

КУРСОВА РОБОТА

з дисципліни

«Конструювання програмного 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

Почапський Денис Ігорович

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 дебіюрократизації, 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 кожен user story в окремій таблиці:

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	<p>У системі має бути можливість створення окремого електронного журналу для кожного предмету, що веде вчитель.</p> <p>Для кожного уроку або заняття має бути можливість введення оцінок за різні види оцінювання, такі як завдання, тести, контрольні роботи тощо.</p> <p>Система повинна забезпечувати можливість введення числових оцінок.</p> <p>Вчителю має бути доступний перегляд електронного журналу з оцінками кожного учня для кожного предмету.</p>

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.

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.

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.

1.1.1. 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 має наступні 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 сутності діють наступні бізнес-правила:

- ідентифікатор вчителя унікально ідентифікує його, оскільки не може бути більше одного вчителя з однаковим ідентифікатором.
- усі атрибути є обов'язковими.

1.1.2. 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 має наступні зв'язки з іншими сутностями:

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

Бізнес-правила. Відносно сутності діють 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.

1.1.3. Сутність Предмет

Короткий опис сутності. Представляє інформацію про кожний навчальний предмет.

Атрибути. Сутність характеризується наступними атрибутами:

- ідентифікатор;
- назва;
- вчитель.

Зв'язки. Сутність має наступні зв'язки з іншими сутностями:

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

Бізнес-правила. Відносно сутності діють наступні бізнес-правила:

- ідентифікатор предмету унікально ідентифікує його.
- усі атрибути, є обов'язковими.

1.1.4. 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 має наступні 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 сутності діють наступні бізнес-правила:

- ідентифікатор уроку має бути унікальним для кожного року навчання;
- всі атрибути є обов'язковими.

1.1.5. 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 має наступні зв'язки з іншими сутностями:

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

Бізнес-правила. Відносно сутності діють наступні бізнес-правила:

- номер класу має бути унікальним для кожного року навчання;
- всі атрибути є обов'язковими.

1.1.6. 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 має наступні зв'язки з іншими сутностями:

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

Бізнес-правила. Відносно сутності діють 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.

1.1.7. 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 має наступні 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 сутності діють наступні бізнес-правила:

- всі атрибути є обов'язковими.

1.1.8. Сутність План уроку

Короткий опис сутності. Представляє інформацію про кожний план уроку.

Атрибути. Сутність характеризується наступними атрибутами:

- ідентифікатор;
- назва;
- опис;
- предмет.

Зв'язки. Сутність має наступні 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 сутності діють наступні бізнес-правила:

- атрибути ідентифікатор, назва, предмет є обов'язковими.

1.1.9. 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 має наступні зв'язки з іншими сутностями:

- завдання можуть відноситися до кількох планів уроків.

Бізнес-правила. Відносно сутності діють наступні бізнес-правила:

- ідентифікатор завдання має бути унікальним;
- всі атрибути є обов'язковими.

1.1.10. 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 має наступні зв'язки з іншими сутностями:

- матеріал може бути використаний у багатьох різних планах уроків.

Бізнес-правила. Відносно сутності діють 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:

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)

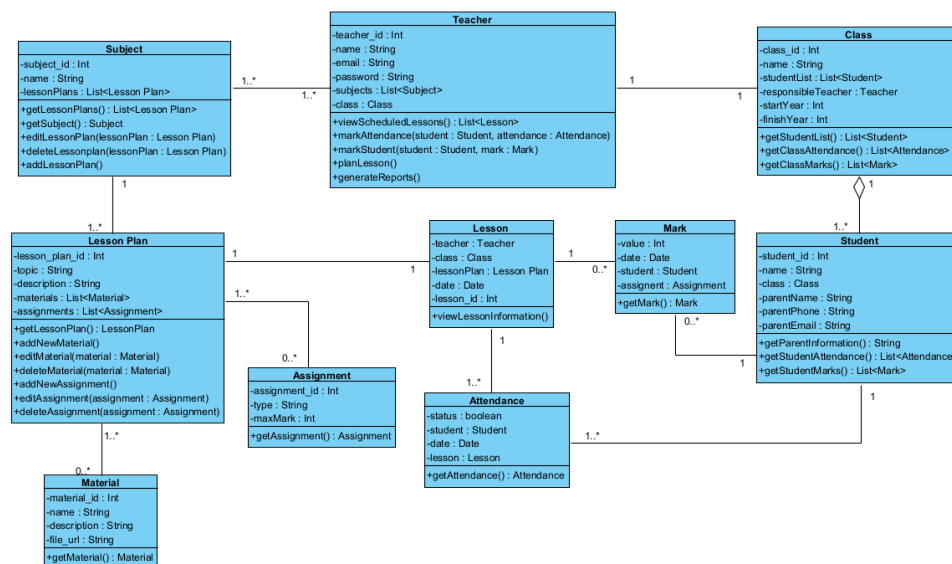


Рисунок 1.6 – Діаграма класів

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.

```
CREATE TABLE Assignments (  
  assignment_id SERIAL NOT NULL,  
  assignment_type char(255) NOT NULL,  
  name char(255) NOT NULL,  
  max_mark int4 NOT NULL,  
  PRIMARY KEY (assignment_id));  
CREATE TABLE Attendance (student_id int4 NOT NULL, "date" date NOT NULL, status bool NOT NULL, lesson_id int4 NOT NULL);  
CREATE TABLE Classes (  
  class_id SERIAL NOT NULL,  
  teacher_id int4 NOT NULL,  
  name char(255) NOT NULL,  
  year_start int4 NOT NULL,  
  year_finish int4 NOT NULL,  
  PRIMARY KEY (class_id));  
CREATE TABLE Lesson (  
  class_id int4 NOT NULL,  
  teacher_id int4 NOT NULL,  
  lesson_id SERIAL NOT NULL,  
  lesson_plan_id int4 NOT NULL,  
  PRIMARY KEY (lesson_id));  
CREATE TABLE Lesson_Assignment (lesson_plan_id int4 NOT NULL, assignment_id int4 NOT NULL, PRIMARY KEY (lesson_plan_id, assignment_id));  
CREATE TABLE Lesson_Plan (  
  lesson_plan_id SERIAL NOT NULL,  
  topic char(255) NOT NULL,  
  description char(255),  
  subject_id int4 NOT NULL,  
  PRIMARY KEY (lesson_plan_id));  
CREATE TABLE Lesson_Plan_Materials (lesson_plan_id int4 NOT NULL, material_id int4 NOT NULL, PRIMARY KEY (lesson_plan_id, material_id));  
CREATE TABLE Mark ("date" date NOT NULL, student_id int4 NOT NULL, mark int4 NOT NULL, lesson_id int4 NOT NULL);  
CREATE TABLE Materials (  
  material_id SERIAL NOT NULL,  
  name char(255) NOT NULL,  
  description char(255),  
  file_url char(255) NOT NULL,  
  PRIMARY KEY (material_id));
```

```

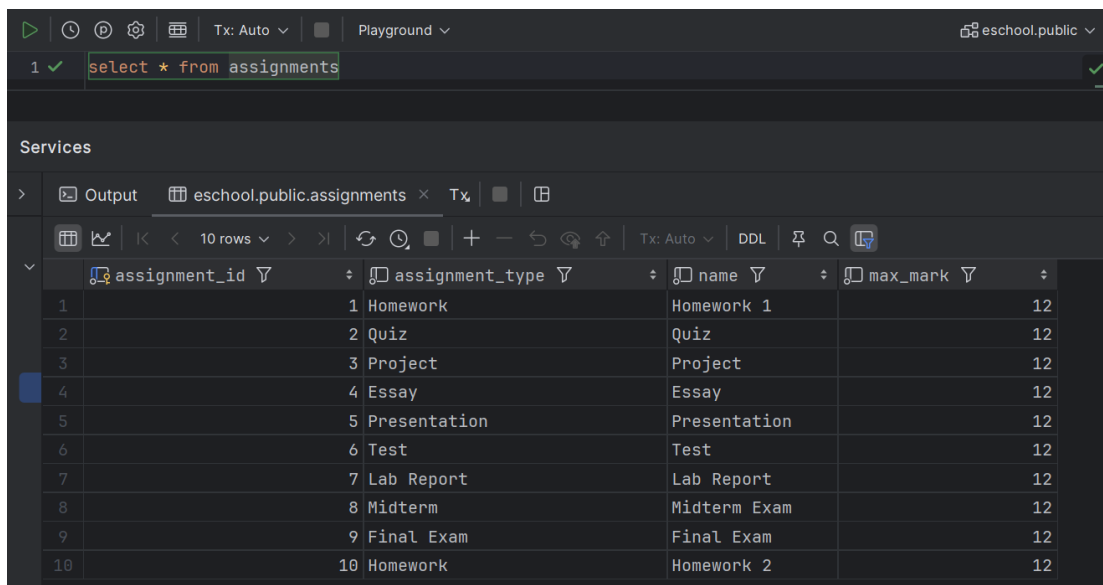
CREATE TABLE Student (
    student_id SERIAL NOT NULL,
    "first name" char(255) NOT NULL,
    "last name" char(255) NOT NULL,
    class_id int4 NOT NULL,
    parent_name char(255) NOT NULL,
    parent_contact_phone char(255) NOT NULL,
    parent_contact_email char(255),
    PRIMARY KEY (student_id));
CREATE TABLE Subject (subject_id SERIAL NOT NULL, subject_name char(255) NOT NULL, PRIMARY KEY (subject_id));
CREATE TABLE Teacher (
    teacher_id SERIAL NOT NULL,
    first_name char(255) NOT NULL,
    last_name char(255) NOT NULL,
    email char(255) NOT NULL,
    password char(255) NOT NULL,
    PRIMARY KEY (teacher_id));
CREATE TABLE Teacher_Subject (teacher_id int4 NOT NULL, subject_id int4 NOT NULL, PRIMARY KEY (teacher_id, subject_id));

ALTER TABLE Lesson ADD CONSTRAINT FK_Lesson_LessonPlan FOREIGN KEY (lesson_plan_id) REFERENCES Lesson_Plan (lesson_plan_id);
ALTER TABLE Lesson ADD CONSTRAINT FK_Lesson_Teacher FOREIGN KEY (teacher_id) REFERENCES Teacher (teacher_id);
ALTER TABLE Mark ADD CONSTRAINT FK_Mark_Student FOREIGN KEY (student_id) REFERENCES Student (student_id);
ALTER TABLE Attendance ADD CONSTRAINT FK_Attendance_Student FOREIGN KEY (student_id) REFERENCES Student (student_id);
ALTER TABLE Lesson ADD CONSTRAINT FK_Lesson_Class FOREIGN KEY (class_id) REFERENCES Classes (class_id);
ALTER TABLE Mark ADD CONSTRAINT FK_Mark_Lesson FOREIGN KEY (lesson_id) REFERENCES Lesson (lesson_id);
ALTER TABLE Attendance ADD CONSTRAINT FK_Attendance_Lesson FOREIGN KEY (lesson_id) REFERENCES Lesson (lesson_id);
ALTER TABLE Lesson_Assignment ADD CONSTRAINT FK_Lesson_Assignment FOREIGN KEY (assignment_id) REFERENCES Assignments (assignment_id);
ALTER TABLE Lesson_Assignment ADD CONSTRAINT FK_Lesson_Assignment_Plan FOREIGN KEY (lesson_plan_id) REFERENCES Lesson_Plan (lesson_plan_id);
ALTER TABLE Lesson_Plan_Materials ADD CONSTRAINT FK_LPM_Material FOREIGN KEY (material_id) REFERENCES Materials (material_id);
ALTER TABLE Lesson_Plan_Materials ADD CONSTRAINT FK_LPM_LessonPlan FOREIGN KEY (lesson_plan_id) REFERENCES Lesson_Plan (lesson_plan_id);
ALTER TABLE Teacher_Subject ADD CONSTRAINT FK_TS_Subject FOREIGN KEY (subject_id) REFERENCES Subject (subject_id);
ALTER TABLE Teacher_Subject ADD CONSTRAINT FK_TS_Teacher FOREIGN KEY (teacher_id) REFERENCES Teacher (teacher_id);
ALTER TABLE Student ADD CONSTRAINT FK_Student_Class FOREIGN KEY (class_id) REFERENCES Classes (class_id);
ALTER TABLE Lesson_Plan ADD CONSTRAINT FK_LessonPlan_Subject FOREIGN KEY (subject_id) REFERENCES Subject (subject_id);
ALTER TABLE Classes ADD CONSTRAINT FK_Class_Teacher FOREIGN KEY (teacher_id) REFERENCES Teacher (teacher_id);

```

Рисунок 2.1 – Створення таблиць БД

Після цього заповнимо створені таблиці даними. Для цього частково згенеруємо дані за допомогою сервісу [Mockaroo](#).



The screenshot shows a database playground interface. At the top, there's a toolbar with icons for running queries, undo, redo, and other functions. Below the toolbar, a SQL query is entered: `select * from assignments`. The results are displayed in a table with 10 rows. The table has four columns: `assignment_id`, `assignment_type`, `name`, and `max_mark`. The data is as follows:

assignment_id	assignment_type	name	max_mark
1	Homework	Homework 1	12
2	Quiz	Quiz	12
3	Project	Project	12
4	Essay	Essay	12
5	Presentation	Presentation	12
6	Test	Test	12
7	Lab Report	Lab Report	12
8	Midterm	Midterm Exam	12
9	Final Exam	Final Exam	12
10	Homework	Homework 2	12

Рисунок 2.2 – Заповнена даними таблиця Classes

Рисунок 2.3 – Заповнена даними таблиця Lesson

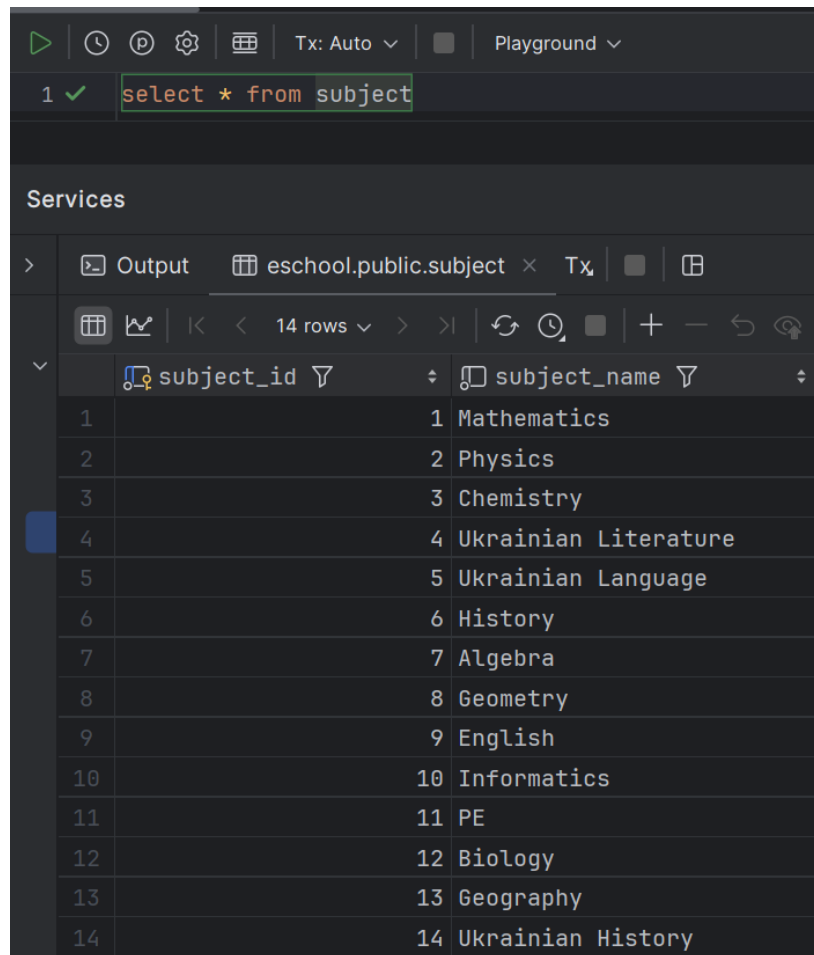
1 ✓ `select * from lesson_assignment`

Services

Output eschool.public.lesson_assignment Tx

	lesson_plan_id	assignment_id	
1	1	1	1
2	2	2	2
3	3	3	3
4	4	4	4
5	5	5	5
6	6	6	6
7	7	7	7
8	8	8	8
9	9	9	9
10	10	10	10
11	11	11	1
12	12	12	2
13	13	13	3
14	14	14	4
15	15	15	5

Рисунок 2.4 – Заповнена даними таблиця Lesson_Assignment



2.2 Створення Spring Boot Application

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

Рисунок 2.15 – Генерація стартового коду

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

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;
}

@AllArgsConstructor

@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);  
}
```

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);  
}
```

LessonServiceImpl

```
@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");
    }
}

```

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
@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
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;

/**
```

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

3.1 Підключення системи авторизації OAuth2

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 проекту відкривається сторінка з логіном.

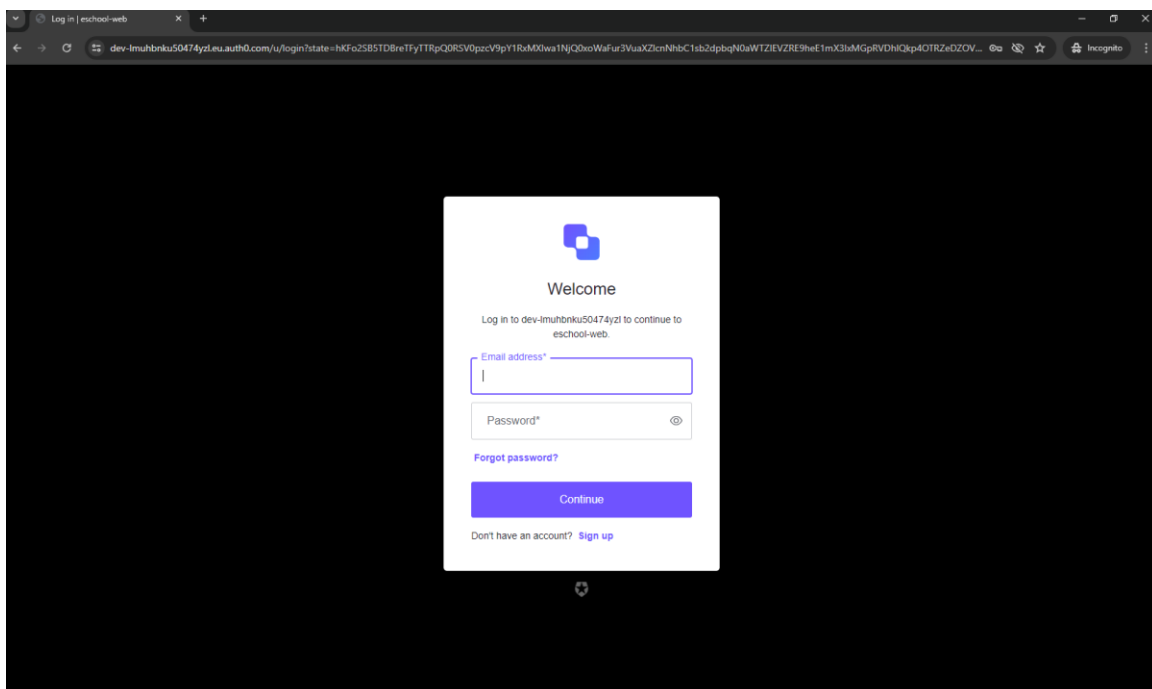


Рисунок 3.5 – Екран входу в систему

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

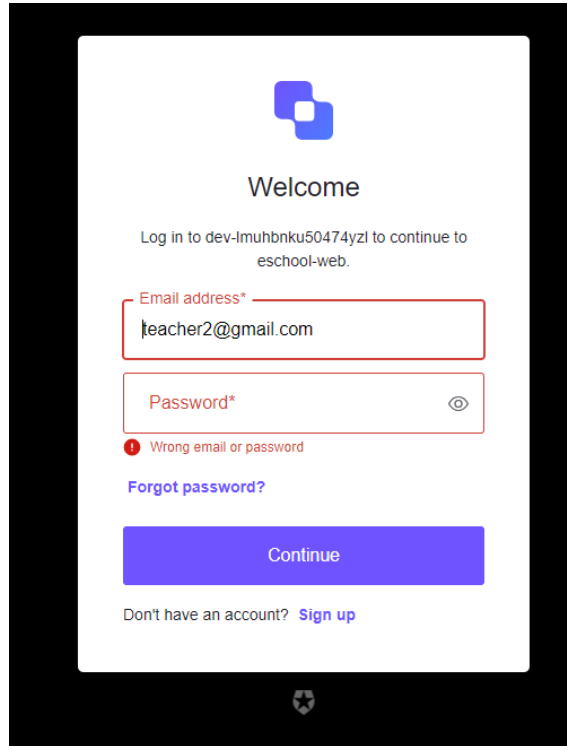


Рисунок 3.6 – Неправильні дані для входу

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.



Authorize App



Hi teacher2@gmail.com,
eschool-web is requesting access to your
dev-lmuhbnku50474yzi account.

- **Profile:** access to your profile and email

Decline

Accept

