaType.*APPLICATION\_JSON*))  
 .andExpect(MockMvcResultMatchers.*status*().isOk())  
 .andExpect(MockMvcResultMatchers.*jsonPath*("$.size()").value(classList.size()));  
 }  
  
@Test  
public void testGetClassAttendance\_Success() throws Exception {  
// Перевірка успішного отримання списку відвідуваності класу.  
TeacherDto teacherDto = TestObjectFactory.*createSampleTeacher*(1L);  
 ClassDto \_class = TestObjectFactory.*createSampleClass*(1L);  
 StudentDto student1 = TestObjectFactory.*createSampleStudent*(1L,"John", "Smith", \_class);  
 StudentDto student2 = TestObjectFactory.*createSampleStudent*(2L, "Jane", "Doe", \_class);  
 LessonPlanDto lp = TestObjectFactory.*createSampleLessonPlanDto*(1L);  
 LessonDto lesson = TestObjectFactory.*createSampleLesson*(1L, teacherDto, \_class, lp);  
  
 List<AttendanceDto> attendanceDtoList = Arrays.*asList*(  
new AttendanceDto(1L, student1, lesson,true, LocalDate.*of*(2024, 4, 28)),  
new AttendanceDto(2L, student2, lesson,true, LocalDate.*of*(2024, 4, 28))  
 );  
  
*when*(classService.getClassAttendance(1L)).thenReturn(attendanceDtoList);  
  
mockMvc.perform(*get*("/api/classes/get/{classId}/attendance", 1L)  
 .header("Authorization", "Bearer " + TestObjectFactory.*getAccessToken*())  
 .contentType(MediaType.*APPLICATION\_JSON*))  
 .andExpect(*status*().isOk())  
 .andExpect(*content*().contentType(MediaType.*APPLICATION\_JSON*))  
 .andExpect(*jsonPath*("$").isArray())  
 .andExpect(*jsonPath*("$.size()").value(attendanceDtoList.size()));  
 }  
  
@Test  
public void testGetClassAttendance\_NotFound() throws Exception {  
// Перевірка випадку, коли список відвідуваності класу не знайдено.  
*when*(classService.getClassAttendance(999L)).thenThrow(EntityNotFoundException.class);  
  
mockMvc.perform(*get*("/api/classes/get/{classId}/attendance", 999L)  
 .header("Authorization", "Bearer " + TestObjectFactory.*getAccessToken*())  
 .contentType(MediaType.*APPLICATION\_JSON*))  
 .andExpect(*status*().isNotFound());  
 }  
  
@Test  
public void testGetClassMarks\_NotFound() throws Exception {  
// Перевірка випадку, коли оцінки для класу не знайдено.  
*when*(classService.getClassMarks(999L)).thenThrow(EntityNotFoundException.class);  
  
mockMvc.perform(*get*("/api/classes/get/{classId}/marks", 999L)  
 .header("Authorization", "Bearer " + TestObjectFactory.*getAccessToken*())  
 .contentType(MediaType.*APPLICATION\_JSON*))  
 .andExpect(*status*().isNotFound());  
 }  
}

Юніт-тести - це фундаментальний аспект розробки програмного забезпечення, який передбачає тестування окремих одиниць або компонентів програми в ізоляції. Ці "одиниці" можуть бути функціями, методами, класами або модулями. Метою юніт-тестів є перевірка того, що кожна одиниця програми працює так, як очікується, і відповідає своїм специфікаціям.

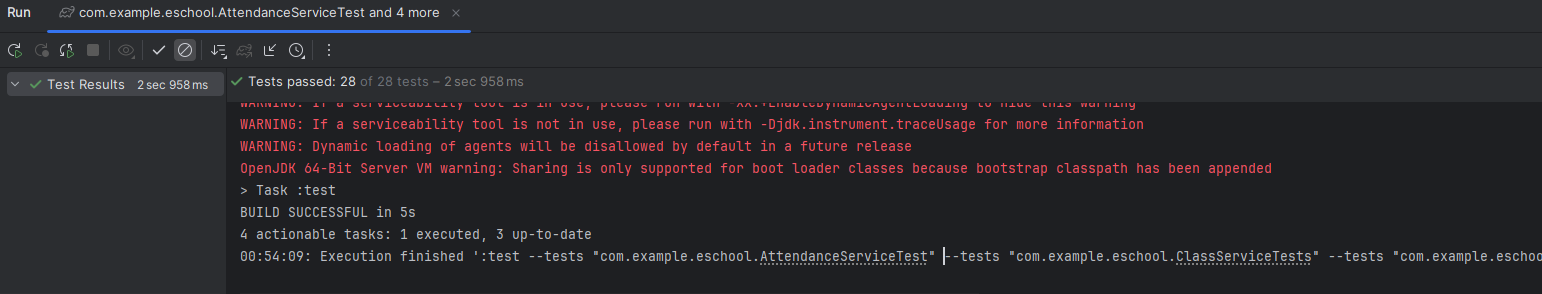


Рисунок 4.5 –Запуск Unit-тестів

Лістинг юніт-тестів надано далі.

AttendanceServiceTests

@ExtendWith(MockitoExtension.class)  
public class AttendanceServiceTest {  
  
@Mock  
private AttendanceRepository attendanceRepository;  
  
@Mock  
private StudentRepository studentRepository;  
  
@Mock  
private LessonRepository lessonRepository;  
  
@InjectMocks  
private AttendanceServiceImpl attendanceService;  
  
@Test  
public void testAddAttendance\_Success() {  
// Mock data  
Long studentId = 123L;  
 Long lessonId = 456L;  
 Boolean presence = true;  
 Student student = new Student();  
 Lesson lesson = new Lesson();  
  
*when*(studentRepository.findById(studentId)).thenReturn(Optional.*of*(student));  
*when*(lessonRepository.findById(lessonId)).thenReturn(Optional.*of*(lesson));  
  
attendanceService.markAttendanceByStudent(studentId, lessonId, presence);  
*verify*(attendanceRepository).save(*any*(Attendance.class));  
 }  
  
@Test  
public void testAddAttendance\_StudentNotFound() {  
 Long studentId = 123L;  
 Long lessonId = 456L;  
 Boolean presence = true;  
  
*when*(studentRepository.findById(studentId)).thenReturn(Optional.*empty*());  
*assertThrows*(EntityNotFoundException.class, () -> {  
attendanceService.markAttendanceByStudent(studentId, lessonId, presence);  
 });  
*verify*(attendanceRepository, *never*()).save(*any*(Attendance.class));  
 }  
  
@Test  
public void testAddAttendance\_LessonNotFound() {  
 Long studentId = 123L;  
 Long lessonId = 456L;  
 Boolean presence = true;  
 Student student = new Student();  
  
*when*(studentRepository.findById(studentId)).thenReturn(Optional.*of*(student));  
*when*(lessonRepository.findById(lessonId)).thenReturn(Optional.*empty*());  
  
*assertThrows*(EntityNotFoundException.class, () -> {  
attendanceService.markAttendanceByStudent(studentId, lessonId, presence);  
 });  
*verify*(attendanceRepository, *never*()).save(*any*(Attendance.class));  
 }  
}

ClassServiceTests

public class ClassServiceTests {  
@Mock  
private ClassRepository classRepository;  
  
@Mock  
private StudentRepository studentRepository;  
  
@Mock  
private AttendanceRepository attendanceRepository;  
  
@Mock  
private MarkRepository markRepository;  
  
@InjectMocks  
private ClassServiceImpl classService;  
  
@BeforeEach  
public void setUp() {  
 MockitoAnnotations.*openMocks*(this);  
 }  
  
@Test  
public void testGetAllClasses() {  
// Given  
List<Subject> subjects = new ArrayList<Subject>();  
 Teacher teacher = new Teacher();  
 teacher.setSubjects(subjects);  
 com.example.eschool.entities.Class class1 = new Class(1L, "className1", teacher, 2012, 2023);  
 Class class2 = new Class(2L, "className2", teacher, 2012, 2023);  
  
*when*(classRepository.findAll()).thenReturn(List.*of*(class1, class2));  
  
// When  
List<ClassDto> result = classService.getAllClasses();  
  
// Then  
*assertEquals*(2, result.size());  
*verify*(classRepository, *times*(1)).findAll();  
 }  
  
@Test  
public void testGetClassById\_Success() {  
// Given  
List<Subject> subjects = new ArrayList<Subject>();  
 Teacher teacher = new Teacher();  
 teacher.setSubjects(subjects);  
 com.example.eschool.entities.Class class1 = new Class();  
 class1.setTeacher(teacher);  
 class1.setId(1L);  
  
*when*(classRepository.findById(1L)).thenReturn(Optional.*of*(class1));  
  
// When  
ClassDto result = classService.getClassById(1L);  
  
// Then  
*assertNotNull*(result);  
*assertEquals*(1L, result.getId());  
*verify*(classRepository, *times*(1)).findById(1L);  
 }  
  
@Test  
public void testGetClassById\_NotFound() {  
// Given  
*when*(classRepository.findById(1L)).thenReturn(Optional.*empty*());  
  
// When & Then  
*assertThrows*(EntityNotFoundException.class, () -> {  
classService.getClassById(1L);  
 });  
*verify*(classRepository, *times*(1)).findById(1L);  
 }  
  
@Test  
public void testGetClassAttendance() {  
// Given  
List<Subject> subjects = new ArrayList<Subject>();  
 Teacher teacher = new Teacher();  
 teacher.setSubjects(subjects);  
 com.example.eschool.entities.Class aClass = new Class();  
 aClass.setTeacher(teacher);  
 aClass.setId(1L);  
*when*(classRepository.findById(1L)).thenReturn(Optional.*of*(aClass));  
 Student student = new Student();  
 student.setId(1L);  
 student.setStudentClass(aClass);  
*when*(studentRepository.findStudentsByStudentClass(aClass)).thenReturn(List.*of*(student));  
 Attendance attendance = new Attendance();  
 attendance.setStudent(student);  
 Lesson lesson = new Lesson();  
 lesson.setClassLesson(aClass);  
 attendance.setLesson(lesson);  
 lesson.setTeacher(teacher);  
  
*when*(attendanceRepository.findAllByStudentId(1L)).thenReturn(List.*of*(attendance));  
  
// When  
List<AttendanceDto> result = classService.getClassAttendance(1L);  
  
// Then  
*assertEquals*(1, result.size());  
*verify*(classRepository, *times*(1)).findById(1L);  
*verify*(studentRepository, *times*(1)).findStudentsByStudentClass(aClass);  
*verify*(attendanceRepository, *times*(1)).findAllByStudentId(1L);  
 }  
  
@Test  
public void testGetClassMarks() {  
// Given  
List<Subject> subjects = new ArrayList<Subject>();  
 Teacher teacher = new Teacher();  
 teacher.setId(1L);  
 teacher.setSubjects(subjects);  
 com.example.eschool.entities.Class aClass = new Class();  
 aClass.setTeacher(teacher);  
*when*(classRepository.findById(1L)).thenReturn(Optional.*of*(aClass));  
 Student student = new Student();  
 student.setId(1L);  
 student.setStudentClass(aClass);  
*when*(studentRepository.findStudentsByStudentClass(aClass)).thenReturn(List.*of*(student));  
 Mark mark = new Mark();  
 mark.setStudent(student);  
 LessonPlan lessonPlan = new LessonPlan();  
 Subject subject = new Subject();  
 List<Assignment> assignments = new ArrayList<Assignment>();  
 List<Material> materials = new ArrayList<Material>();  
 lessonPlan.setSubject(subject);  
 lessonPlan.setAssignments(assignments);  
 lessonPlan.setMaterials(materials);  
 Lesson lesson = new Lesson(1L,aClass, lessonPlan, teacher);  
  
 mark.setLesson(lesson);  
*when*(markRepository.findAllByStudentId(1L)).thenReturn(List.*of*(mark));  
  
// When  
List<MarkDto> result = classService.getClassMarks(1L);  
  
// Then  
*assertEquals*(1, result.size());  
*verify*(classRepository, *times*(1)).findById(1L);  
*verify*(studentRepository, *times*(1)).findStudentsByStudentClass(aClass);  
*verify*(markRepository, *times*(1)).findAllByStudentId(1L);  
 }  
  
@Test  
public void testGenerateStudentReport() {  
// Given  
Student student = new Student();  
 student.setId(1L);  
 student.setFirstName("John");  
 student.setLastName("Doe");  
*when*(studentRepository.findById(1L)).thenReturn(Optional.*of*(student));  
  
 Attendance attendance1 = new Attendance();  
 attendance1.setStatus(false);  
 Attendance attendance2 = new Attendance();  
 attendance2.setStatus(true);  
*when*(attendanceRepository.findAllByStudentId(1L)).thenReturn(List.*of*(attendance1, attendance2));  
  
 Mark mark = new Mark();  
 Lesson lesson = new Lesson();  
 LessonPlan lessonPlan = new LessonPlan();  
 Subject subject = new Subject();  
 subject.setSubjectName("Math");  
 lessonPlan.setSubject(subject);  
 lesson.setLessonPlan(lessonPlan);  
 mark.setLesson(lesson);  
 mark.setMark(10);  
*when*(markRepository.findAllByStudentId(1L)).thenReturn(List.*of*(mark));  
  
// When  
StudentReportDto result = classService.generateStudentReport(1L);  
  
// Then  
*assertNotNull*(result);  
*assertEquals*("John Doe", result.getStudentName());  
*assertEquals*(1, result.getMissedLessons());  
*assertEquals*(1, result.getAttendedLessons());  
*assertEquals*(1, result.getAverageMarksBySubject().size());  
*assertEquals*(10.0, result.getAverageMarksBySubject().get("Math"));  
*assertEquals*(10.0, result.getAverage());  
 }  
  
@Test  
public void testDeleteClassById() {  
// Given  
Long classId = 1L;  
  
// When  
classService.deleteClassById(classId);  
  
// Then  
*verify*(classRepository, *times*(1)).deleteById(classId);  
 }  
}

LessonPlanTests

@ExtendWith(MockitoExtension.class)  
public class LessonPlanServiceTest {  
  
@Mock  
private LessonPlanRepository lessonPlanRepository;  
  
@Mock  
private AssignmentRepository assignmentRepository;  
  
@Mock  
private MaterialRepository materialRepository;  
  
@Mock  
private SubjectRepository subjectRepository;  
  
@InjectMocks  
private LessonPlanServiceImpl lessonPlanService;  
  
private LessonPlan lessonPlan;  
private LessonPlanDto lessonPlanDto;  
private Subject subject;  
private Assignment assignment;  
private Material material;  
  
@BeforeEach  
public void setUp() {  
lessonPlan = new LessonPlan();  
lessonPlan.setLessonId(1L);  
lessonPlan.setTopic("Math");  
lessonPlan.setDescription("Math lesson plan");  
  
subject = new Subject();  
subject.setSubjectId(1L);  
subject.setSubjectName("Mathematics");  
lessonPlan.setSubject(subject);  
  
assignment = new Assignment();  
assignment.setId(1L);  
assignment.setName("Homework");  
assignment.setType("Homework");  
assignment.setMaxMark(10);  
  
material = new Material();  
material.setId(1L);  
material.setDescription("Textbook");  
material.setFileURL("http://example.com");  
  
lessonPlan.setAssignments(List.*of*(assignment));  
lessonPlan.setMaterials(List.*of*(material));  
  
lessonPlanDto = LessonPlanMapper.*mapToLessonPlanDto*(lessonPlan);  
 }  
  
@Test  
public void testCreateLessonPlan() {  
*when*(subjectRepository.findById(*anyLong*())).thenReturn(Optional.*of*(subject));  
*when*(assignmentRepository.findById(*anyLong*())).thenReturn(Optional.*of*(assignment));  
*when*(materialRepository.findById(*anyLong*())).thenReturn(Optional.*of*(material));  
*when*(lessonPlanRepository.save(*any*(LessonPlan.class))).thenReturn(lessonPlan);  
  
 LessonPlanDto result = lessonPlanService.createLessonPlan(  
"Math", "Math lesson plan", 1L, List.*of*(1L), List.*of*(1L));  
  
*assertNotNull*(result);  
*assertEquals*("Math", result.getTopic());  
*verify*(subjectRepository, *times*(1)).findById(1L);  
*verify*(assignmentRepository, *times*(1)).findById(1L);  
*verify*(materialRepository, *times*(1)).findById(1L);  
*verify*(lessonPlanRepository, *times*(1)).save(*any*(LessonPlan.class));  
 }  
  
@Test  
public void testGetLessonPlanById() {  
*when*(lessonPlanRepository.findById(*anyLong*())).thenReturn(Optional.*of*(lessonPlan));  
  
 LessonPlanDto result = lessonPlanService.getLessonPlanById(1L);  
  
*assertNotNull*(result);  
*assertEquals*("Math", result.getTopic());  
*verify*(lessonPlanRepository, *times*(1)).findById(1L);  
 }  
  
@Test  
public void testGetLessonPlanById\_NotFound() {  
*when*(lessonPlanRepository.findById(*anyLong*())).thenReturn(Optional.*empty*());  
  
 RuntimeException thrown = *assertThrows*(RuntimeException.class, () -> {  
lessonPlanService.getLessonPlanById(1L);  
 });  
  
*assertEquals*("Failed to get lesson plan by ID: 1", thrown.getMessage());  
*verify*(lessonPlanRepository, *times*(1)).findById(1L);  
 }  
  
@Test  
public void testGetAllLessonPlans() {  
*when*(lessonPlanRepository.findAll()).thenReturn(List.*of*(lessonPlan));  
  
 List<LessonPlanDto> result = lessonPlanService.getAllLessonPlans();  
  
*assertNotNull*(result);  
*assertEquals*(1, result.size());  
*assertEquals*("Math", result.get(0).getTopic());  
*verify*(lessonPlanRepository, *times*(1)).findAll();  
 }  
  
@Test  
public void testUpdateLessonPlan() {  
*when*(lessonPlanRepository.findById(*anyLong*())).thenReturn(Optional.*of*(lessonPlan));  
*when*(subjectRepository.findById(*anyLong*())).thenReturn(Optional.*of*(subject));  
*when*(assignmentRepository.findById(*anyLong*())).thenReturn(Optional.*of*(assignment));  
*when*(materialRepository.findById(*anyLong*())).thenReturn(Optional.*of*(material));  
*when*(lessonPlanRepository.save(*any*(LessonPlan.class))).thenReturn(lessonPlan);  
  
 LessonPlanDto result = lessonPlanService.updateLessonPlan(  
1L, "Math", "Updated Math lesson plan", 1L, List.*of*(1L), List.*of*(1L));  
  
*assertNotNull*(result);  
*assertEquals*("Updated Math lesson plan", result.getDescription());  
*verify*(lessonPlanRepository, *times*(1)).findById(1L);  
*verify*(subjectRepository, *times*(1)).findById(1L);  
*verify*(assignmentRepository, *times*(1)).findById(1L);  
*verify*(materialRepository, *times*(1)).findById(1L);  
*verify*(lessonPlanRepository, *times*(1)).save(*any*(LessonPlan.class));  
 }  
  
@Test  
public void testDeleteLessonPlan() {  
*when*(lessonPlanRepository.existsById(*anyLong*())).thenReturn(true);  
*doNothing*().when(lessonPlanRepository).deleteById(*anyLong*());  
  
lessonPlanService.deleteLessonPlan(1L);  
  
*verify*(lessonPlanRepository, *times*(1)).existsById(1L);  
*verify*(lessonPlanRepository, *times*(1)).deleteById(1L);  
 }  
  
@Test  
public void testDeleteLessonPlan\_NotFound() {  
*when*(lessonPlanRepository.existsById(*anyLong*())).thenReturn(false);  
  
 RuntimeException thrown = *assertThrows*(RuntimeException.class, () -> {  
lessonPlanService.deleteLessonPlan(1L);  
 });  
  
*assertEquals*("Failed to delete lesson plan", thrown.getMessage());  
*verify*(lessonPlanRepository, *times*(1)).existsById(1L);  
 }  
}

MarkServiceTests

public class MarkServiceTests {  
  
@Mock  
private StudentRepository studentRepository;  
  
@Mock  
private LessonRepository lessonRepository;  
  
@Mock  
private MarkRepository markRepository;  
  
@InjectMocks  
private MarkServiceImpl markService;  
  
@BeforeEach  
public void setUp() {  
 MockitoAnnotations.*openMocks*(this);  
 }  
  
@Test  
public void testGetMarkById\_Success() {  
// Given  
Long markId = 1L;  
 Mark mark = new Mark();  
 mark.setId(markId);  
 mark.setMark(10);  
 Class aclass = new Class();  
 List<Subject> subjects = new ArrayList<>();  
 Teacher teacher = new Teacher();  
 teacher.setSubjects(subjects);  
 aclass.setTeacher(teacher);  
 Student student = new Student();  
 student.setStudentClass(aclass);  
 Lesson lesson = new Lesson();  
 lesson.setClassLesson(aclass);  
 lesson.setTeacher(teacher);  
 LessonPlan lessonPlan = new LessonPlan();  
 Subject subject = new Subject();  
 List<Assignment> assignments = new ArrayList<Assignment>();  
 List<Material> materials = new ArrayList<Material>();  
 lessonPlan.setSubject(subject);  
 lessonPlan.setAssignments(assignments);  
 lessonPlan.setMaterials(materials);  
 lesson.setLessonPlan(lessonPlan);  
 mark.setLesson(lesson);  
 mark.setStudent(student);  
*when*(markRepository.findById(markId)).thenReturn(Optional.*of*(mark));  
  
// When  
MarkDto result = markService.getMarkById(markId);  
  
// Then  
*assertEquals*(mark.getId(), result.getId());  
*assertEquals*(mark.getMark(), result.getMark());  
 }  
  
@Test  
public void testGetMarkById\_NotFound() {  
// Given  
Long markId = 1L;  
*when*(markRepository.findById(markId)).thenReturn(Optional.*empty*());  
  
// Then  
*assertThrows*(RuntimeException.class, () ->markService.getMarkById(markId));  
 }  
  
@Test  
public void testGetMarksByStudentId\_Success() {  
// Given  
Long studentId = 1L;  
 Class aclass = new Class();  
 List<Subject> subjects = new ArrayList<>();  
 Teacher teacher = new Teacher();  
 teacher.setSubjects(subjects);  
 aclass.setTeacher(teacher);  
 Student student = new Student();  
 student.setStudentClass(aclass);  
 Lesson lesson = new Lesson();  
 lesson.setClassLesson(aclass);  
 lesson.setTeacher(teacher);  
 LessonPlan lessonPlan = new LessonPlan();  
 Subject subject = new Subject();  
 List<Assignment> assignments = new ArrayList<Assignment>();  
 List<Material> materials = new ArrayList<Material>();  
 lessonPlan.setSubject(subject);  
 lessonPlan.setAssignments(assignments);  
 lessonPlan.setMaterials(materials);  
 lesson.setLessonPlan(lessonPlan);  
 Mark mark1 = new Mark();  
 mark1.setId(1L);  
 mark1.setLesson(lesson);  
 mark1.setStudent(student);  
 Mark mark2 = new Mark();  
 mark2.setId(2L);  
 mark2.setStudent(student);  
 mark2.setLesson(lesson);  
*when*(markRepository.findAllByStudentId(studentId)).thenReturn(List.*of*(mark1, mark2));  
  
// When  
List<MarkDto> result = markService.getMarksByStudentId(studentId);  
  
// Then  
*assertEquals*(2, result.size());  
 }  
  
@Test  
public void testMarkStudent\_Success() {  
// Given  
Long studentId = 1L;  
 Long lessonId = 1L;  
 Integer markValue = 10;  
 Student student = new Student();  
 student.setId(studentId);  
 Lesson lesson = new Lesson();  
 lesson.setLessonId(lessonId);  
*when*(studentRepository.findById(studentId)).thenReturn(Optional.*of*(student));  
*when*(lessonRepository.findById(lessonId)).thenReturn(Optional.*of*(lesson));  
  
// When  
markService.markStudent(studentId, lessonId, markValue);  
  
// Then  
*verify*(markRepository, *times*(1)).save(*any*(Mark.class));  
 }  
  
@Test  
public void testMarkStudent\_InvalidMarkValue() {  
// Given  
Long studentId = 1L;  
 Long lessonId = 1L;  
 Integer invalidMarkValue = 15;  
  
// Then  
*assertThrows*(IllegalArgumentException.class, () ->markService.markStudent(studentId, lessonId, invalidMarkValue));  
 }  
  
@Test  
public void testMarkStudent\_StudentNotFound() {  
// Given  
Long studentId = 1L;  
 Long lessonId = 1L;  
 Integer markValue = 10;  
*when*(studentRepository.findById(studentId)).thenReturn(Optional.*empty*());  
  
// Then  
*assertThrows*(EntityNotFoundException.class, () ->markService.markStudent(studentId, lessonId, markValue));  
 }  
  
@Test  
public void testMarkStudent\_LessonNotFound() {  
// Given  
Long studentId = 1L;  
 Long lessonId = 1L;  
 Integer markValue = 10;  
 Student student = new Student();  
 student.setId(studentId);  
*when*(studentRepository.findById(studentId)).thenReturn(Optional.*of*(student));  
*when*(lessonRepository.findById(lessonId)).thenReturn(Optional.*empty*());  
  
// Then  
*assertThrows*(EntityNotFoundException.class, () ->markService.markStudent(studentId, lessonId, markValue));  
 }  
  
}