ОДЕСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ ІМЕНІ І. І. МЕЧНИКОВА

(повне найменування закладу вищої освіти)

Факультет математики, фізики та інформаційних технологій

(повне найменування факультету)

Кафедра комп'ютерних систем та технологій

(повна назва кафедри)

Кваліфікаційна робота

на здобуття ступеня вищої освіти «бакалавр»

«Розробка інформаційної системи електронного обліку студентів та їх успішності»

(тема кваліфікаційної роботи українською мовою)

«Development of an information system for electronic accounting of students and their academic performance»

(тема кваліфікаційної роботи англійською мовою)

Виконав: здобувач денної форми навчання
спеціальності 123 – Комп'ютерна інженерія
(код, назва спеціальності)
Освітня програма «Комп'ютерна інженерія»
(назва)
Шух Микита Сергійович
(прізвище, ім'я, по-батькові здобувача)

Керівник д.т.н, проф. Михайленко В.С. _____
(науковий ступінь, вчене звання, прізвище, ініціали) (підпис)

Рецензент доц. Чепок А.О.
(науковий ступінь, вчене звання, прізвище, ініціали)

Рекомендовано до захисту:
Протокол засідання кафедри
_____
№ ___ від ____.____. 20___ р.

Завідувач(ка) кафедри
_____ Гунченко Ю.
(підпис) (прізвище, ім'я)

Захищено на засіданні ЕК № _____
протокол № __від ____.____.20___ р.

Оцінка_____/___ ___/_____
(за національною шкалою/шкалою ECTS/ бали)
Голова ЕК
_____ _____
(підпис) (прізвище, ім'я)

Одеса 2024

АНОТАЦІЯ

Мета даної кваліфікаційної роботи – проєктування та реалізація інформаційної системи обліку студентів та їх успішності.

Користувачами даної системи є адміністратори, викладачі, а також студенти учбового порталу.

Реалізація серверної частини виконана за допомогою мови програмування Java у зв’язці з фреймворком Spring Boot, Spring Data JPA та СУБД PostgresSQL. Архітектура системи відповідає шаблону MVC. Клієнтська частина виконана за допомогою мови програмування JavaScript у зв'язці з фреймворком React.

У розробленій системі передбачена можливість керування інформацією про всіх користувачів, академічних структур, предметів та оцінок.

В інформаційній системі реалізовано захист від несанкціонованого доступу та розділ повноважень для різних категорій користувачів.

Результатом кваліфікаційної роботи є інформаційна система учбового порталу з мінімалістичним та зручним користувацьким інтерфейсом.

ANNOTATION

The purpose of this qualification work is the design and implementation of an information system for recording students and their performance.

Users of this system are administrators, teachers, and students of the educational portal.

The server part is implemented using the Java programming language in conjunction with the Spring Boot framework, Spring Data JPA and the PostgresSQL DBMS. The system architecture follows the MVC pattern. The client part is made using the JavaScript programming language in connection with the React framework.

The developed system provides the ability to manage information about all users, academic structures, subjects and grades.

The information system implements protection against unauthorized access and the division of powers for different categories of users.

The result of the qualification work is an educational portal information system with a minimalistic and convenient user interface.

ЗМІСТ

ВСТУП

Університети та навчальні заклади потребують ефективних інструментів для обліку студентів та їх успішності. За відсутності можливості очного навчання та з зростанням потреби в онлайн-освіті виникають серйозні проблеми обміну інформацією між студентами та викладачами. Нагальною є потреба у передачі різноманітного контенту: конспектів лекцій, аудіо- та відеоматеріалів, завдань для вирішення в аудиторіях, самостійної та індивідуальної роботи, інструкцій щодо їх виконання, бази нормативної інформації [1]. Відсутність платформи для двостороннього спілкування стає критичною та уповільнює освітні процеси.

Зважаючи на значний обсяг даних та розгалуженість, а також враховуючи застосування неефективних сервісів, наприклад, звичайних поштових програм, студенти та викладачі не можуть забезпечити швидкий обмін даними та належне структурування інформації, що обмежує можливості їх освітніх процесів [2]. Вже існуючі сервіси мають ряд недоліків та обмежень, які можуть спричинити деякі труднощі під час їх використання. Тому виправдано необхідність створення програмного сервісу, який спростить процес роботи та вирішить зазначені проблеми.

Мета даної кваліфікаційної роботи є проєктування і реалізація інформаційної системи обліку студентів та їх успішності, де викладачі зможуть публікувати завдання, оцінювати роботи студентів та переглядати рейтинговий бал за предметом, а студенти здавати роботи та переглядати їх розклад.

Для досягнення цієї мети необхідно розв’язати наступні задачі:

– проаналізувати предметну область, виокремити користувачів системи;

– розробити базу даних;

– розробити клієнтську частину застосунку;

– розробити серверну частину застосунку;

– протестувати програмний застосунок.

1 ПОСТАНОВКА ЗАДАЧІ

1.1 Задачі інформаційної системи

До основних задач інформаційної системи відносяться наступні:

– облік студентів та оцінка їх успішності;

– додавання, редагування та видалення академічних структур, користувачів та предметів;

– обмін файлами між студентами та викладачами;

– створення та оновлення розкладу для навчальних груп.

1.2. Типи користувачів ІС

В інформаційній системі існують три категорії користувачів, кожна з яких має свої унікальні можливості та вимоги до системи. Категорії користувачів представлені нижче:

– адміністратор – додавати, видаляти чи редагувати користувачів, факультети, спеціальності, навчальні групи, предмети. Створювати та оновлювати розклад навчальних груп. Переглядати успішність усіх навчальних груп за предметами;

– викладач – створювати та видаляти завдання та повідомлення, оцінювати роботи студентів. Переглядати успішність навчальної групи за предметом.

– студент – перегляд розкладу, предметів, завдань та повідомлень. Прикріплювати виконані роботи.

Задачі користувачів ІС обліку студентів та їх успішності, із вказанням вхідних та вихідних даних перелічені у додатку А.

2 ПРОЕКТУВАННЯ ІНФОРМАЦІЙНОЇ СИСТЕМИ

ІС дає можливість користувачам обмінюватись файлами та повідомленнями, тому доречніше розповсюджувати систему через інтернет у вигляді вебсайту, аніж десктоп-додатку. Тому обрано трирівневу архітектуру з застосуванням RESTful API, що є класичним рішенням при створенні вебресурсу.

RESTful API (Representational State Transfer) — це архітектурний стиль для розробки мережевих програм. Зазвичай він використовує стандартні методи HTTP, такі як GET, POST, PUT, DELETE для виконання операцій CRUD (створення, читання, оновлення, видалення) над ресурсами [3].

Класична трирівнева архітектура передбачає, що людина зайде на ресурс і через браузер (рівень представлення даних) передасть запит до сервера додатків (рівень прикладного компонента), де клієнтський запит буде оброблено і передано у вигляді SQL запиту до СУБД (рівня управління ресурсами). Потім запит буде повернено до сервера додатків, що обробить запит від БД, і сформує відповідь, що буде відправлено до клієнта. Цей процес зображено на рис. 2.1.

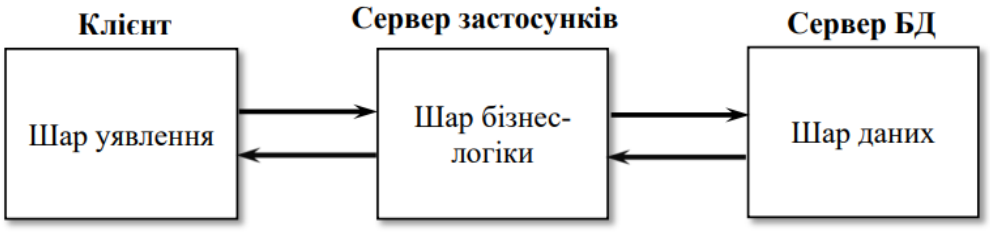

Рисунок 2.1 – Трирівнева клієнт-серверна архітектура

Ця архітектура дозволяє відокремити клієнта, сервер та базу даних. Вона має наступні переваги:

– цілісність даних;

– захищеність бази даних порівняно з дворівневою архітектурою;

– спрощений механізм обміну даними між клієнтом та сервером завдяки стандартизованому інтерфейсу (RESTful API),

– В якості шаблону проєктування обрано MVC (Model-View-Controller). Цей шаблон передбачає, що користувач бачить інтерфейс (View) і взаємодіє з ним. Ця взаємодія передається до контролера (Controller), що змінює модель (Model), яка, в свою чергу, змінює користувацький інтерфейс.

Схема роботи шаблону MVC наведено на рис. 2.2.

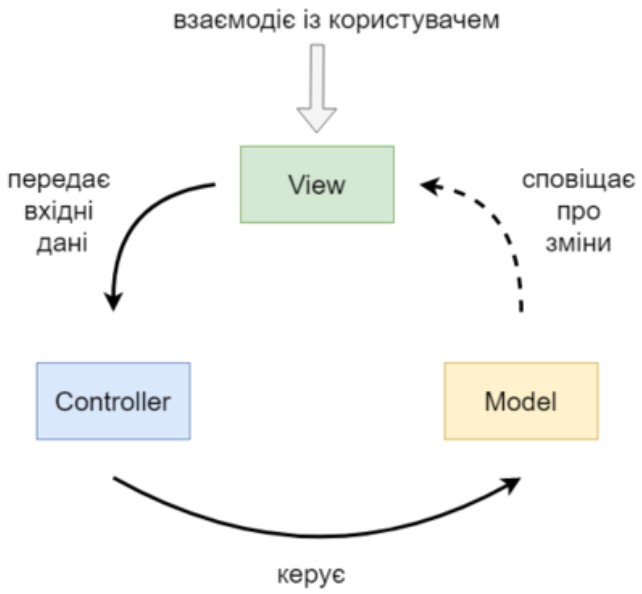

Рисунок 2.2 – Схема взаємодії користувацького інтерфейсу, контролера та моделі

Можна виділити наступні переваги використання шаблону MVC:

– поділ функціональності додатку, що спрощує роботу над великими проєктами;

– легкість виведення різних уявлень для різних типів пристроїв, при цьому використовуючи спільні дані;

– спрощення підтримки та тестування коду.

3 ІНФОРМАЦІЙНА МОДЕЛЬ ПРЕДМЕТНОЇ ОБЛАСТІ

В інформаційній системі освітнього порталу слід виділити наступні сутності:

– member – це сутність, що становить собою користувача. Містить номер користувача, логін (email), хешований пароль, роль користувача;

– personal_data – це сутність, що описує особисті дані користувача. Містить номер даних користувача, номер користувача до якого відносяться дані, ім'я, прізвище та по-батькові;

– faculty – це сутність, що становить собою факультет. Містить номер факультету та його назву;

– major – це сутність, що описує спеціальність. Містить код спеціальності, назву та номер факультета, до якого вона належить;

– study_group – це сутність, що становить собою навчальну групу. Містить номер навчальної групи, назву, код спеціальності та рік навчання;

– teacher – це сутність, що становить собою викладача. Має номер особистих даних, що належать даному користувачу;

– student – це сутність, що становить собою студента. Має номер особистих даних, що належать даному користувачу та номер навчальної групи в якій він числичься;

– subject – це сутність, що описує предмет. Містить номер предмету, назву та номер викладача;

– subject_study_group – це сутність, що описує групи які навчаються на предметі. Містить номер предмету та номер навчальної групи;

– schedule – це сутність, що описує розклад предметів навчальної групи. Містить номер розкладу, номер навчальної групи, номер предмету, тип пари, день та номер пари;

– task – це сутність, що становить собою завдання. Містить номер завдання, номер предмету, тип завдання, заголовок, опис та дату створення;

– graded_task – це сутність, що описує оцінюваєме завдання. Містить номер завдання, максимальну кількість балів та дату, до якого воно потрібне бути виконане;

– score – це сутність, що становить собою оцінку. Містить номер завдання, номер студента, номер вчителя, який виставив оцінку, значення та дату оцінки;

– file – це сутність, що описує завантажений файл. Містить номер файлу, посилання на цей файл, назва файлу, тип файлу та дату завантаження;

– task_file – це сутність, що становить собою файл, що прикріплено викладачем до завдання. Містить номер файлу та номер завдання;

– student_task_file – це сутність, що становить собою файл, що прикріплен студентом до завдання. Містить номер завдання, номер студента та номер файлу;

– chat – це сутність, що становить собою повідомлення у чаті завдання. Містить номер повідомлення, номер завдання, номер студента, номер відправника, дату відправлення та зміст повідомлення.

Детальніше сутності та їх властивості з описом обмежень, що потрібні для розв’язання поставлених задач, наведено в таблиці у додатку Б.

Між сутностями у базі даних наявні три типи зв'язку - «один-до-одного», «один-до-багатьох» та «багато-до-багатьох». Розглянемо кожен зв'язок.

Один-до-одного:

– між користувачем та особистими даними користувача. Кожен користувач має особисті дані, та особисті дані можуть належить тільки одному користувачу. Для формалізації зв’язку у таблиці personal_data є зовнішній ключ, який посилається на первинний ключ member;

– між особистими даними користувача та студентом. Кожен студент має особисті дані, та особисті дані можуть належить тільки одному студенту. Для формалізації зв'язку у таблиці student первинний ключ є зовшнішнім ключем, що посилається на первинний ключ personal_data;

– між особистими даними користувача та викладачем. Кожен викладач має особисті дані, та особисті дані можуть належить тільки одному викладачу. Для формалізації зв'язку у таблиці teacher первинний ключ є зовшнішнім ключем, що посилається на первинний ключ personal_data;

– між завданням та описом оцінюваного завдання. Кожне завдання може мати опис оцінюваного завдання, але опис оцінюваного завдання може належить тільки одному завданню. Для формалізації зв’язку у таблиці graded_task первинний ключ є зовшнішнім ключем, що посилається на первинний ключ task.

Один-до-багатьох:

– між факультетом та спеціальністю. Кожен факультет може бути у багатьох спеціальностях, але кожна спеціальність може належати тільки до одного факультету. Для формалізації зв’язку у таблиці major є зовнішній ключ, який посилається на первинний ключ faculty;

– між спеціальністю та навчальною групою. Кожна спеціальність може бути у багатьох навчальних груп, але кожна навчальна група може навчатись тільки на однієї спеціальності. Для формалізації зв'язку у таблиці study_group є зовнішній ключ, який посилається на первинний ключ major;

– між навчальною групою та студентом. Кожна навчальна група може бути у багатьох студентів, але кожен студент може вчитись тільки в однієї навчальної групи. Для формалізації зв'язку у таблиці student є зовнішній ключ, який посилається на первинний ключ study_group;

– між викладачем та предметом. Кожен викладач може вести багато предметів, але кожен предмет може вести тільки один викладач. Для формалізації зв'язку у таблиці subject є зовнішній ключ, який посилається на первинний ключ teacher;

– між розкладом та навчальною групою. Кожен розклад може бути у багатьох навчальних груп, але кожна навчальна група може мати тільки один

розклад. Для формалізації зв'язку у таблиці schedule є зовнішній ключ, який посилається на первинний ключ study_group;

– між розкладом та предметом. Кожен розклад може бути у багатьох предметів, але кожен предмет може бути тільки в одному розкладі. Для формалізації зв'язку у таблиці schedule є зовнішній ключ, який посилається на первинний ключ subject;

– між завданням та предметом. Кожен предмет може мати багато завдань, але кожне завдання може належати тільки до одного предмету. Для формалізації зв'язку у таблиці task є зовнішній ключ, який посилається на первинний ключ subject;

– між оцінкою та викладачем. Кожен викладач може мати багато оцінок, але кожна оцінка може бути надана тільки одним викладачем. Для формалізації зв'язку у таблиці score є зовнішній ключ, який посилається на первинний ключ teacher;

– між завданням та файлом. К кожному завданню може бути прикріплено декілька або жодного файлу, але кожен файл може бути прикріплено тільки до одного завдання. Для формалізації зв'язку є таблиця task_file, що має два зовнішніх ключа, що посилаються на первинні ключі таблиць task та file;

– між файлом студента та завданням. К кожному завданню студент може прикріпити декілька або жодного файлу, але кожен файл може бути прикріплено тільки до одного завдання. Для формалізації зв'язку є таблиця student_task_file, що має три зовнішніх ключа, що посилаються на первинні ключі таблиць student, task та file;

– між чатом та завданням. К кожному завданню користувач може написати повідомлення, але кожне повідомлення може відноситись тільки до одного завдання. Для формалізації цього зв'язку у таблиці chat є зовнішній ключ, який посилається на первинний ключ task;

– між чатом та студентом. Кожному студенту можливо написати повідомлення, але кожне повідомлення може відноситись тільки до одного студента. Для формалізації цього зв'язку у таблиці chat є зовнішній ключ, який посилається на первинний ключ student;

– між чатом та персональними даними користувача. Кожен користувач може надіслати повідомлення, але кожне повідомлення може мати дані тільки одного відправника. Для формалізації цього зв'язку у таблиці chat є зовнішній ключ, який посилається на первинний ключ personal_data.

Багато-до-багатьох:

– між начальною групою та предметом. Кожна навчальна група може мати декілька предметів, та один і той же предмет може бути у різних навчальних груп. Для формалізації зв'язку є таблиця subject_study_group, що має два зовнішніх ключа, що посилаються на первинні ключі таблиць study_group та subject.

ER-діаграма отриманої в результаті формалізації БД наведена на рис 3.1. Для створення ER-діаграми використовувалося ПЗ PgAdmin.

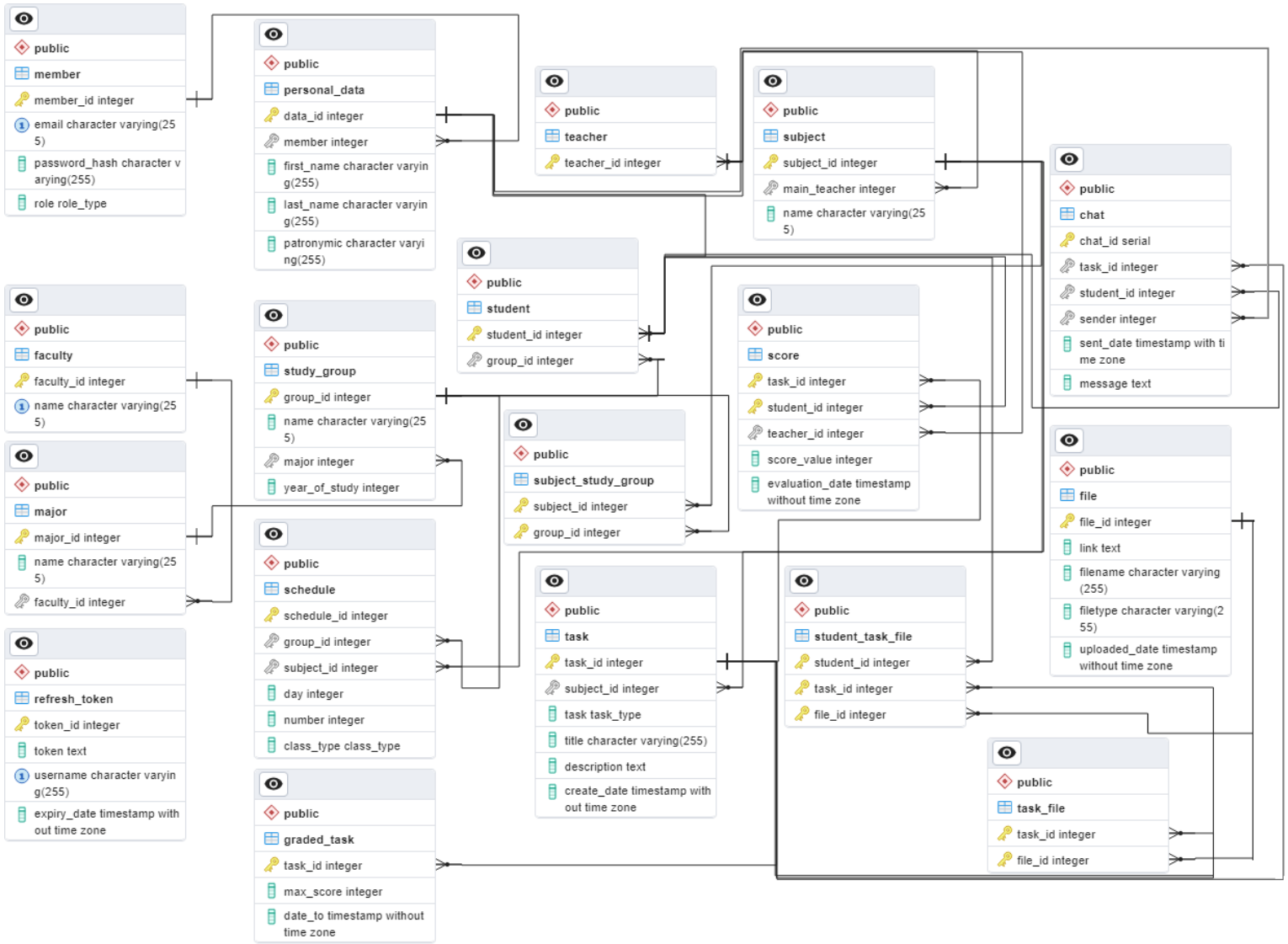

Рисунок 3.1 – ER діаграма бази даних ІС освітнього порталу

4 ВИБІР ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

Рівень управління ресурсами ІС учбового порталу розроблено за допомогою СУБД PostgreSQL за такими причинами:

– Вихідний код PostgreSQL доступний безкоштовно за ліцензією з відкритим кодом. Завдяки цього існує можливість використовувати, змінювати та впроваджувати його без обмежень;

– PostgreSQL має активну спільноту розробників та користувачів. Це забезпечує доступ до великої документації, форумів, ресурсів та інструментів;

– можливість розширити функціонал стандартних типів власними типами або доменами;

– PostgreSQL повністю відповідає стандарту SQL та підтримує широкий набір функцій, включаючи складні запити, транзакції, під запити, подання та багато іншого. Це робить його потужним інструментом для роботи з даними та забезпечує сумісність з іншими СУБД.

Для зберігання файлів використовувався Amazon S3 за наступними причинами:

– AWS S3 пропонує високорентабельне рішення для зберігання даних порівняно з традиційними базами даних. Він працює за моделлю оплати за використання, де оплата лише за фактично використаний обсяг пам’яті. Це усуває потребу в початкових інвестиціях в інфраструктуру та забезпечує гнучкість масштабування сховища в міру зростання потреб у даних;

– На відміну від традиційних баз даних із жорсткими схемами, AWS S3 дозволяє зберігати будь-які типи даних, будь то структуровані, напівструктуровані чи неструктуровані [4].

Рівень прикладного компонента розроблено за допомогою мови програмування Java з використанням фреймворку Spring Boot та наступних його модулів:

– Spring Boot Security – надає потужні функції автентифікації та гнучкі механізми авторизації, що дозволяють визначити права доступу користувачів до різних ресурсів програми;

– Spring Data JPA – це потужна структура в екосистемі Spring, яка спрощує впровадження рівнів доступу до даних на основі JPA (Java Persistence API) у програмі Java. JPA – це специфікація Java для доступу, збереження та керування даними між об’єктами Java та реляційними базами даних. Цей модуль поєднує в собі такі якості, як висока продуктивність та простота використання.

– Spring Boot Starter Web – використовується для створення RESTful програм за допомогою Spring MVC. Він використовує Tomcat як стандартний вбудований контейнер. У совокупності має всі необхідні залежності, пов'язані з веб-розробкою, що зменшує кількість залежностей збірки. [5]

Вибір Spring Boot як головного фреймворку обумовлений його простотою, великою документною базою, можливістю підключення різних готових модулів, що дозволяє збільшити швидкість розробки та її надійність.

Для розробки клієнтської частини було застосовано JavaScript-бібліотеку React, що спрощує створення інтерактивних інтерфейсів. Його перевагою є, що зробивши опис, як різні частини інтерфейсу виглядають у кожному стані додатку, React сам оновить та відрендерить лише потрібні компоненти при зміні даних. [6]

Для обміну даними між клієнтською та серверною частиною застосунку використовувався клієнт HTTP Axios. Його перевагами являються автоматичне перетворення даних запиту та відповіді у JSON формат, підтримка Promise API та можливість перехоплень запитів та відповідей. [7]

Для розробки використовувались IDE Intelij IDEA, Intelij WebStorm та Intelij DataGrip.

5 МОДЕЛЬ ПРОГРАМНОГО ЗАСТОСУНКУ

Користувачі ІС учбового порталу користуються одним веб додатком, проте існує механізм обмеження доступу до створюваного інтерфейсу для різних користувачів: в залежності від ролі користувача, який отримується під час авторизації, змінюється доступ до сторінок додатка.

Список ролей та веб сторінки, на які користувач зможе перейти перелічені нижче:

- студент має доступ до 6 сторінок, а саме доступ до сторінки перегляду розкладу, сторінки списку предметів за його навчальною групою, сторінки предмета, де він може переглянути завдання та повідомлення за цім предметом, сторінки завдання, де студент може переглянути опис завдання та завантажити зроблену роботу та сторінки зі списком усіх завдань за всіма предметами. Також студент має доступ до сторінці зміну пароля;
- викладач має доступ до 10 сторінок, а саме сторінка зі списком курсів, що веде викладач, сторінка опису курса з завданнями та повідомленнями, сторінка створення завдання, сторінка перегляду завдання зі списком студентів, що його виконали, сторінка оцінювання роботи студента, сторінка зі списком студентів за цім предметом, сторінка успішності групи, сторінка оновлень за цим курсом та сторінка усіх оновлень. Також викладач має доступ до сторінці зміну пароля;
- адміністратор має доступ до 24 сторінок, а саме сторінок керування, де вони можуть змінити, створити да побачити список факультетів, спеціальностей, навчальних груп, користувачів, курсів та розкладів. Також адміністратор має доступ до сторінки перегляду курсів, сторінки курсу зі списком завдань, сторінки зі списком студентів, сторінки з успішності групи та сторінки зміни пароля.

Повна ієрархія сторінок ІС інтернет-провайдера приведена на рис. 5.1.

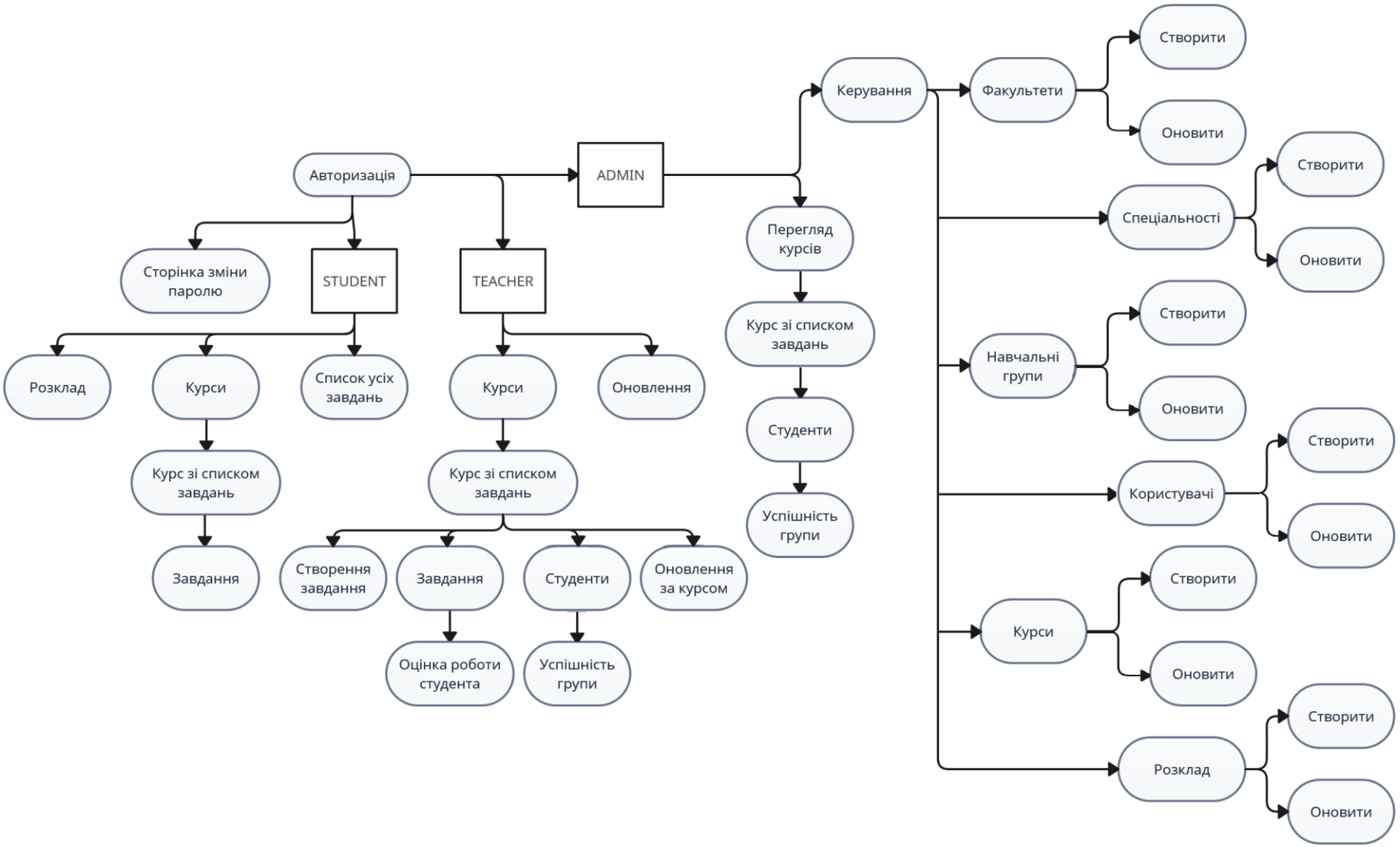

Рисунок 5.1 – Повна ієрархія сторінок ІС учбового порталу

Сутності рівня прикладного компонента майже всі відповідають за кількістю полів своїм аналогам з БД. Приклад сутності наведено на рис 5.2. Інші моделі побудовані аналогічно.

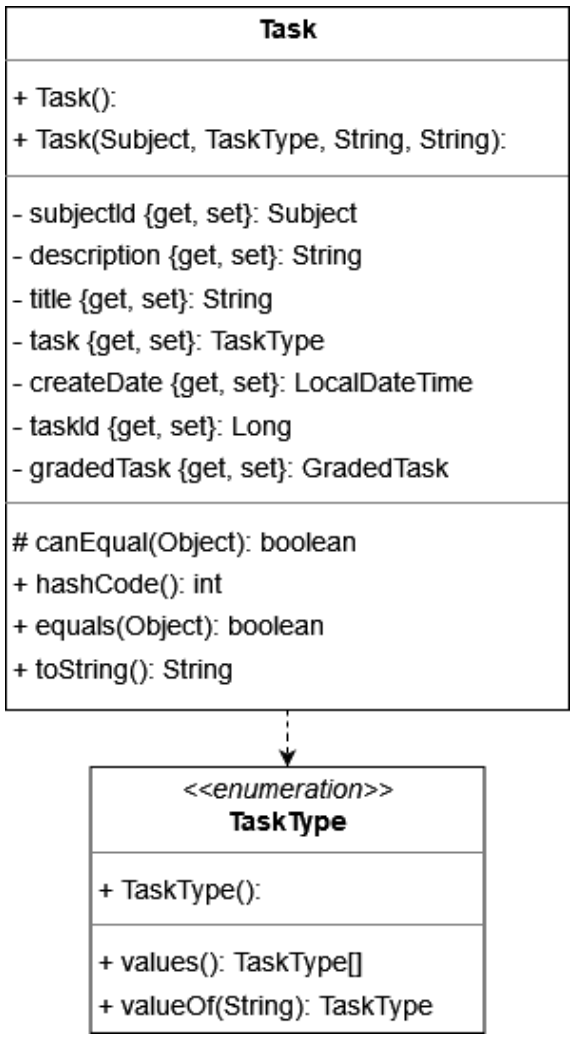

Рисунок 5.2 – UML діаграма об'єкту Task

Типічна сутність рівня прикладного компонента має аналогічну кількість полів, як і в БД, але з іншими типами даних. Наприклад, при типу зв’яку один-до-багатьох на відміну від сутності в БД, де вказується ідентифікатор на якого посилається дана сутність, на рівні прикладного компонента вказується сама сутність. Розглянемо запит на створення таблиці в БД сутності Task.

```
CREATE TABLE task(
    task_id     SERIAL PRIMARY KEY,
    subject_id  INTEGER      NOT NULL REFERENCES subject
(subject_id) ON DELETE CASCADE,
    task        task_type    NOT NULL,
    title       VARCHAR(255) NOT NULL,
    description TEXT         NOT NULL,
    create_date TIMESTAMP    NOT NULL DEFAULT now()::timestamp
);
```

Наприкладі зв'язку один-до-багатьох між завданням та предметом, можемо побачити, що у таблиці task є зовнішній ключ, який посилається на первинний ключ subject. Розглянемо код даної сутності на рівні прикладного компонента (лістинг 5.1).

```
@Data
@Entity
@NoArgsConstructor
public class Task {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long taskId;
    @ManyToOne
    @JoinColumn(name = "subject_id")
    private Subject subjectId;
    @Enumerated(EnumType.STRING)
    @Column(columnDefinition = "task_type")
    private TaskType task;
    private String title;
    private String description;
    @Generated
    private LocalDateTime createDate;
    @OneToOne(mappedBy = "task", optional = true)
    @JoinColumn(name = "graded_task")
    private GradedTask gradedTask;
    public enum TaskType {INFO, LAB, MODULAR}
    public Task(Subject subjectId, TaskType task, String title,
String description) {
        this.subjectId = subjectId;
        this.task = task;
        this.title = title;
        this.description = description;}
}
```

Лістинг 5.1 – Клас сутності Task

Можемо побачити, що реалізації зв’язку один-до-багатьох між завданням та предметом поле subjectId має тип Subject та використовуються анотації @ManyToOne та @JoinColumn. Анотація @ManyToOne вказує JPA, що між сутністю Task та Subject існує зв'язок, а анотація @JoinColumn визначає стовпець таблиці для приєднання до асоціації сутності. Таким самим чином реалізуються і інші зв’язки, такі як один-до-одного, багато-до-багатьох.

Також до класу Task додано анотацію @Entity, що вказує JPA, що цей клас є сутністю. Кожна сутність повинна мати первинний ключ, тому анатацією @Id ми вказуємо, що поле taskId являється первинним ключем. Завдяки анотії @Data автоматично створюється гетери та сетери для усіх полей класа, а @NoArgsConstuctor створює конструктор без аргументів.

На прикладі сутності Score розглянемо як реалізується складений первинний ключ. Для цього було створено клас ScoreId, код якого наведений у лістингу 5.2.

```
@Data
@Embeddable
@AllArgsConstructor
@NoArgsConstructor
public class ScoreId implements Serializable {
    private Long taskId;
    private Long studentId;
}
```

Лістинг 5.2 – Клас складеного первинного ключа сутності Score

Клас ScoreId має два поля, які є складеним первинним ключем таблиці score у БД. Анотація @Embeddable визначає клас, екземпляри якого зберігаються як невід’ємна частина сутності-власника та мають спільну ідентифікацію сутності. Далі даний клас можливо використовувати як первинний ключ, додавши анотацію @EmbeddedId.

Контролери відповідають за передачу та отримання даних використовуючи сервіси, що обробляють, передають або отримають дані з БД. Взаємодія з БД відбувається через інтерфейси репозиторію. В них описуються

методи, що здійснюють основні операції з сутностями, наприклад CRUD (Create, Read, Update, Delete). Схема взаємодії контролера, сервіса та репозиторіїв проілюстровано на рис. 5.3.

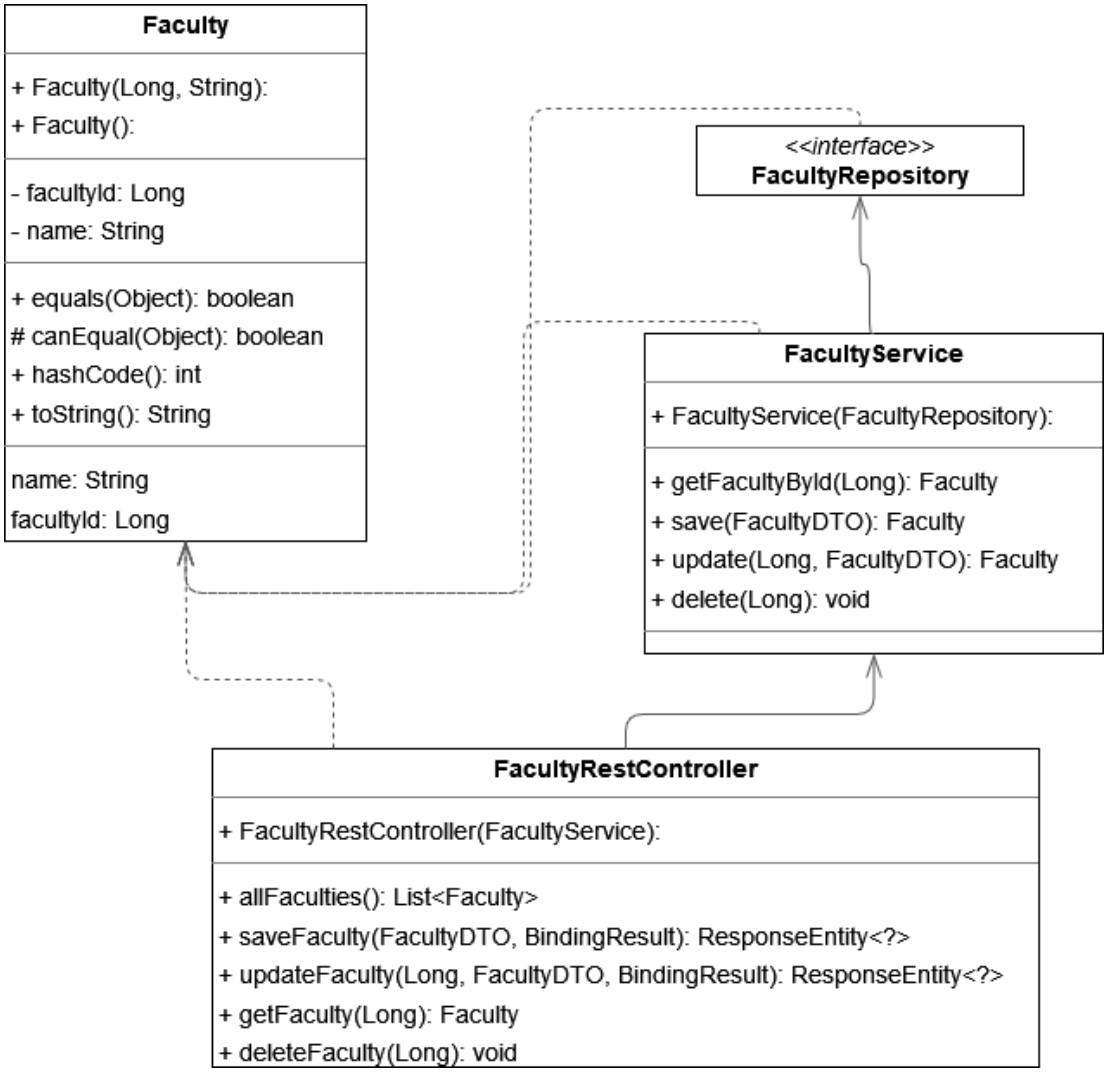

Рисунок 5.3 – UML діаграма контролеру FacultyRestController, сервісу FacultyService, сутності Faculty та репозиторія FacultyRepository

Діаграма показує ланцюжок роботи. Для прикладу розберемо випадок, коли клієнт подав запит для отримання інформації щодо факультету. Клієнт надсилає запит на відповідний шлях, за цим шляхом контролер оброблює запрос, та визиває необхідний метод у сервісі, сервіс намагається отримати факультет з цим ідентифікатор з бази даних використовуючи репозиторій.

Далі сервіс перевіряє отримані дані, якщо дані не відсутні або сталася помилка під час їх отримання він генерує помилку та логує її. Наступним кроком йде повернення даних до контролера та відправка їх до клієнта.

Така архітектура має свої переваги, кожен компонент виконує тільки ті обов'язки, які він і має виконувати, що спрощує розробку компонентів та робить їх легко замінними.

Треба зазначити, що інколи необхідно повертати чи отримувати не саму сутність, а використовувати Data Transfer Object (DTO). Це об'єкт, який використовується для передачі даних між різними рівнями системи, наприклад між клієнтом і сервером. DTO містить в собі лише дані та методи до доступу них, що спрощує їх передачу. Приклад DTO наведено у лістингу 5.3.

```
@Data
@AllArgsConstructor
public class RenewInfoDTO {
    private Long taskId;
    private String taskTitle;
    private Long subjectId;
    private String subjectName;
    private Long studentId;
    private String studentFirstname;
    private String studentSurname;
    private String studentPatronymic;
    private Integer maxScore;
    private ScoreDTO scoreDTO;
    private LocalDateTime lastUpdated;
    private TypeRenew type;

    public RenewInfoDTO(Long taskId, String taskTitle, Long
subjectId, String subjectName, Long studentId, Integer maxScore,
PersonalData personalData, LocalDateTime lastUpdated, TypeRenew
type) {
        this.taskId = taskId;
        this.taskTitle = taskTitle;
        this.subjectId = subjectId;
        this.subjectName = subjectName;
        this.studentId = studentId;
        this.maxScore = maxScore;
        this.studentFirstname = personalData.getFirstName();
```

Лістинг 5.3 – Клас RenewInfoDTO

```
        this.studentSurname = personalData.getLastName();
        this.studentPatronymic = personalData.getPatronymic();
        this.lastUpdated = lastUpdated;
        this.type = type;
    }

    public enum TypeRenew {
        TASK, MESSAGE
    }
}
```

Лістинг 5.3, лист 2

Цей клас є комбінацією одразу декількох сутностей, а саме завдання, оцінки, персональних даних та файлів студента. Він описує оновлення викладача, а саме інформацією який студент, за яким курсом і яким завданням закріпив файл чи надіслав повідомлення. Об'єднання цієї інформації в один клас спрощує обмін даними між клієнтом та сервером. Також DTO використовується при створення нового об'єкта, приклад наведено у лістингу 5.4.

```
@Data
public class MajorCreateDTO {
    @NotNull(message = "Потрібен код спеціальності")
    @Min(value = 1, message = "Код спеціальності потрібен бути
більше 0")
    @Max(value = 1000, message = "Код спеціальності потрібен
бути менше 1000")
    private Long majorId;
    @NotEmpty(message = "Потрібна назва спеціальності")
    private String name;
    @NotNull(message = "Потрібен код факультета")
    @Min(value = 1, message = "Код факультета потрібен бути
більше 0")
    private Long facultyId;
}
```

Лістинг 5.4 – Клас MajorCreateDTO

Створивши цей клас, спрощується етап створення об'єктів Major. Цей клас не тільки вказує, які самі дані необхідні для створення нової спеціальності, а ще прописавши різні анотації для всіх полів, на етапі

отримання даного об'єкта в контролері відбуватиметься перевірка на валідацію. Якщо одне із полей його не пройде, клієнту повернеться повідомлення з помилкою, яка вказана в анотації.

Окремо необхідно виділити роботу з файлами, діаграма контролеру, сервісів, сутностей та репозиторіїв наведено на рисунку 5.4.

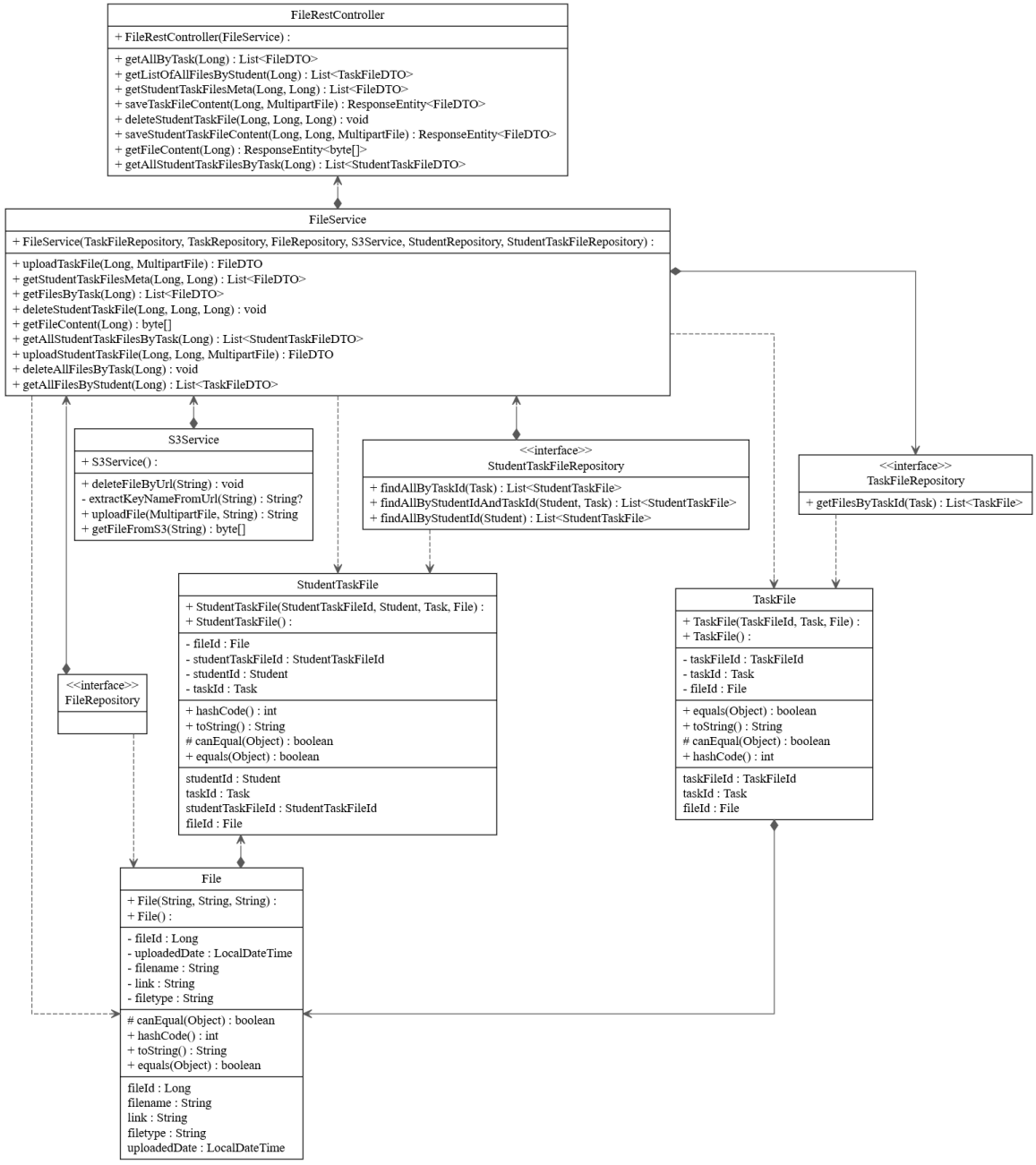

Рисунок 5.4 – UML діаграма контролера, сервісів, сутностей та репозиторіїв файлів

Коли клієнт надсилає запит на збереження файлу, наприклад файл, що викладач прикріпив до завдання, спочатку контролер FileRestController перевіряє розмір файлу і якщо він не перевищує лімит, викликається необхідний метод у FileService. В ньому отримується мета дані файлу, а саме назва та тип. Після цього використавши необхідний метод сервісу S3Service виконується спроба надіслати даний файл до вказаного заздалегідь у налаштуваннях бакету. Після вдалого завантаження, повертається унікальне посилання на файл у бакеті. Мета дані файлу та посилання на файл завантажуються у базу даних, використовуючи FileRepository, після цього отриманий у ході завантаження файлу ідентифікатор і ідентифікатор завдання зберігаються у сутності TaskFile, що завантажується до бази даних через TaskFileRepository.

Таким чином, коли студенту необхідно бути завантажити цей файл, у базі даних вже є посилання на збережений файл у бакеті, звідки цей файл можливо бути завантажити та повернути клієнту.

Діаграми класів моделі та репозиторіїв проілюстровані у додатку В.

6 СТВОРЕННЯ БАЗИ ДАНИХ

Розглянемо запит на створення таблиці file.

```
CREATE TABLE file
(
    file_id       SERIAL PRIMARY KEY,
    link          TEXT         NOT NULL,
    filename      VARCHAR(255) NOT NULL,
    filetype      VARCHAR(255) NOT NULL,
    uploaded_date TIMESTAMP    NOT NULL DEFAULT now()::timestamp
);
```

Для створення таблиці використовується команда CREATE TABLE. Для кожного поля вказується тип даних. Поле file_id є первинним ключем та має тип SERIAL, в основі цього типу лежить тип INTEGER, проте значенням за умовчанням для величин цього є не NULL, а наступне ціле число. Таким чином, якщо додавати записи до таблиці, не вказуючи для полів типу SERIAL, ці значення будуть надаватися автоматично як порядкові цілі числа. Обмеження NOT NULL означає, що поле не може бути порожнім.

Розглянемо запит на створення таблиці task_file, яка посилається на таблицю task та file.

```
CREATE TABLE task_file
(    task_id INTEGER REFERENCES task (task_id) ON DELETE
CASCADE,
    file_id INTEGER REFERENCES file (file_id) ON DELETE CASCADE,
    PRIMARY KEY (task_id, file_id)
);
```

Поле task_id посилається на поле task_id таблиці task та є зовнішнім ключем. Зовнішній ключ — це поле (або набір полів) в одній таблиці, яке посилається на первинний ключ в іншій таблиці. Тексти команд SQL-запитів, які використовуються для створення таблиць бази даних, наведені в додатку Г.

7 ПРОГРАМНА РЕАЛІЗАЦІЯ ІНТЕРФЕЙСУ

Розглянемо клас типового контролера MajorRestController, який відповідає за повернення списку спеціальностей, повернення спеціальності за кодом, збереженння нової спеціальності, оновлення та видалення спеціальності. Код наведений у лістингу 7.1.

```
@RestController
@RequestMapping(path = "/v1")
public class MajorRestController {
    private final MajorService majorService;

    public MajorRestController(MajorService majorService) {
        this.majorService = majorService;
    }

    @GetMapping("/majors")
    public List<MajorDTO> getAllMajors() {
        return majorService.getAllMajors();
    }

    @GetMapping("/majors/{majorId}")
    public MajorDTO getMajorById(@PathVariable Long majorId) {
        return majorService.getMajorById(majorId);
    }

    @PostMapping("/majors")
    public ResponseEntity<?> saveMajor(@Valid @RequestBody
MajorCreateDTO majorCreateDTO,
                                       BindingResult bindingResult) {
        if (bindingResult.hasErrors()) {
            Map<String, String> errors =
bindingResult.getFieldErrors().stream()
                    .collect(Collectors.toMap(
                            FieldError::getField,
                            FieldError::getDefaultMessage));
            return new ResponseEntity<>(errors,
HttpStatus.BAD_REQUEST);
        }
        MajorDTO majorDTO = majorService.save(majorCreateDTO);
```

Лістинг 7.1 – Клас MajorRestController

```
        return majorDTO != null ? new ResponseEntity<>(majorDTO,
HttpStatus.CREATED) : new
ResponseEntity<>(HttpStatus.BAD_REQUEST);
    }

    @PutMapping("/majors/{majorId}")
    public ResponseEntity<?> updateMajor(@PathVariable Long
majorId, @Valid @RequestBody MajorCreateDTO majorCreateDTO,
                                         BindingResult
bindingResult) {
        if (bindingResult.hasErrors()) {
            Map<String, String> errors =
bindingResult.getFieldErrors().stream()
                    .collect(Collectors.toMap(
                            FieldError::getField,
                            FieldError::getDefaultMessage));
            return new ResponseEntity<>(errors,
HttpStatus.BAD_REQUEST);
        }
        MajorDTO majorDTO = majorService.update(majorId,
majorCreateDTO);

        return majorDTO != null ? ResponseEntity.ok(majorDTO) :
new ResponseEntity<>(HttpStatus.BAD_REQUEST);
    }

    @DeleteMapping("/majors/{majorId}")
    public void delete(@PathVariable Long majorId) {
        majorService.delete(majorId);
    }
}
```

Лістинг 7.1, лист 2

Розглянемо даний контролер.

У конструктор даного класу передається об’єкт сервісу MajorService, що використовується для отримання, обробки та збереження даних у БД.

Клієнт взаємодіє з rest контролером через відповідний шлях та HTTP метод. Наприклад, для того щоб клієнт отримав список усіх спеціальностей йому необхідно зробити GET запит за маршрутом “…/v1/majors”, після цього цей запит обробе метод getAllMajors(). Для отримання спеціальності за її кодом, клієнту необхідно вказати його у маршруті, наприклад зробити GET

запит за маршрутом “…/v1/majors/3”, після цього цей запит обробе метод getMajorById(), в який завдяки анотації @PathVariable автоматично передасться параметр majorId що і є кодом спеціальності. Якщо спеціальність за цим кодом буде знайдена, вона буде повернена клієнту у тілі відповіді, інакше поверне помилку.

Для додавання нової спеціальності клієнту необхідно зробити POST запит за маршрутом “…/v1/majors” та передати у тілі запиту опис нової спеціальності, після цього цей запит обробе метод saveMajor(), в який завдяки анотації @RequestBody тіло запиту перетворюється у Java об'єкт вказаного типу автоматично передасться як параметр majorCreateDTO. Завдяки анотації @Valid, переданий об'єкт перевіряється на валідність. У разі помилок, застосовуючи параметр bindingResult, повертається помилка з відповідним повідомленням. Якщо об'єкт вдало пройшов валідацію завдяки сервісу majorService спробується його зберегти у базі даних. При вдалому збереженні у тілі відповіді у json форматі повертається збережена спеціальність та статус відповіді визнається як CREATED, що позначає успішне завантаження даних, а якщо при спробі зберегти новий об'єкт виникла помилка, наприклад спеціальність з таким кодом вже існує, повертається порожнє тіло зі статусом відповіді BAD_REQUEST.

Для оновлення спеціальності клієнту необхідно зробити PUT запит за маршрутом “…/v1/majors/код_спеціальності” та передати у тілі запиту опис оновленої спеціальності, після цього цей запит обробне метод updateMajor(). Як і при створенні нової спеціальності, спочатку проводиться валідація переданого об'єкту. Якщо об'єкт вдало пройшов валідацію викликається відповідний метод сервісу majorService, який намагається оновити вже існуючу спеціальність за кодом, який передається як аргумент. При вдалому оновлені у тілі відповіді у json форматі повертається збережена спеціальність та статус відповіді визнається як OK, що позначає успішне завершення запиту, а якщо при спробі оновити об'єкт виникла помилка, наприклад спеціальності

з таким кодом не існує, повертається порожнє тіло зі статусом відповіді BAD_REQUEST.

Для можливості обміну даними між клієнтською та серверною частиною застосунку використовувався клієнт HTTP Axios. Розглянемо типовий запит до сервера з клієнтського застосунку. Код наведений у лістингу 7.2.

```
export const retrieveMajorById =
    (majorId) => apiClient.get(`/majors/${majorId}`)
```

Лістинг 7.2 – Запит для отримання спеціальності за кодом

Визвавши дану функцію та передавши код спеціальності, створюється GET запит за маршрутом “…/majors/код_спеціальності” та повертається відповідь запита у json форматі. Треба зазначити, що apiClient є налаштований об'єкт axios. Код налаштування axios наведений у лістингу 7.3.

```
export const apiClient = axios.create(
    {
        baseURL: 'http://localhost:8081/v1'
    }
)
```

Лістинг 7.3 – Створення об'єкта axios з вказаним базовим маршрутом

Подальший обмін даними між клієнтом та сервером здійснюється тільки через об’єкт apiClient. Усі запити зберігаються у файлі MemberApi.js. Розглянемо типовий приклад використання функції з MemberApi. Код наведений у лістингу 7.4.

```
function refreshMajors() {
    setIsLoading(true)
    Promise.all([
        retrieveMajors()
    ])
        .then((responses) => setMajors(responses[0].data))
        .finally(() => setIsLoading(false))
}
```

Лістинг 7.4 – Отримання всіх спеціальностей

Дана функція виконується кожний раз, коли користувач оновлює сторінку зі списком спеціальностей. Розглянемо дану функцію.

Спочатку стан сторінки визначається як у процесі завантаження. Цез роблено для того, щоб компонент сторінки не почав завантаження з ще не завантаженими даними, що може повести за собою помилку при його рендері. Після цього використовується інструмент Promise в якому викликається функція для запиту всіх спеціальностей. Використання Promise.all() дає гарантію, що код у конструкції then() виконається тільки після вдалого завершення усіх “обіцянок”, що передані до Promise, тобто тільки після отримання даних, виконується їх збереження у змінні. Конструкція finally виконується після завершення then(), в цьому прикладі після завершення збереження отриманих спеціальностей у змінні. Саме тут змінюється стан сторінки на завантаження завершено, після чого починається рендер компоненту.

Розглянемо приклад збереження даних на сервері. Код наведений у лістингу 7.5.

```
const validate = values => {
    const errors = {};
    if (values.name.length === 0) {
        errors.name = 'Потрібна назва'
    }
    if (values.faculty === '') {
        errors.faculty = 'Потрібен факультет'
    }

    if (majorId == null) {
        if (!Number.isInteger(values.createdMajorId)) {
            errors.createdMajorId = 'Потрібен код спеціальності'
        } else {
            if (values.createdMajorId < 0 ||
values.createdMajorId >= 1000) {
                errors.createdMajorId = 'Потрібен валідний код
спеціальності'
            }
        }
```

Лістинг 7.5 – Збереження нової або оновленої спеціальності

```
    }
    return errors
}
const formik = useFormik({
    initialValues: {
        createdMajorId: '', name: '', faculty: ''
    }, validate, onSubmit: values => {
        if (majorId != null) {
            Promise.all([updateMajor(majorId, ({
                majorId: majorId,
                name: values.name,
                facultyId: values.faculty}))])
                .then(() => navigator('/admin/manage/majors'))
                .catch(() => setErrorMessage('Помилка'))} else {
            Promise.all([saveMajor(({
                majorId: values.createdMajorId,
                name: values.name,
                facultyId: values.faculty}))])
                .then(() => navigator('/admin/manage/majors'))
                .catch(() => setErrorMessage('Спеціальність з
таким кодом вже існує')) }}});
```

Лістинг 7.5, лист 2

Розглянемо цей код. Коли користувач заповнив форму спеціальності, виконується валідація цих даних на клієнтський частині, а саме перевіряється, чи не порожнє якесь поле та чи валідні введені дані щодо коду спеціальності. Якщо якесь поле не задовольняє вказаним умовам, у формі до цього поля прикріпляється повідомлення про помилку, з відповідно вказаним текстом. Якщо всі дані вдало пройшли валідацію виконується їх відправка на сервер. В залежності від маршруту даної сторінки, а саме чи вказано в маршруті код спеціальності, надсилаються різні запити, а саме якщо код спеціальності вказано, то викликається метод для створення запиту на оновлення спеціальності updateMajor(), в який передається код спеціальності та json об'єкт спеціальності. Якщо ж код спеціальності у шляху сторінці відсутній, то викликається метод для створення запиту на збереження нової спеціальності saveMajor(), в який передається тільки об'єкт спеціальності. Після отримання відповіді відповідного запиту, в залежності від статусу відповіді виконується різні блоки коду. Якщо код стану відповіді належить до класу успішного,

виконується перенаправлення користувача до сторінки зі списком усіх спеціальностей. У іншому випадку, користувач побачить повідомлення про помилку з відповідно вказаним текстом.

Для роботи з файлами створені наступні функції запитів, код яких наведен у лістингу 7.6.

```
export const retrieveFilesMetaByTask =
    (taskId) => apiClient.get(`/tasks/${taskId}/files`)
export const retrieveFileContent =
    (fileId) => apiClient.get(`/files/${fileId}`, {
responseType: 'blob' })
export const uploadStudentTaskFileContent =
    (taskId, studentId, formData) =>
apiClient.post(`/tasks/${taskId}/students/${studentId}/files`,
formData, {headers: {'Content-Type': 'multipart/form-data'}})
export const uploadTaskFileContent =
    (taskId, formData) =>
apiClient.post(`/tasks/${taskId}/teach/files`, formData, {
        headers: {'Content-Type': 'multipart/form-data'}})
export const retrieveStudentTaskFileMeta =
    (studentId, taskId) =>
apiClient.get(`/tasks/${taskId}/students/${studentId}/files`)
export const deleteStudentTaskFile =
(studentId, taskId, fileId) =>
apiClient.delete(`/tasks/${taskId}/students/${studentId}/files/$
{fileId}`)
export const retrieveAllFilesByStudent =
    (studentId) => apiClient.get(`/student/${studentId}/files`)
```

Лістинг 7.6 – Функції запитів для операцій з файлами

Перед рендером сторінки, де присутня можливість завантаження файлу з сервера, отримуються метадані прикріплених файлів завдяки функцій retrieveFilesMetaByTask(), retrieveStudentTaskFileMeta() або retrieveAllFilesByStudent(). Якщо користувач бажає завантажити файл з серверу, визивається функція retrieveFileContent() з аргументом ідентифікатора файлу.

Лістинг файлу MemberApi з усіма запитами наведен у додатку Д.

Лістинг прикладу компонента сторінки наведен у додатку Е.

8 БЕЗПЕКА ІНФОРМАЦІЙНОЇ СИСТЕМИ

Для захисту даних, доступ до всіх rest-точок дозволен тільки при наявності access JWT (Json Web Token) [8]. Вийнятком є точка авторизації та оновлення токена. Access токен надається клієнту тільки після успішної аторизації, разберемо цей етап більше детально. Коли клієнт надсилає POST запит за шляхом "…/login" з даними для авторизації у тілі запиту (лістинг 8.1), починається спроба авторизації.

```
@PostMapping("/login")
public JwtAuthenticationResponse signIn(@RequestBody LoginDTO
loginDTO) {
    return authenticationService.authenticate(loginDTO);
}
```

Лістинг 8.1 – Метод rest-точки для авторизації

Використовуючи метод authenticate сервісу AuthenticationService та передавши дані користувача як атрибут, сервер намагається виконати авторизацію. При успішній авторизації у тілі відповіді запита повертається об'єкт JwtAuthenticationResponse, що містить в собі access-токен та refresh-токен. Код методу authenticate наведено у лістингу 8.2.

```
public JwtAuthenticationResponse authenticate(LoginDTO loginDTO)
{
    authenticationManager.authenticate(new
UsernamePasswordAuthenticationToken(
            loginDTO.getUsername(),
            loginDTO.getPassword()));

    UserDetails user =
memberService.userDetailsService().loadUserByUsername(loginDTO.g
etUsername());

    return new
JwtAuthenticationResponse(jwtService.generateToken(user),
jwtService.getRefreshToken(user.getUsername()).getToken());
}
```

Лістинг 8.2 – Метод authenticate класу AuthenticationService

Використовуючи метод authenticate об'єкту класу AuthenticationManager, виконується спроба авторизувати користувача згідно переданим даним. Необхідно зазначити, що перед його використанням необхідно його налаштувати та вказати необхідний провайдер для авторизації. Провайдер описує, як само буде проходити перевірка даних клієнта для авторизації. Так як було обрано авторизацію на основі логіну та паролю, буде використовуватись вбудована реалізація, а саме DaoAuthenticationProvider. Також при його конфігурації необхідно передати об'єкт класу PasswordEncoder, так як задля безпеки, усі паролі у базі даних зберігаються у хешованому вигляді. Усі лістинги класів, що пов'язані з безпекою наведені у додатку Ж.

Якщо при авторизації сталась помилка, вона повертається клієнту, інакше з бази даних використовуючи MemberService отримуються облікові дані користувача з логіном, що був вказано під час авторизації. Після цього генеруються access-токен так refresh-токен завдяки сервісу jwtService. У лістингу 8.3 наведено методи генерації access-токену.

```
public String generateToken(UserDetails userDetails) {
    Map<String, Object> claims = new HashMap<>();
    if (userDetails instanceof Member customUserDetails) {
        claims.put("id", customUserDetails.getMemberId());
        claims.put("email", customUserDetails.getEmail());
        claims.put("role", customUserDetails.getRole());
    }
    return generateToken(claims, userDetails);
}

private String generateToken(Map<String, Object> extraClaims,
UserDetails userDetails) {
    return
Jwts.builder().setClaims(extraClaims).setSubject(userDetails.get
Username())
            .setIssuedAt(new Date(System.currentTimeMillis()))
            .setExpiration(new Date(System.currentTimeMillis() +
JWT_EXPIRATION))
            .signWith(getSigningKey(),
SignatureAlgorithm.HS256).compact();
}
```

Лістинг 8.3 – Методи генерації access-токену сервіса JwtService

Передавши дані користувача, спочатку створюється колекція корисного навантаження до токену, це буде ідентифікатор користувача, його логін та роль. Ці дані клієнт зможе отримати прочитавши токен. Далі створюється сам access-токен, де вказується його дата підпису, строк дії, алгоритм та ключ яким він підписан, корисне навантаження та кому він видан. Треба зазначити, що строк дії access-токена 5 хвилин, після чого він не є дійсним. Аналогічно створюється refresh-токен, тільки він не має корисного навантаження, а його строк – 6 годин. Також refresh-токен зберігається у базі даних та прикріплюється до користувачу, якому він був видан.

Розглянемо етап аутентифікації користувача на стороні клієнта. Після заповнення форми авторизації, на сервер надсилається запит з даними користувача. Після чого перевіряється статус відповіді і якщо трапилась помилка під час авторизації користувач побачить помилку. Якщо ж статус відповіді має код 200, тоб то OK, корисне навантаження, а саме ідентифікатор користувача, логін та роль зберігається у локальному сховищі. Також там зберігається access-токе та refresh-токен. Після цього створюється перехоплювач запитів, код якого наведено у лістингу 8.4.

```
apiClient.interceptors.request.use(
    (config) => {
        config.headers.Authorization = `Bearer
${response.data.accessToken}`
        return config
    }
)
```

Лістинг 8.4 – Перехоплювач запитів клієнта

Даний перехоплювач додає у голову кожного запиту клієнту збережений у локальному сховищі access-токен.

Щоб обмежити доступ до усіх інших rest-точок неавторизованим користувачам існує механізм фільтру. Код методу фільтра наведено у лістингу 8.5.

```
@Override
protected void doFilterInternal(
        @NonNull HttpServletRequest request,
        @NonNull HttpServletResponse response,
        @NonNull FilterChain filterChain
) throws ServletException, IOException {

    var authHeader = request.getHeader(HEADER_NAME);
    if (!StringUtils.hasLength(authHeader) ||
!StringUtils.startsWithIgnoreCase(authHeader, BEARER_PREFIX)) {
        filterChain.doFilter(request, response);
        return;
    }

    var jwt = authHeader.substring(BEARER_PREFIX.length());
    var username = jwtService.extractUserName(jwt);

    if (StringUtils.hasLength(username) &&
SecurityContextHolder.getContext().getAuthentication() == null)
{
        UserDetails userDetails = memberService
                .userDetailsService()
                .loadUserByUsername(username);

        if (jwtService.isTokenValid(jwt, userDetails)) {
            SecurityContext context =
SecurityContextHolder.createEmptyContext();

            UsernamePasswordAuthenticationToken authToken = new
UsernamePasswordAuthenticationToken(
                    userDetails,
                    null,
                    userDetails.getAuthorities()
            );

            authToken.setDetails(new
WebAuthenticationDetailsSource().buildDetails(request));
            context.setAuthentication(authToken);
            SecurityContextHolder.setContext(context);
        }
    }
    filterChain.doFilter(request, response);
}
```

Лістинг 8.5 – Фільтр запитів до сервера

Даний фільтр перехоплює усі запити до сервера, та перевіряє, чи є у хедері запроса access-токен та чи він валідний. Розглянемо його принцип роботи.

Спочатку в фільтрі перевіряється наявність токена. Далі з наявного токена отримується логін користувача на основі якого отримуються облікові дані цього користувача. Після цього завдяки сервісу JwtService токен перевіряється на валідність, а саме перевіряється його строк та належність до даного користувача. Потрібно зазначити, що у будь-якому випадку, фільтр пропустить даний запит далі по ланцюжку фільтрів, але дані користувача додаються у тимчасовий контекст запита тільки в тому випадку, якщо access-токен валідний. Якщо ж ні, запит не пропустить інший фільтр, який перевіряє доступ користувача до шляху rest-точки на основі ролі користувача у контексті. Ці правила прописані в конфігураторі безпеки. Для деяких точок проводиться перевірка доступу саме цього користувача для саме цих даних, наприклад перед тим як повернути викладачу рейтинговий список студентів за певним курсом, окрім перевірки ролі користувача, перевіряється чи для цього предмету викладачем є саме цій користувач.

Так як в даної системі існує механізм оновлення токену доступу на основі refresh-токену, розглянемо цей механізм. На стороні клієнта створено перехоплювач відповідей на запити до сервера, код якого наведено у лістинг 8.6.

```
apiClient.interceptors.response.use(
    response => response, async error => {
        const originalRequest = error.config;
        if (error.response.status === 403 &&
!originalRequest._retry) {originalRequest._retry = true;
            try {const refreshToken =
localStorage.getItem('refreshToken');
                const response = await
apiClient.post('/refresh', {refreshToken});
                const newAccessToken =
response.data.accessToken; localStorage.setItem('accessToken',
newAccessToken);
              apiClient.defaults.headers.common['Authorization']
= `Bearer ${newAccessToken}`;
                originalRequest.headers['Authorization'] =
`Bearer ${newAccessToken}`;
                return apiClient(originalRequest);
```

Лістинг 8.6 – Перехоплювач для оновлення токену доступу

```
            } catch (e) {
                console.error(e);
            }}
        return Promise.reject(error)
    }
)
```

Лістинг 8.6, лист 2

Даний перехоплювач перевіряє усі стани відповідей на запити до сервера, і якщо код статусу відповіді дорівнює 403, він надсилає запит до "…/refresh" з refresh-токеном у тілі запиту. Після цього отриманий новий токен доступу зберігається у локальному сховищі та оновлюється у перехоплювачі запитів до сервера. Розглянемо механізм оновлення токена на сервері. Після отримання POST запита за маршрутом "…/refresh" виконується спроба оновити access-токен у методі refreshToken сервісу AuthenticationService, код метода якого наведено у лістингу 8.7.

```
public Map<String, String> refreshToken(String refreshTokenStr){
    return jwtService.findByToken(refreshTokenStr)
            .map(jwtService::verifyExpiration)
            .map(RefreshToken::getUsername)
            .map(username -> {
                String newAccessToken =
jwtService.generateToken(memberService.userDetailsService().load
UserByUsername(username));
                Map<String, String> tokens = new HashMap<>();
                tokens.put("accessToken", newAccessToken);
                tokens.put("refreshToken",
jwtService.getRefreshToken(username).getToken());
                return tokens; }).orElseGet(() -> null); }
```

Лістинг 8.7 – Метод refreshToken сервісу AuthenticationService

Завдяки сервісу JwtService спочатку даний токен оновлення шукається в базі даних, після чого перевіряється його валідність, а саме перевіряється його дата закінчення. Далі створюється новий access-токен на основі облікових даних користувача знайдених через логін. Якщо все пройшло успішно, повертається новий access-токен та refresh-токен, інакше нічого зі статусом відповіді помилки.

9 ІНСТРУКЦІЇ КОРИСТУВАЧА

Потрапивши на сайт, кожен відвідувач спочатку повинен авторизуватись у системі (рис. 9.1).

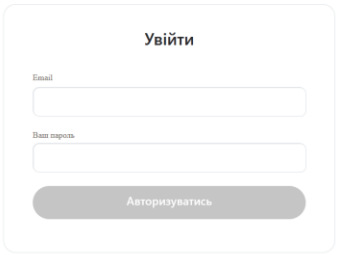

Рисунок 9.1 – Сторінка авторизації користувача

Після введення логіну, що є поштою, паролю та успішної авторизації в залежності від ролі користувача йому надається доступ до різних сторінок та функціоналу веб додатка. Спочатку розглянемо сторінки студента. Після авторизації він автоматично потрапляє на сторінку розкладу (рис. 9.2). На цій сторінці студент може переглянути розклад предметів та тип занять за навчальною групою, до якої він належить.

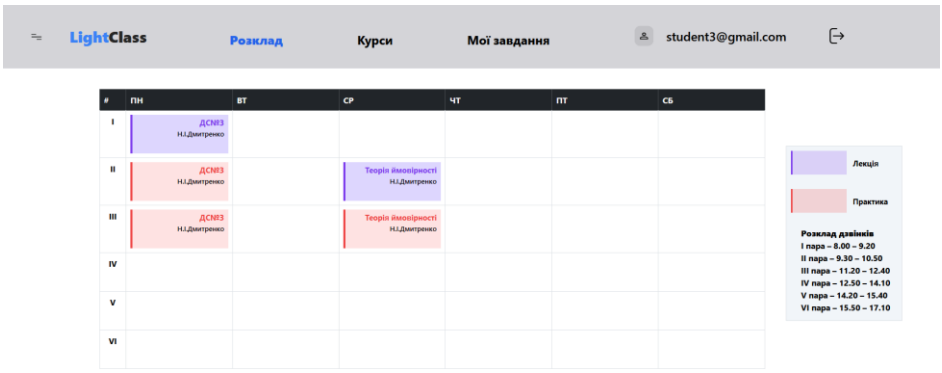

Рисунок 9.2 – Сторінка розкладу студента

Для перегляду списку предметів студент може перейти до сторінки курсів (рис. 9.3) через меню навігації на верху сторінки.

Рисунок 9.3 – Список предметів студента

На даній сторінки студент може побачити список предметів з їх назвою та прізвищем викладача. Щоб переглянути сторінку зі списком завдань та повідомлень за даним предметом (рис. 9.4), користувачу необхідно натиснути на блок цього предмета.

Рисунок 9.4 – Список завдань за предметом

На цій сторінки студент може переглянути список завдань за предметом, дату їх створення, дата закінчення та оцінки за виконані завдання, якщо такі існують. Також для зручності оцінені завдання візуально виділені зеленим кольором, а завдання які ще не оцінені – червоним. Також студент може переглянути список повідомлень за цим курсом (рис. 9.5). Повідомлення це не оцінюване завдання, яке може містити новини за предметом, файли с методичними матеріали, посилання на ресурси та інше.

При натисканні на завдання або повідомлення студент перейде до сторінки з описом завдання (рис. 9.6). Якщо це оцінюване завдання, студент може завантажити файли з виконаною роботою, після чого викладач зможе їх оцінити. Також студент може завантажити файл, при його наявності, що прикріплено викладачем до завдання. У кожному завданні та повідомленні існує чат між викладачем та студентом, де вони можуть обмінюватись питаннями щодо виконання завдання.

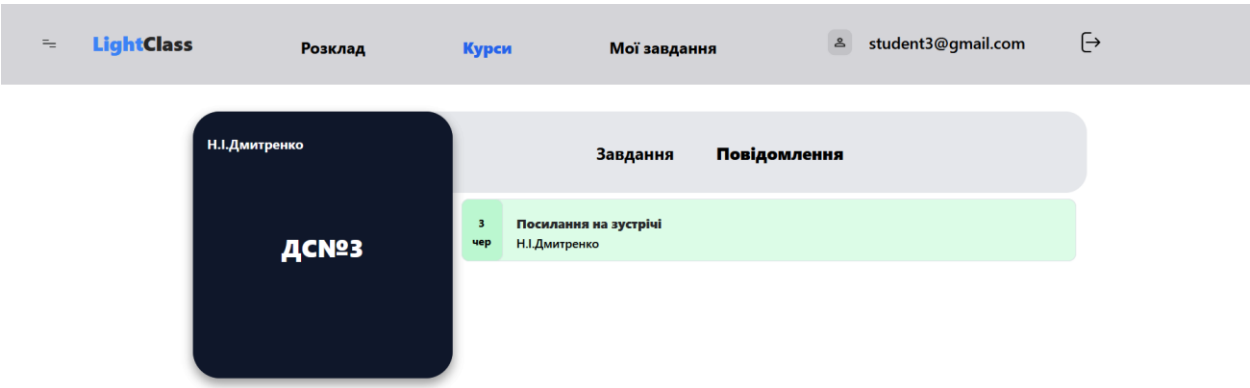

Рисунок 9.5 – Список повідомлень за предметом

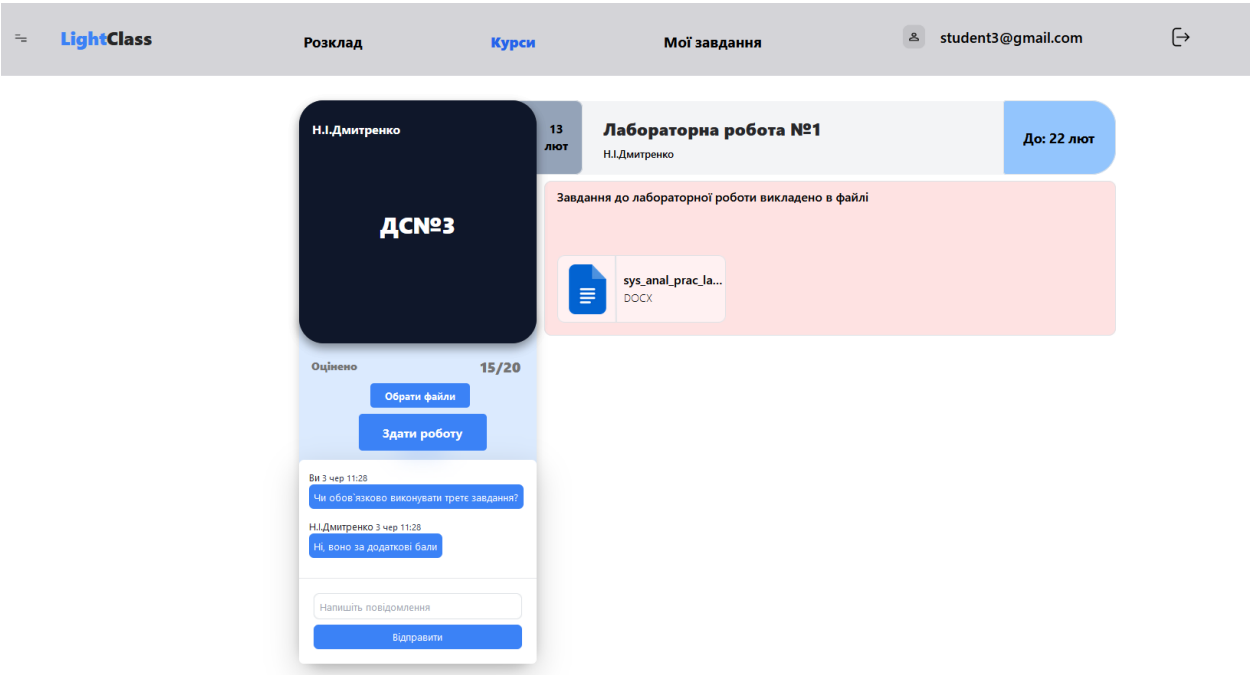

Рисунок 9.6 – Сторінка з описом завдання

Також студент може переглянути список усіх завдань (рис. 9.7) через меню навігації на верху сторінки. На цій сторінці перелічено усі оцінювані завдання с можливістю сортування за параметром. За замовченням виводиться список завдань у порядку їх створення.

Рисунок 9.7 – Список оцінюваних завдань за всіма предметами

Натиснувши на завдання, студент може перейти до сторінки з його описом.

Розглянемо сторінки для викладача. Після успішної авторизації викладач бачить сторінку з предметами у яких цей користувач є викладачем (рис. 9.8).

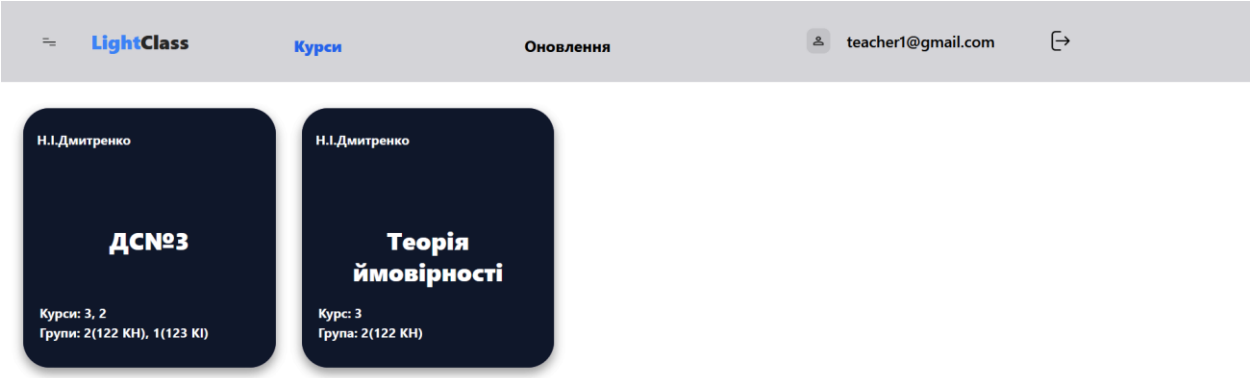

Рисунок 9.8 – Список предметів для викладача

На цій сторінки відображено назви предметів, прізвище викладача та інформації про навчальні групи, що вивчають цей предмет. Натиснувши на блок предмета відбувається перехід до сторінки предмета (рис. 9.9).

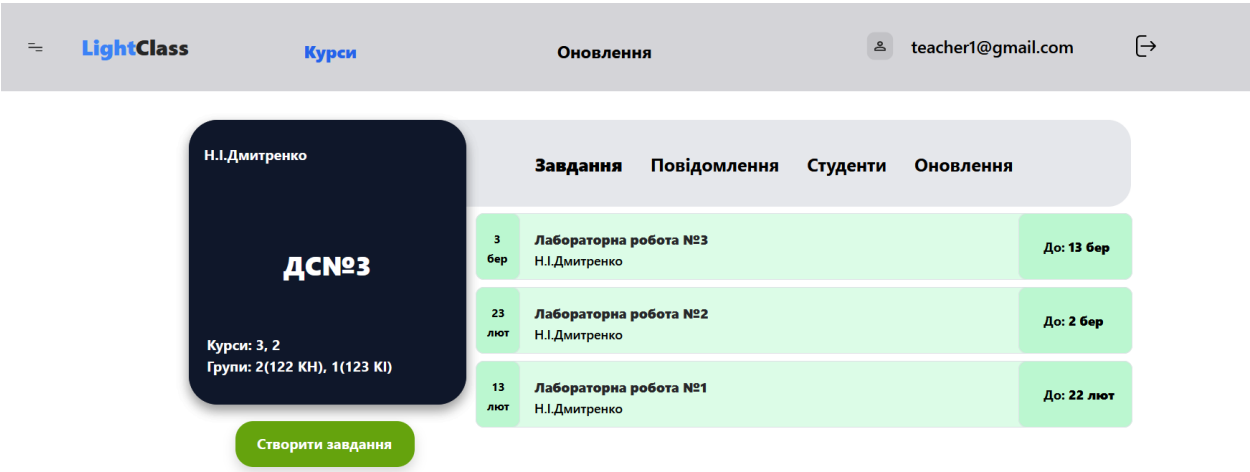

Рисунок 9.9 – Сторінка предмета та завдань до нього

На початковій сторінці предмету відображається список оцінюваємих завдань. Щоб створити завдання потрібно натиснути відповідну кнопку, після чого здійснюється перехід до сторінки з формою для створення завдання (рис. 9.10). В залежності від обраного типу завдання, необхідно заповнити відповідні поля: для повідомлення необхідно вказати заголовок та опис повідомлення та при необхідності додати файли, а для оцінюваного завдання або модульної роботи вказати максимальну кількість балів та дату закінчення. Після натиснення кнопки “Створити завдання” створене завдання додається до предмету, а користувача буде перенаправлено на початкову сторінку предмета. Для перегляду списку повідомлень для викладача (рис. 9.11) необхідно обрати вікно “Повідомлення”.

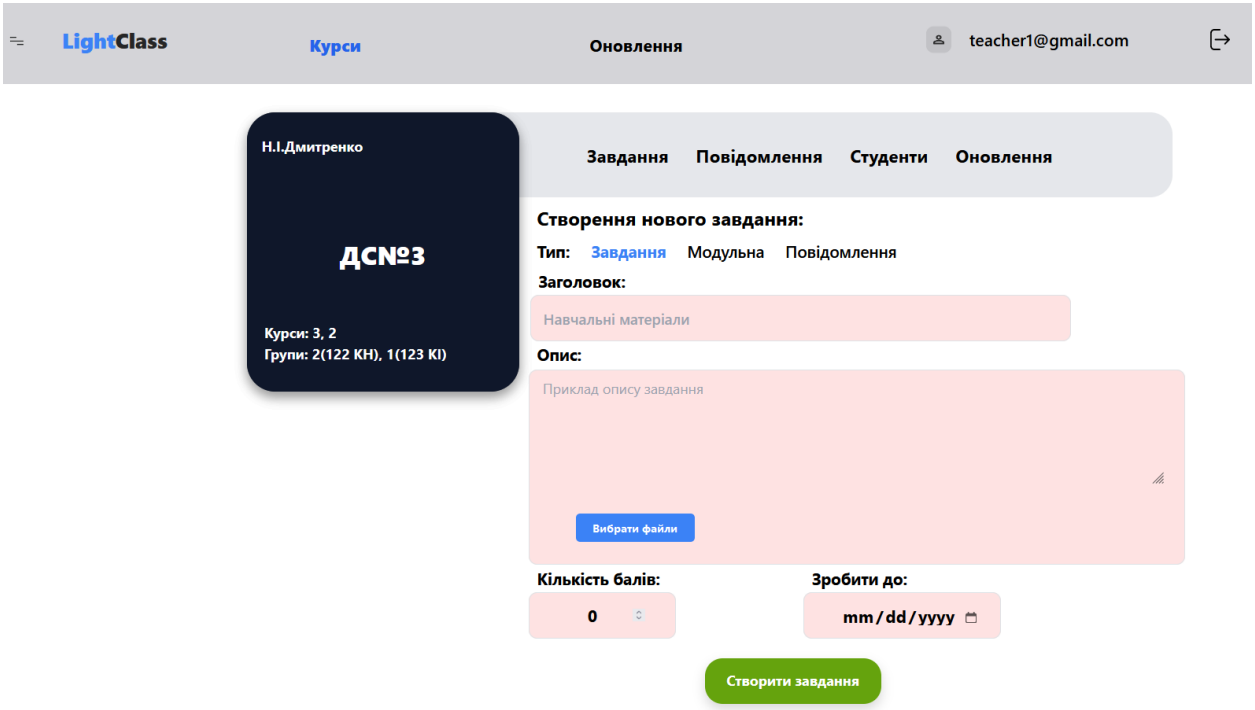

Рисунок 9.10 – Форма для створення завдання

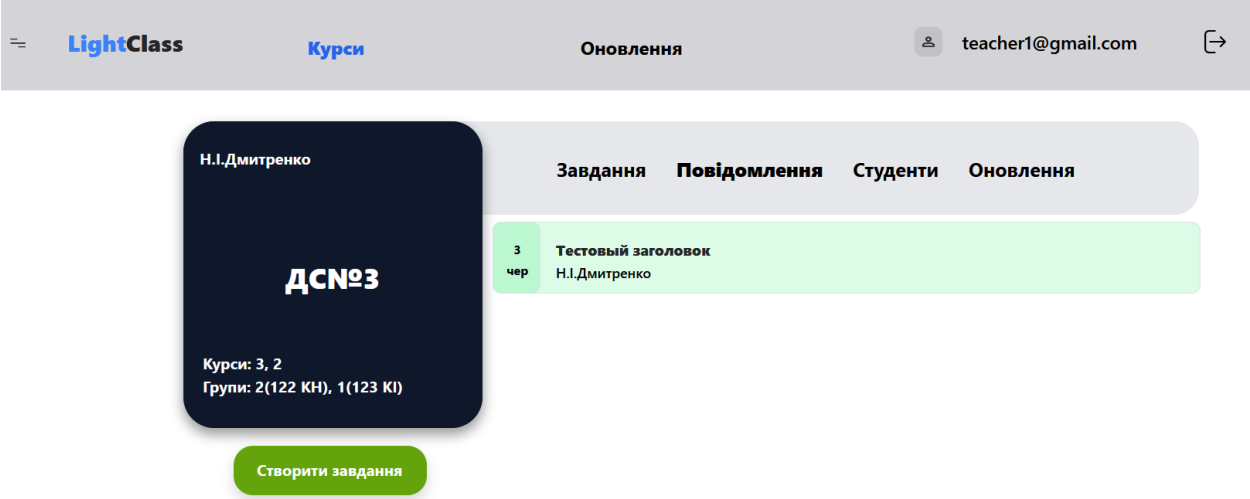

Рисунок 9.11 – Список повідомлень

Натиснувши на завдання, відкриється сторінка з описом завдання та списком студентів, що його виконали (рис. 9.12). На цій сторінки викладач може видалити завдання, натиснувши відповідну кнопку, та переглянути список усіх студентів, що надіслали роботи за цим завданням.

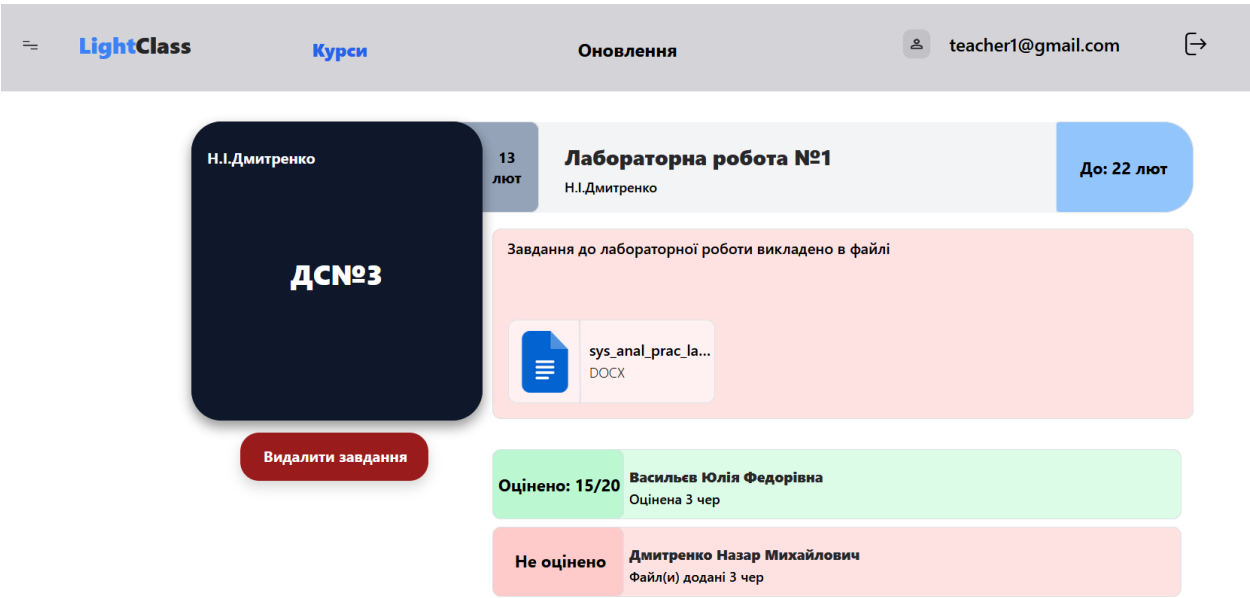

Рисунок 9.12 – Сторінка завдання зі списком студентів що його виконало

Для зручності зеленим кольором візуально позначається студента, роботу якого вже оцінено, а студентів, роботи яких ще не оцінено – червоним.

Натиснувши на блок студента відкриється сторінка оцінювання роботи студента (рис. 9.13).

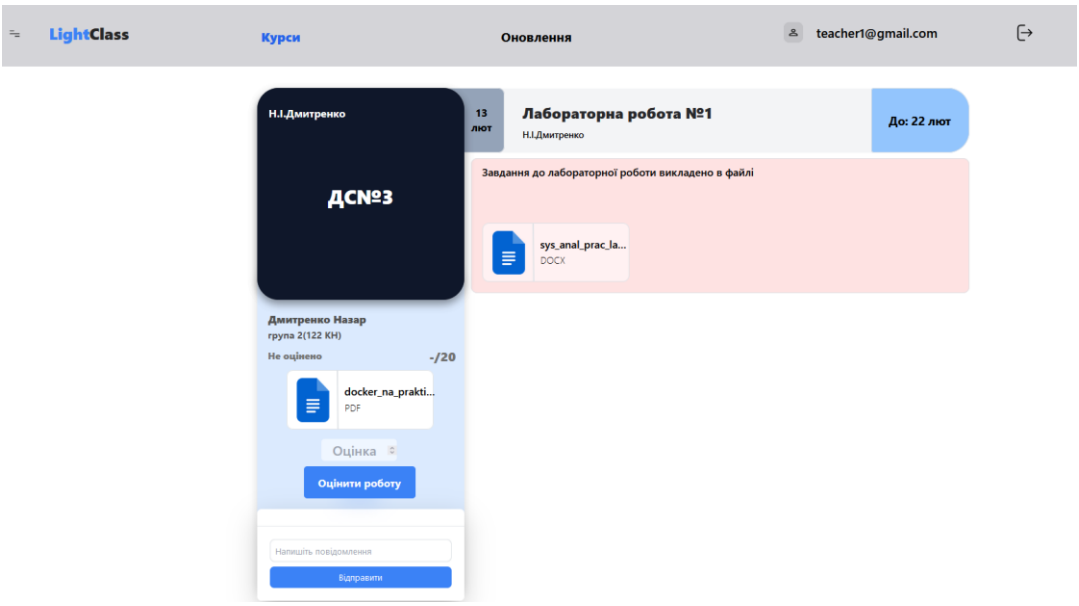

Рисунок 9.13 – Сторінка оцінювання роботи студента та чату з ним

Після перегляду файлів студента, які можливо завантажити натиснувши на блоки з ними викладач може виставити оцінку за цю роботу.

Також викладач може переглянути список оновлень за цим предметом (рис. 9.14) та список учнів (рис. 9.15).

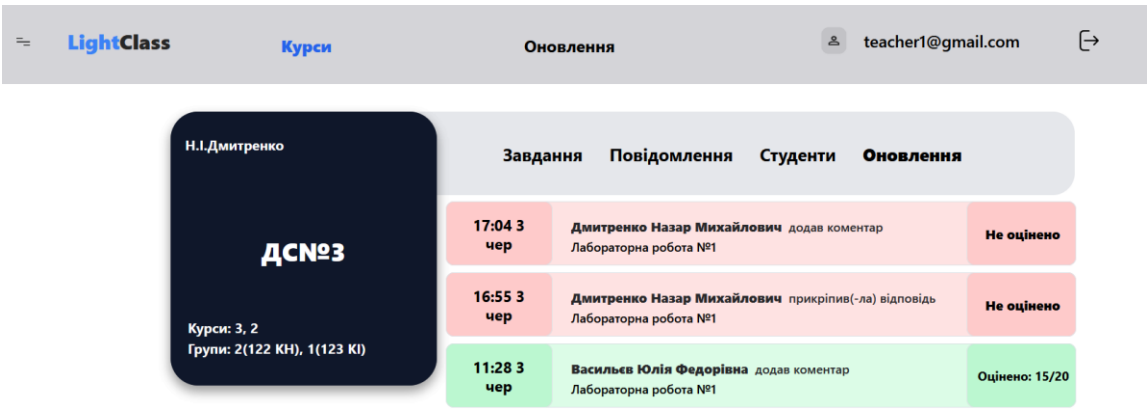

Рисунок 9.14 – Список оновлень за предметом

Якщо студент додасть повідомлення у чаті з завданням чи додає файли до завдання, ця інформація буде відображатись на цій сторінки. При натисканні на відповідний блок відчиниться сторінка з оцінюванням роботи.

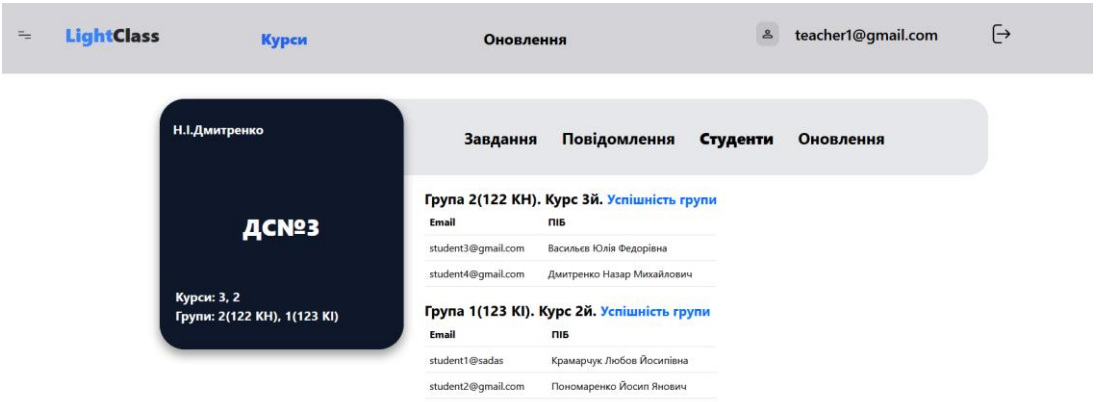

Рисунок 9.15 – Список студентів за предметом

На цій сторінки відображається навчальні групи та списки студентів що належать до них. При натисканні на кнопку “Успішність групи” відкриється сторінка з успішністю навчальної групи за цим предметом (рис. 9.16).

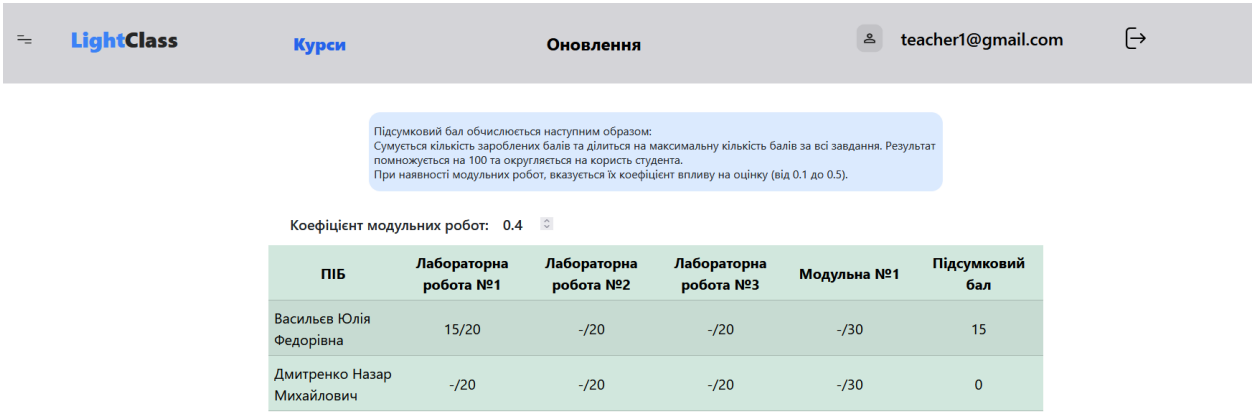

Рисунок 9.16 – Сторінка успішності навчальної групи за предметом

На цій сторінці відображається оцінки студентів за завдання та підсумковий бал за проходження практичної частини курсу. Підсумковий бал обчислюється наступним образом: сумується кількість зароблених балів та ділиться на максимальну кількість балів за всі завдання. Результат помножується на 100 та округляється на користь студента. При наявності модульних робот, вказується їх коефіцієнт впливу на оцінку (від 0.1 до 0.5).

Також викладач може переглянути список оновлень за усі предмети, де він є викладачем (рис. 9.17) через меню навігації на верху сторінки. У даному списку перелічені студенти які прикріпили файл до завдання чи додали коментар у чаті. Також вказується предмет, завдання, час оновлення та ПІБ студента. Натиснувши на блок, викладач потрапляє до сторінки з оцінюванням роботи відповідного студента.

Рисунок 9.17 – Список оновлень за всі предмети

Розглянемо сторінки для адміністратора. Після успішної авторизації адміністратор бачить сторінку керування, а саме зі списком факультетів (рис. 9.18).

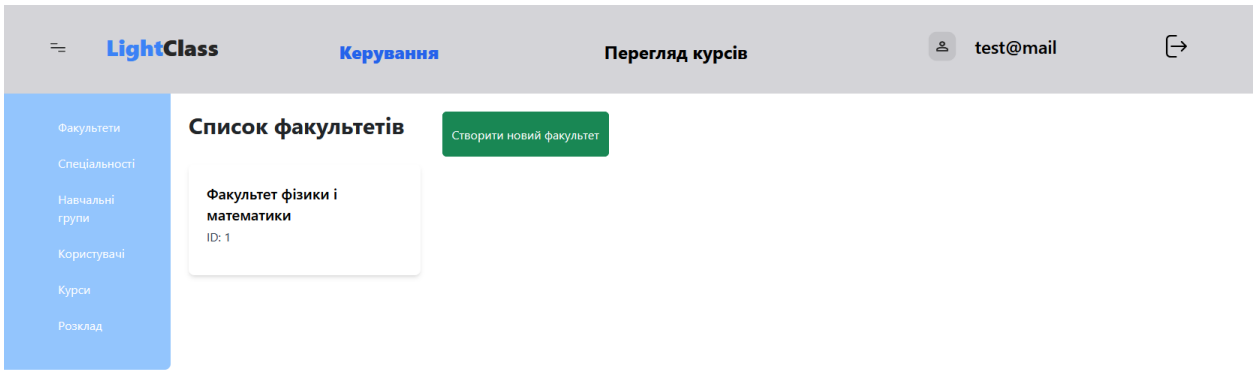

Рисунок 9.18 – Список факультетів

На цій сторінці перелічено факультети. При натисканні на кнопку “Створити факультет” відчиниться сторінка з формою створення нового факультету (рис. 9.19). Для відкриття сторінки з оновленням факультета (рис. 9.20) необхідно натиснути на блок з інформацією про факультет який необхідно оновити.

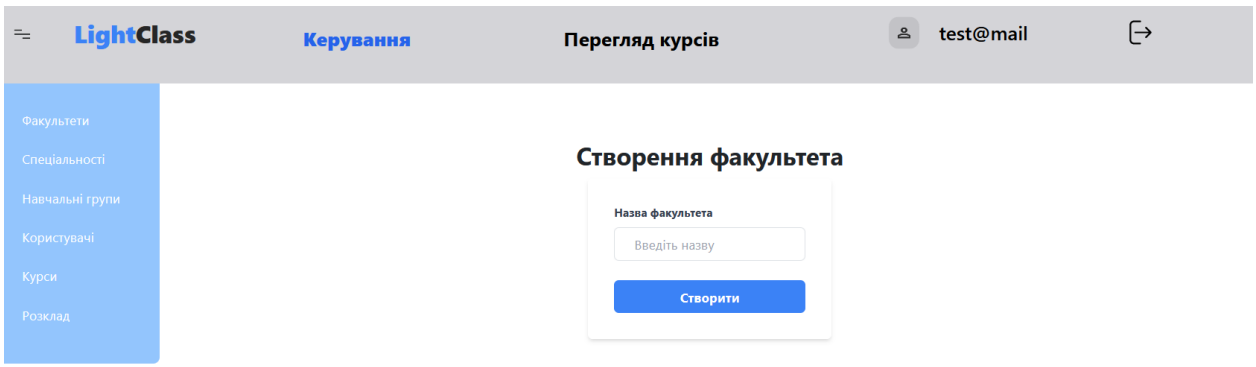

Рисунок 9.19 – Форма додавання нового факультету

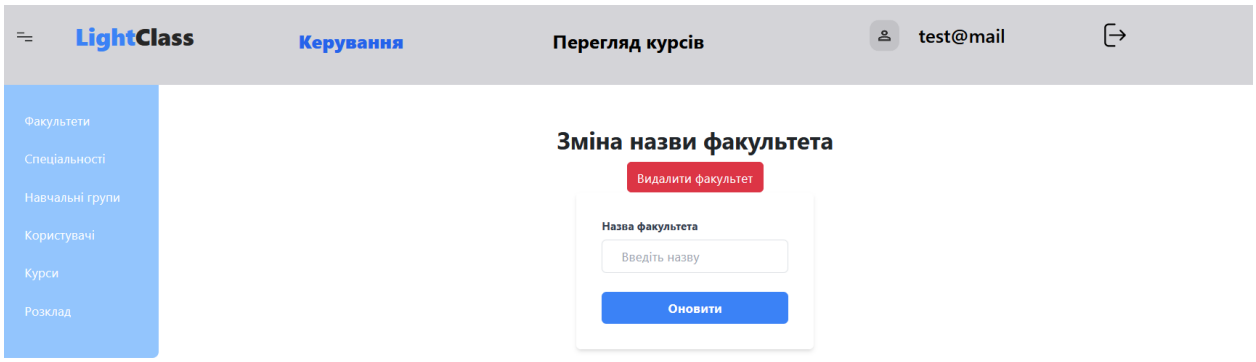

Рисунок 9.20 – Форма оновлення факультету

При натисканні на кнопку “Видалити факультет” обраний факультет буде видалено.

Для перегляду списку спеціальностей (рис. 9.21) необхідно натиснути відповідну кнопку з меню що находиться зліва.

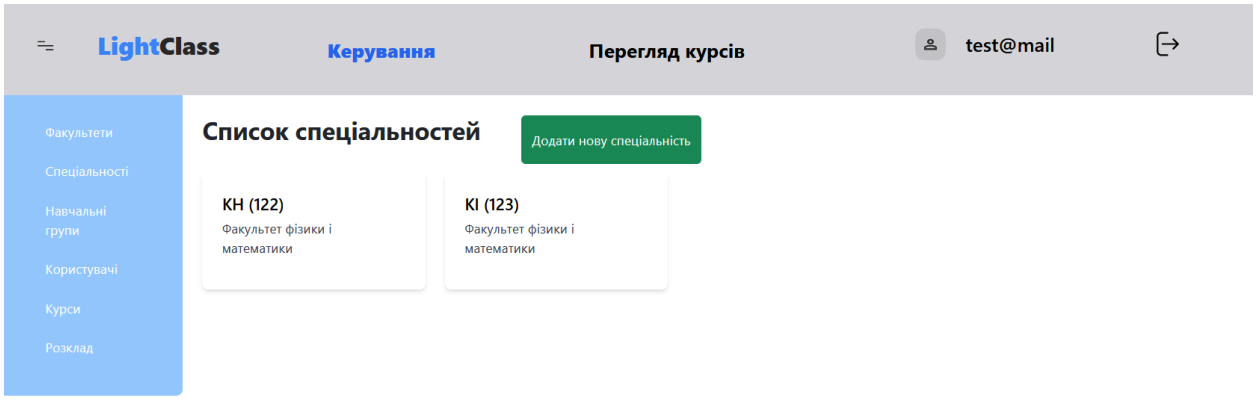

Рисунок 9.21 – Список спеціальностей

На цій сторінці перелічено спеціальності. При натисканні на кнопку “Додати нову спеціальність” відчиниться сторінка з формою створення нової спеціальності (рис. 9.22). Для відкриття сторінки з оновленням спеціальності

(рис. 9.23) необхідно натиснути на блок з інформацією про спеціальність яку необхідно оновити.

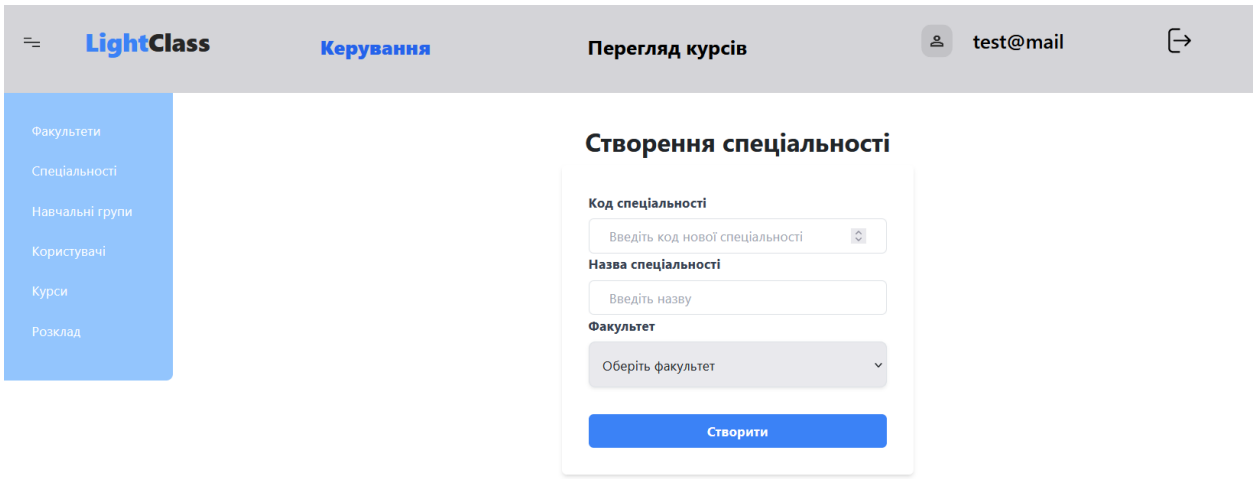

Рисунок 9.22 – Форма додавання нової спеціальності

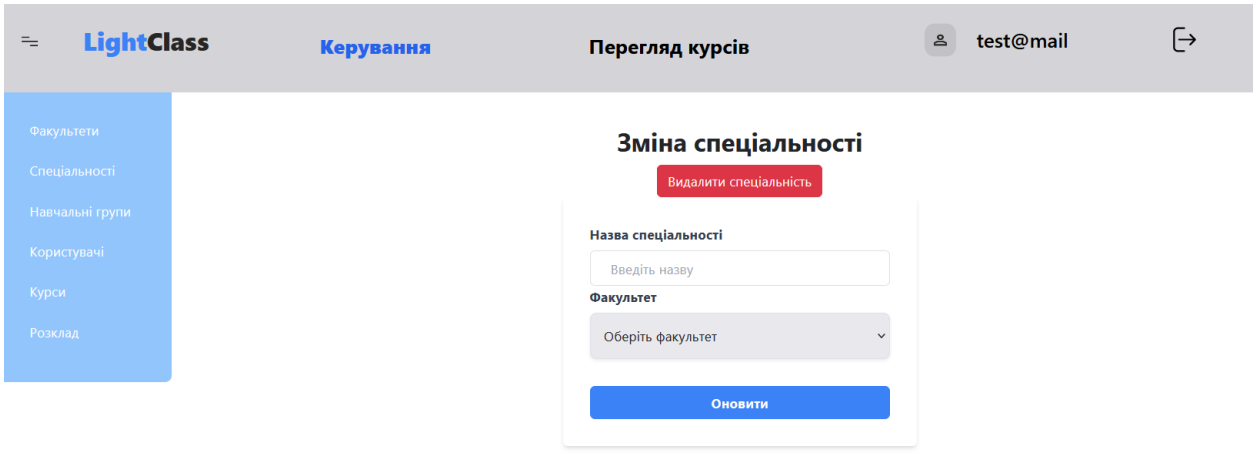

Рисунок 9.23 – Форма оновлення спеціальності

При натисканні на кнопку “Видалити спеціальність” обрану спеціальність буде видалено.

Для перегляду списку навчальних груп (рис. 9.24) необхідно натиснути відповідну кнопку з меню що находиться зліва.

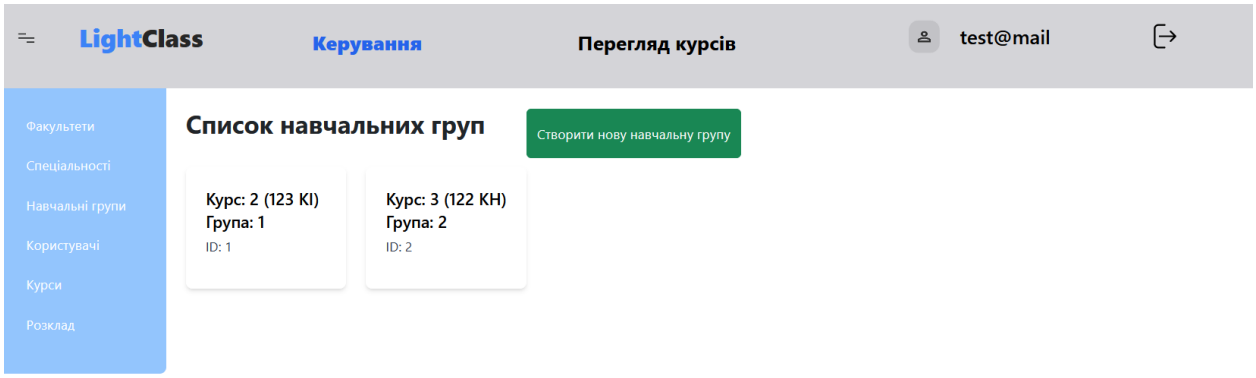

Рисунок 9.24 – Список навчальних груп

На цій сторінці перелічено навчальні групи. При натисканні на кнопку “Створити нову навчальну групу” відчиниться сторінка з формою створення нової навчальної групи (рис. 9.25). Для відкриття сторінки з оновленням навчальної групи (рис. 9.26) необхідно натиснути на блок з інформацією про навчальну групу яку необхідно оновити.

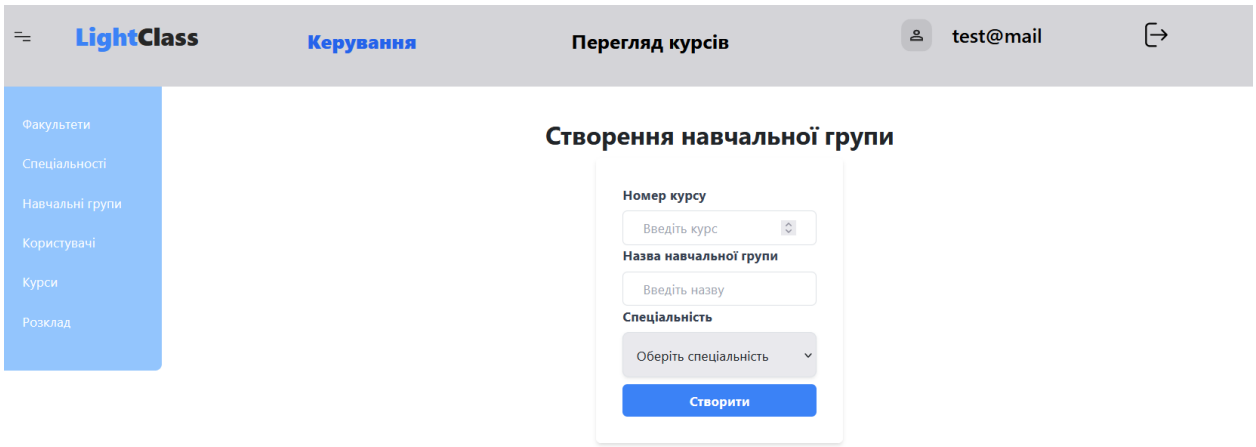

Рисунок 9.25 – Форма створення нової навчальної групи

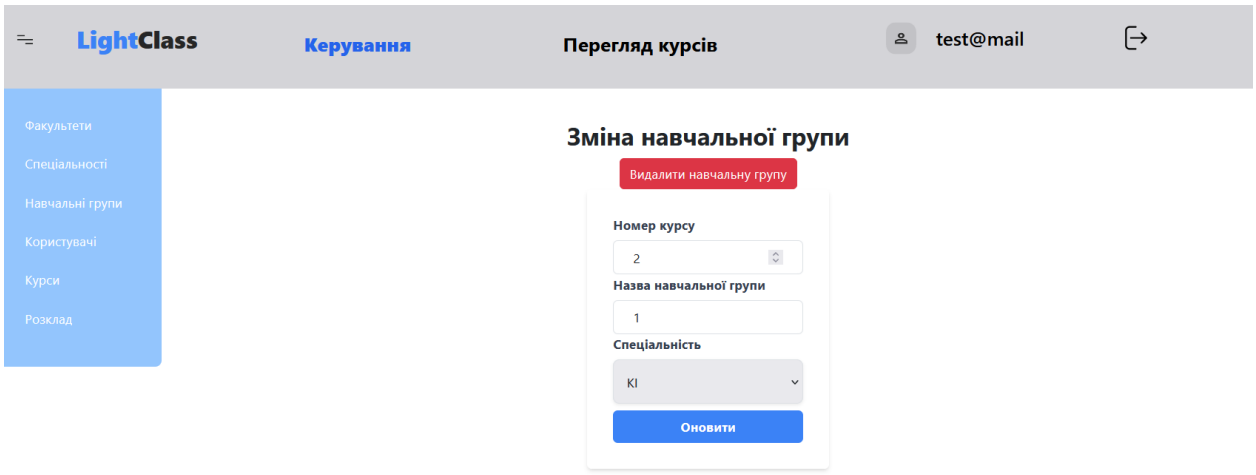

Рисунок 9.26 – Форма оновлення навчальної групи

При натисканні на кнопку “Видалити навчальну групу” обрану навчальну групу буде видалено.

Для перегляду списку студентів (рис. 9.27) необхідно натиснути кнопку “Користувачі” з меню що находиться зліва.

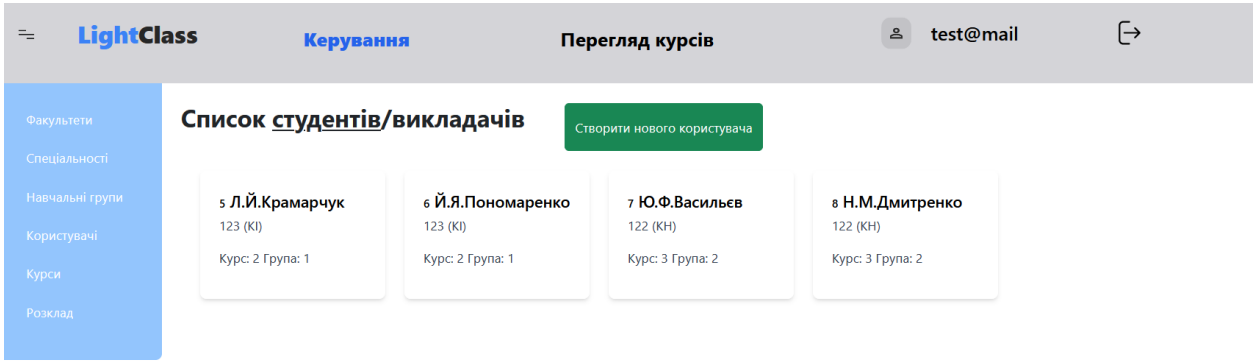

Рисунок 9.27 – Список студентів

На цій сторінці перелічено студентів. При натисканні на кнопку “Створити нового користувача” відчиниться сторінка з формою створення нового користувача (рис. 9.28). Для відкриття сторінки з оновленням студента

(рис. 9.29) необхідно натиснути на блок з інформацією про студента якого необхідно оновити.

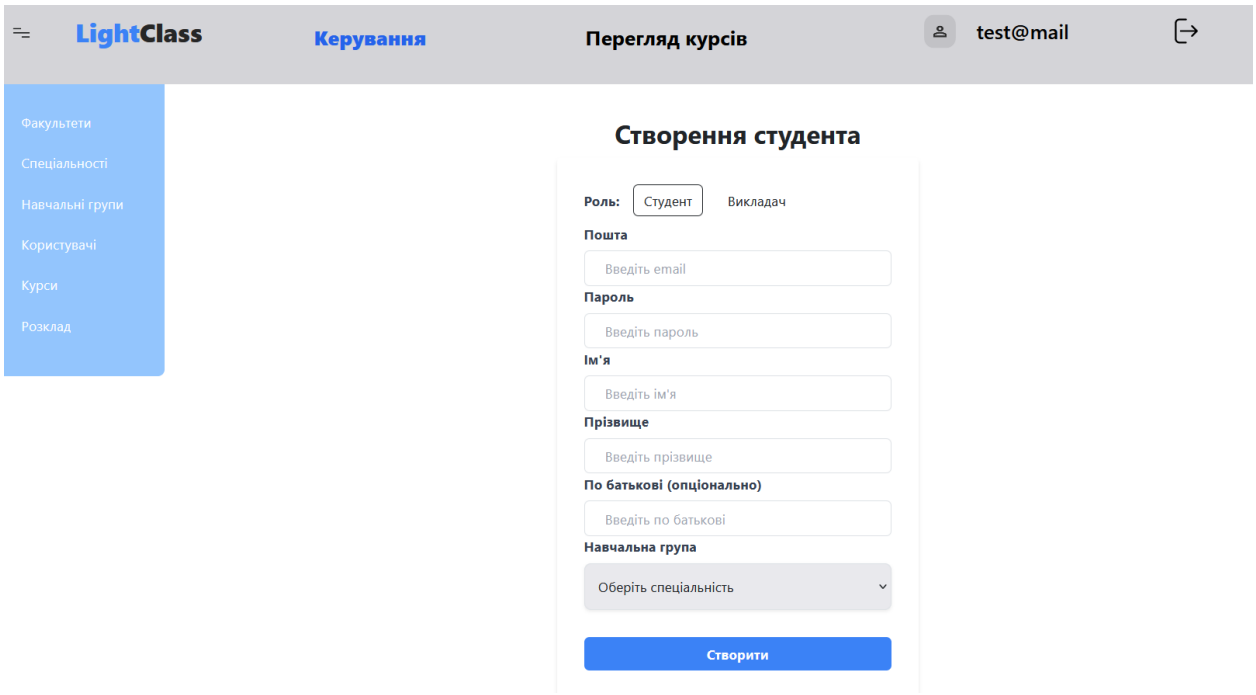

Рисунок 9.28 – Форма створення нового користувача

У відповідності з обраною ролью користувача, необхіно заповнити різні дані, для викладача це пошта, пароль, та ПІБ, а для студента ще необхідно вказати навчальну групу.

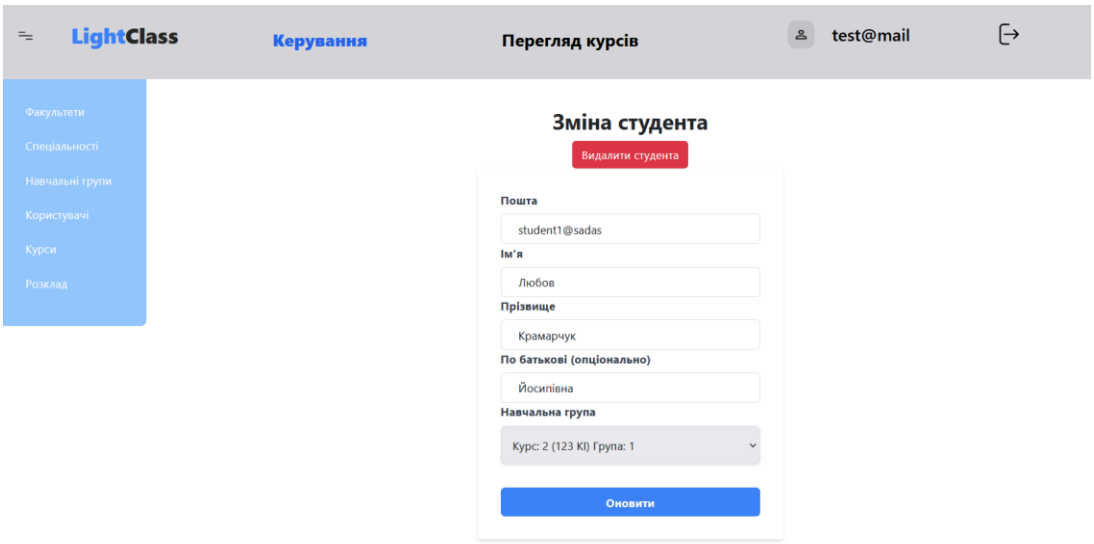

Рисунок 9.29 – Форма оновлення студента

При натисканні на кнопку “Видалити студента” обраного студента буде видалено.

Для перегляду списку викладачів (рис. 9.30) необхідно натиснути кнопку “Користувачі” з меню що находиться зліва та у заголовку обрати викладачів.

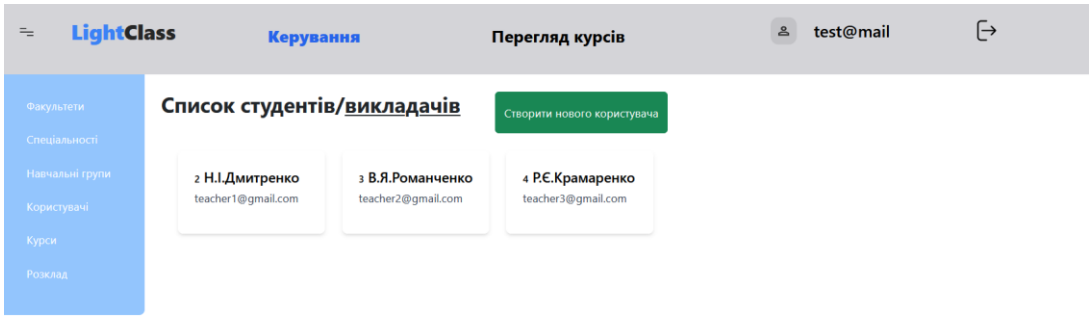

Рисунок 9.30 – Список викладачів

На цій сторінці перелічено викладачів. Для відкриття сторінки з оновленням викладача (рис. 9.31) необхідно натиснути на блок з інформацією про викладача якого необхідно оновити.

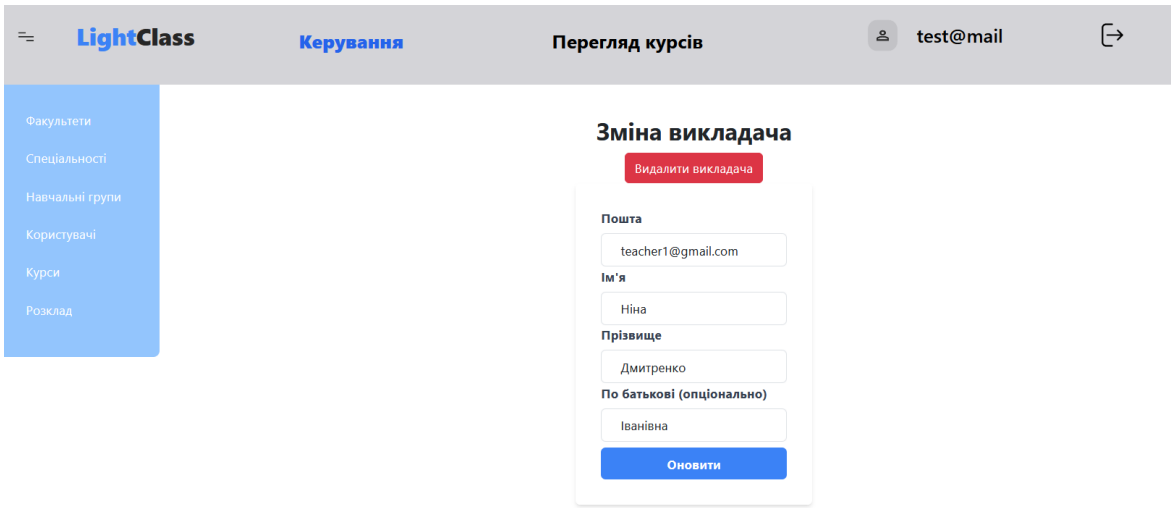

Рисунок 9.31 – Форма оновлення викладача

Для перегляду списку предметів (рис. 9.32) необхідно натиснути кнопку “Курси” з меню що находиться зліва.

Рисунок 9.32 – Список предметів

На цій сторінці перелічено предмети. При натисканні на кнопку “Створити новий предмет” відчиниться сторінка з формою створення нового предмету (рис. 9.33). Для відкриття сторінки з оновленням предмета (рис. 9.34) необхідно натиснути на блок з інформацією про предмет який необхідно оновити.

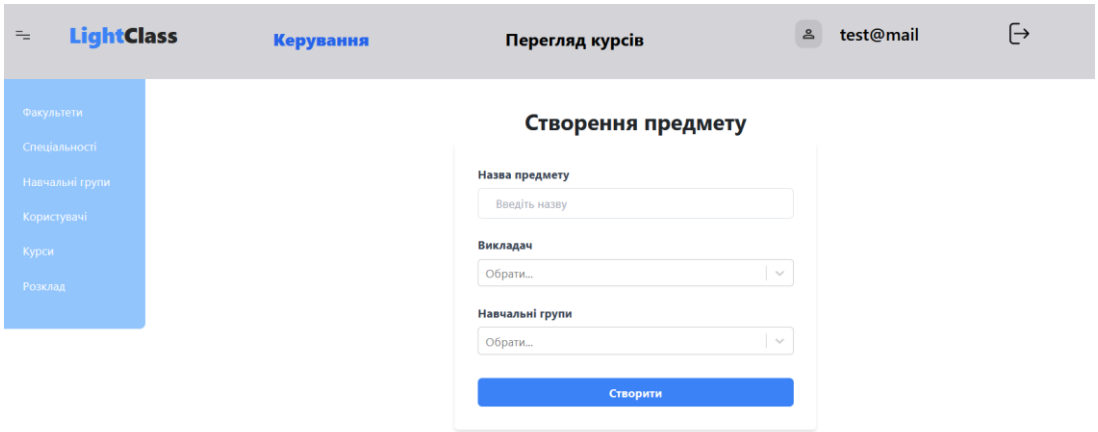

Рисунок 9.33 – Форма створення нового предмета

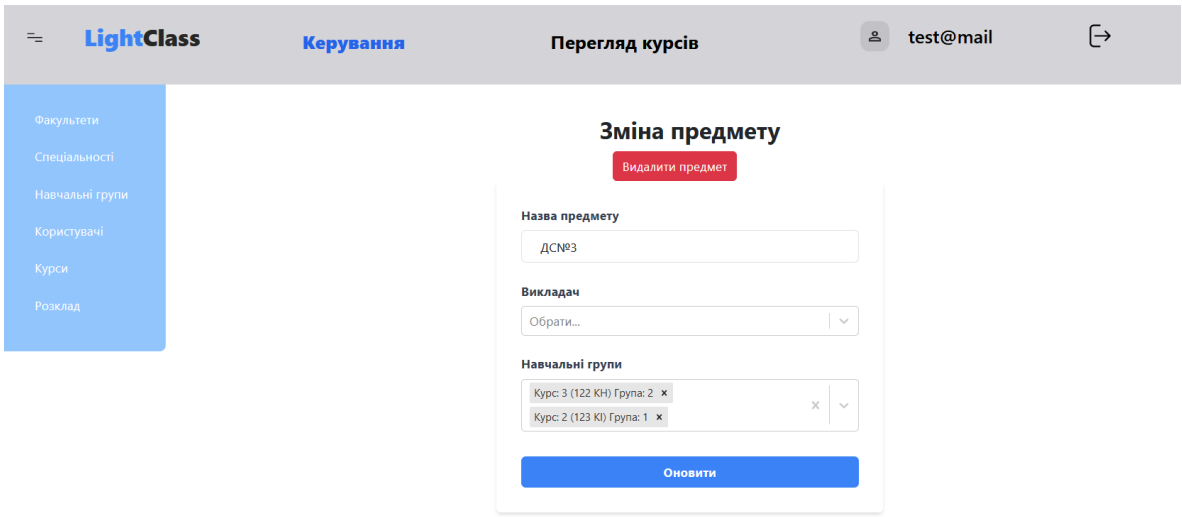

Рисунок 9.34 – Форма оновлення предмета

При натисканні на кнопку “Видалити предмет” обраний предмет буде видалено.

Для перегляду розкладу (рис. 9.35) необхідно натиснути кнопку “Розклад” з меню що находиться зліва та обрати навчальну групу.

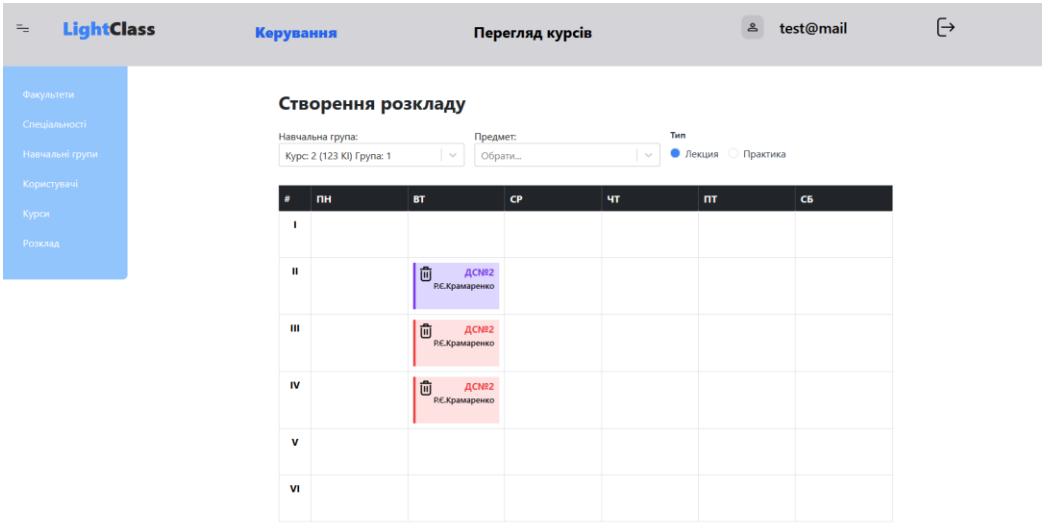

Рисунок 9.35 – Розклад навчальної групи

На цій сторінці можна змінити розклад для обраної навчальної групи. Для додавання предмету у розклад, необхідно обрати предмет зі списку,

обрати тип пари (лекція або практика) та натиснути на бажану комірку. Для вилучення предмета з розкладу необхідно натиснути на кнопку смітника.

Також адміністратор може переглядати предмети навчальних груп (рис. 9.36) через меню навігації на верху сторінки. Обравши навчальну групу, зв’яться список предметів, які викладаються для неї. Натиснувши на блок з предметом адміністратор потрапляє на сторінку предмета (рис. 9.37). Далі сторінки та функціонал зв’язані з предметом співпадають з ролью викладача за вийнятком відсутності можливості оцінювання робот та спілкування у чатах.

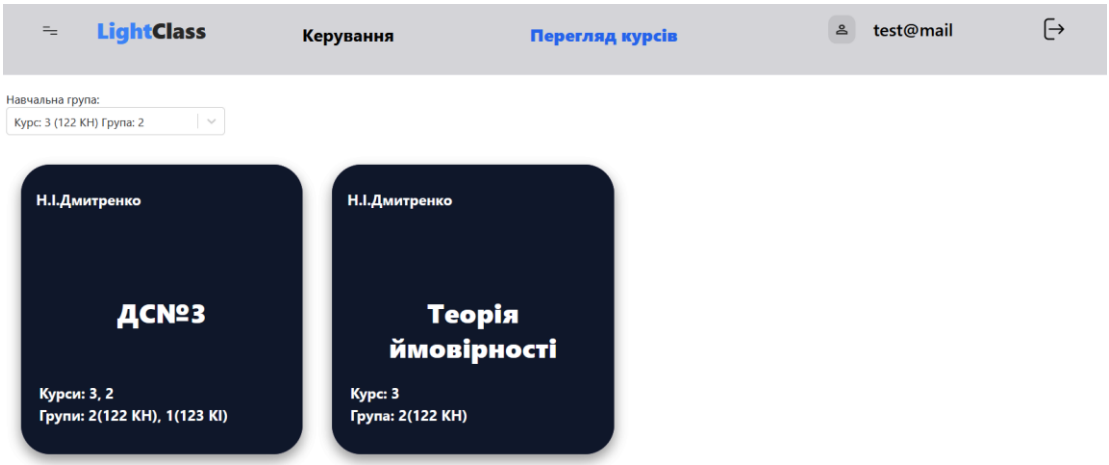

Рисунок 9.36 – Сторінка предметів для навчальної групи

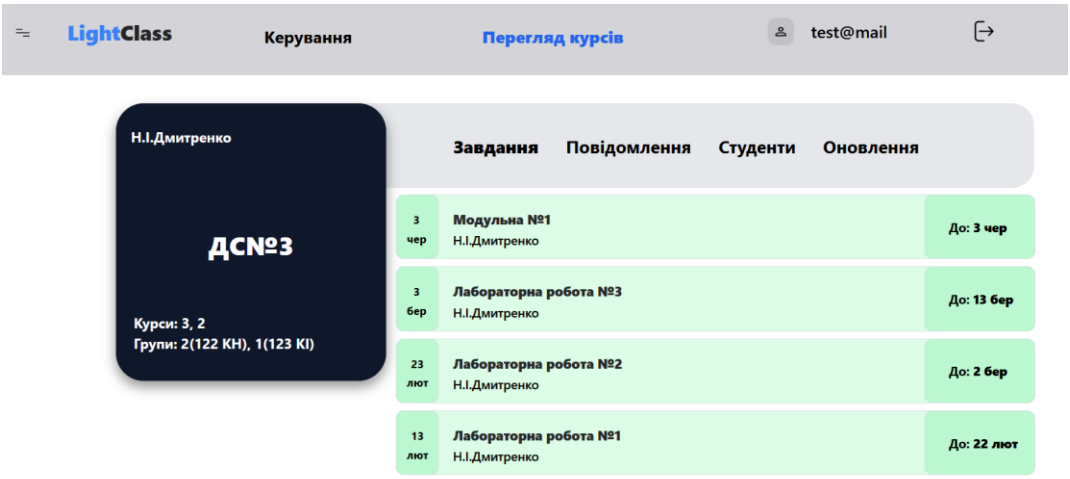

Рисунок 9.37 – Сторінка предмета

Кожний авторизований користувач, незалежно від його ролі, може змінити пароль відкривши відповідну сторінку (рис. 9.38) у меню навігації зліва від логіна користувача. Щоб закінчити поточну сесію необхідно натиснути кнопку праворуч від логіна користувача.

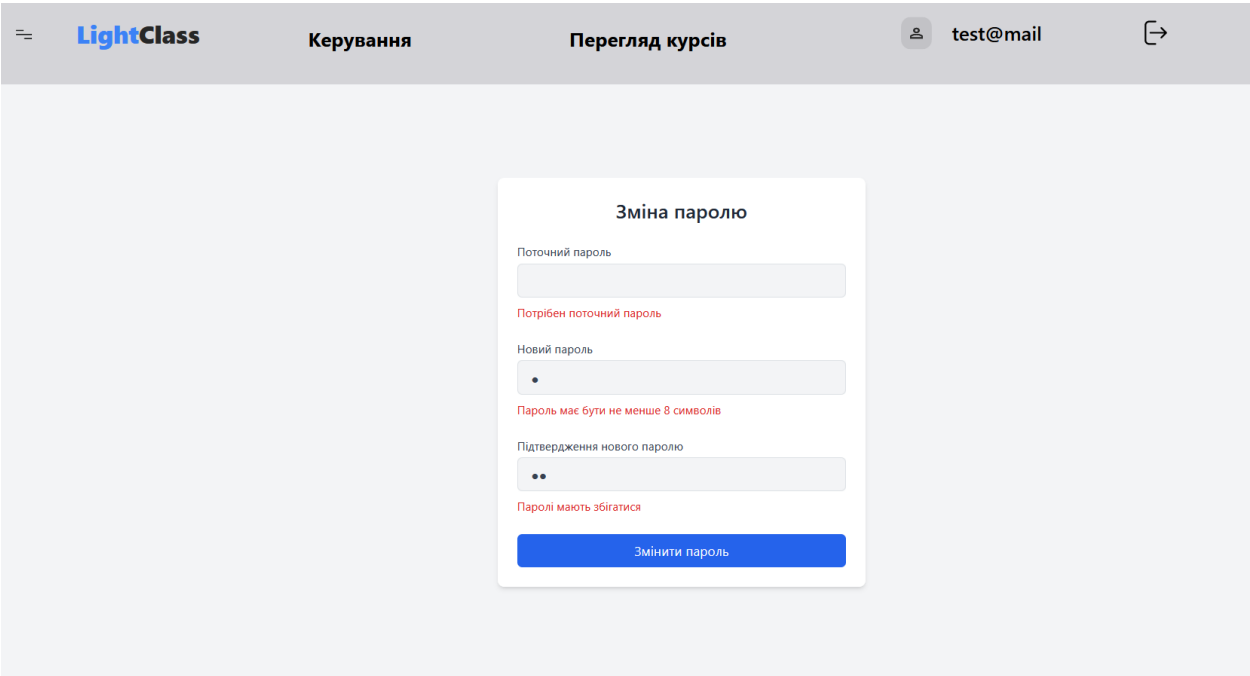

Рисунок 9.38 – Форма зміни паролю

Для зміни паролю необхідно вказати поточний пароль, новий пароль та для підтвердження повторити його.

ВИСНОВКИ

В результаті аналізу предметної області сформульована мета створення ІС, спроектовано і реалізовано інформаційну систему обліку студентів і їх успішності, визначено список категорій її користувачів (Студент, Викладач, Адміністратор) і задач, які користувачі повинні вирішувати за допомогою ІС. Сформовані вимоги до даних і спроектована база даних для зберігання і маніпулювання даними предметної області.

Для створення ІС обрані трирівнева архітектура клієнт-сервер з застосуванням RESTfull API, шаблон проєктування MVC, СУБД PostgreSQL, для серверної частини мова програмування Java і Spring Boot, для клієнтської частини Javascript і React для розробки дружнього користувальницького інтерфейсу. Інтерфейс клієнтської програми розроблений з урахуванням вимог кожної категорії користувачів і надає необхідний функціонал для розв'язання відповідних задач. Доступ до даних з боку різних категорій користувачів розмежований за допомогою механізму ролей і привілеїв. Захист від несанкціонованого доступу реалізований шляхом використання механізму автентифікації та авторизації.

Інтерфейс клієнтської програми розроблений з урахуванням вимог кожної категорії користувачів і надає необхідний функціонал для розв'язання відповідних задач.

При застосуванні даної системи вона дозволить підвищити ефективність роботи вищих учбових закладів, покращити учбовий процес та забезпечити надійний обмін файлів и повідомлень між студентами та викладачами.

Дана система має чималі перспективи. Вона може розширюватись, оскільки до неї можна додавати нові модулі організації, а завдяки використання RESTfull API – створювання мобільного застосунку.

В майбутньому можливо реалізувати:

- можливість завантаження рейтингу успішності студентів у виді PDF;
- інтернаціоналізація веб-застосунку;

- підключення поштових служб для сповіщень користувачів;
- можливість створювати тести.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Гевлич І. Г. Використання інформаційних систем і технологій в освіті і бізнесі під час пандемії коронавірусу. 2021. С. 185-186.
2. Шух М.С., Михайленко В.С., Інформаційна система обліку студентів та їх успішності / Інформатика, інформаційні системи та технології: тези доповідей двадцять першої всеукраїнської конференції студентів і молодих науковців. Одеса, 26 квітня 2024 р. – Одеса, 2024
3. Lokesh Gupta, What is REST? [Electronic resource] / December 12, 2023 – Access mode: https://restfulapi.net/
4. Prakhar Srivastava, Unlocking the Potential of AWS S3 as a Database: Advantages, Disadvantages, and Practical Implementation Guide [Electronic resource] / July 14, 2023 – Access mode: https://medium.com/@prakhar740/unlocking-the-potential-of-aws-s3-as-a-database-advantages-disadvantages-and-practical-441f40b3c731
5. Difference Between Spring Boot Starter Web and Spring Boot Starter Tomcat [Electronic resource] / Marth 11, 2022 – Access mode: https://www.geeksforgeeks.org/difference-between-spring-boot-starter-web-and-spring-boot-starter-tomcat/
6. React. JavaScript-бібліотека для створення користувацьких інтерфейсів [Електронний ресурс] – Режим доступу: https://uk.legacy.reactjs.org/
7. Axios. Promise based HTTP client for the browser and node.js [Electronic resource] – Access mode: https://axios-http.com/uk/docs/intro
8. Understanding JSON Web Tokens (JWT): A Secure Approach to Web Authentication [Electronic resource] / July 28, 2023 – Access mode: https://medium.com/@extio/understanding-json-web-tokens-jwt-a-secure-approach-to-web-authentication-f551e8d66deb

ДОДАТОК А

Задачі користувачів ІС обліку студентів та їх успішності

Таблиця А.1 – Список задач користувачів ІС учбового порталу

Номер	Задача	Вхідні дані	Вихідні дані
Адміністратор			
А1	Створити факультет	Назва	Новий факультет
А2	Редагувати факультет	Факультет, назва	Оновлений факультет
А3	Видалити факультет	Факультет	Відсутні
А4	Створити спеціальність	Код, назва, факультет	Нова спеціальність
А5	Редагувати спеціальність	Спеціальність, назва, факультет	Оновлена спеціальність
А6	Видалити спеціальність	Спеціальність	Відсутні
А7	Створити навчальну групу	Номер курсу, назва, спеціальність	Нова навчальна група
А8	Редагувати навчальну групу	Навчальна група, номер курсу, назва, спеціальність	Оновлена навчальна групу
А9	Видалити навчальну групу	Навчальна група	Відсутні
А10	Створити викладача	Логін (email), пароль, ПІБ	Новий викладач

Продовження таблиці А.1

Номер	Задача	Вхідні дані	Вихідні дані
A11	Редагувати викладача	Викладач, логін (email), ПІБ	Оновлений викладач
A12	Видалити викладача	Викладач	Відсутні
A13	Створити студента	Логін (email), пароль, ПІБ, навчальна група	Новий студент
A14	Редагувати студента	Студент, логін (email), ПІБ, навчальна група	Оновлений студент
A15	Видалити студента	Студент	Відсутні
A16	Створення предмету	Назва, викладач, навчальні групи	Новий предмет
A17	Редагувати предмет	Предмет, назва, викладач, навчальні групи	Оновлений предмет
A18	Видалити предмет	Предмет	Відсутні
A19	Переглянути розклад навчальної групи	Навчальна група	Розклад навчальної групи
A20	Додати предмет у розклад	Навчальна група, предмет, тип (лекція або практичне заняття), день тиждня, номер пари	Оновлений розклад для навчальної групи

Продовження таблиці А.1

Номер	Задача	Вхідні дані	Вихідні дані
А21	Видалити предмет з розкладу	Навчальна група, день тиждня, номер пари	Оновлений розклад для навчальної групи
А22	Переглянути успішність навчальної групи за предметом	Навчальна група, предмет, коефіцієнт модульних робот	Студенти, оцінки, ітоговий бал за проходження практичного курсу предмета
Викладач			
В1	Створити повідомлення	Предмет, заголовок, опис, файли, дата створення	Нове повідомлення
В2	Видалити повідомлення	Повідомлення	Відсутні
В3	Створити завдання	Предмет, тип завдання (завдання або модульна робота), заголовок, опис, файли, максимальна кількість балів, кінцева дата здачі	Нове завдання
В4	Видалити завдання	Завдання	Відсутні
В5	Переглянути оновленя за предметом	Предмет	Список: дата оновлення, ПІБ студента, завдання, бал (за наявності)

Продовження таблиці А.1

Номер	Задача	Вхідні дані	Вихідні дані
В6	Оцінити роботу	Завдання, студент, кількість балів	Оцінка студента за завдання
В7	Переглянути список студентів за предметом	Предмет	Назва навчальних груп, спеціальності, номер курсу, список студентів: Логін (email), ПІБ
В8	Переглянути успішність навчальної групи за предметом	Навчальна група, предмет, коефіцієнт модульних робот	Студенти, оцінки, ітоговий бал за проходження практичного курсу предмета
В9	Переглянути усі оновлення	Відсутні	Список: дата оновлення, ПІБ студента, предмет, завдання, бал (за наявності)
В10	Надіслати повідомлення у чаті	Завдання, студент, відправник, зміст повідомлення, дата відправки	Нове повідомлення у чаті
Студент			
С1	Переглянути розкладу	Відстутні	Розклад студента: назва предмету, ПІБ викладача, день тижня, номер пари

Продовження таблиці А.1

Номер	Задача	Вхідні дані	Вихідні дані
С2	Переглянути предметів	Відсутні	Список предметів: назва предмету, ПІБ викладача
С3	Переглянути завдань та повідомлень за предметом	Предмет	Список завдань та повідовлень: дата створення, заголовок, ПІБ викладача, кінцева дата здачі(за наявності), отримана кількість балів (за наявності)
С4	Прикріпити файл до завдання	Предмет, завдання, файл(и)	Відсутні
С5	Змінити пароль	Старий пароль, новий пароль	Оновлена інформація про студента
С6	Надіслати повідомлення у чаті	Завдання, студент, відправник, зміст повідомлення, дата відправки	Нове повідомлення у чаті

ДОДАТОК Б

Опис сутностей предметної області

Таблиця Б.1 – Опис сутностей предметної області інтернет провайдер

Ім’я атрибуту	Призначення атрибуту	Обмеження цілістності
Member (Користувач)		
member_id	Ідентифікатор користувача	Первинний ключ, не порожнє
email	Логін (пошта) користувача	Не порожнє, унікальне, шаблон ('^[A-Za-z0-9._%-]+@[A-Za-z0-9.-]+[.][A-Za-z]+$')
password_hash	Пароль користувача у вигляді хеш-значення	Не порожнє
role	Роль користувача	Не порожнє, приймає одне із значень (‘STUDENT, ‘TEACHER’, ‘ADMIN’), за замовчуванням ‘STUDENT’
Personal_Data (Особисті дані)		
data_id	Ідентифікатор особистих даних	Первинний ключ, не порожнє
member	Ідентифікатор користувача	Зовнішний ключ, не порожнє
first_name	Ім’я користувача	Не порожнє
last_name	Прізвище користувача	Не порожнє
patronymic	По-батькові	-

Продовження таблиці Б.1

Ім’я атрибуту	Призначення атрибуту	Обмеження цілістності
Faculty (Факультет)		
faculty_id	Ідентифікатор факультета	Первинний ключ, не порожнє
name	Назва факультута	Не порожнє, унікальне
Major (Спеціальність)		
major_id	Код спеціальності	Первинний ключ, не порожнє
name	Назва спеціальності	Не порожнє, унікальне
faculty_id	Ідентифікатор факультета	Зовнішний ключ, не порожнє
Study_Group (Навчальна група)		
group_id	Ідентифікатор навчальної групи	Первинний ключ, не порожнє
name	Назва навчальної групи	Не порожнє
major	Код спеціальності	Зовнішний ключ, не порожнє
year_of_study	Рік навчання	Не порожнє, примймає значення від 1 до 6
Teacher (Викладач)		
teacher_id	Ідентифікатор викладача	Композитний ключ, не порожнє
Student (Студент)		
student_id	Ідентифікатор студента	Композитний ключ, не порожнє
group_id	Ідентифікатор навчальної групи	Зовнішний ключ, не порожнє
Subject (Предмет)		

Продовження таблиці Б.1

Ім’я атрибуту	Призначення атрибуту	Обмеження цілістності
subject_id	Ідентифікатор предмета	Первинний ключ, не порожнє
main_teacher	Ідентифікатор викладача	Зовнішний ключ, не порожнє
name	Назва предмету	Не порожнє
Subject_Study_Group (Предмети навчальних груп)		
subject_id	Ідентифікатор предмета	Зовнішний ключ, не порожнє
group_id	Ідентифікатор навчальної групи	Зовнішний ключ, не порожнє
		Складений первинний ключ (‘subject_id’, ‘group_id’)
Task (Завдання)		
task_id	Ідентифікатор завдання	Первинний ключ, не порожнє
subject_id	Ідентифікатор предмета	Зовнішний ключ, не порожнє
task	Тип завдання	Не порожнє, приймає одне із значень (‘INFO, ‘LAB, ‘MODULAR’)
title	Заголовок завдання	Не порожнє
description	Опис завдання	Не порожнє
create_date	Дата створення	Не порожнє, не менше поточної дати, за замовчуванням поточна дата

Продовження таблиці Б.1

Ім’я атрибуту	Призначення атрибуту	Обмеження цілістності
Graded_Task (Оцінюваєме завдання)		
task_id	Ідентифікатор завдання	Композитний ключ, не порожнє
max_score	Максимальна кількість балів	Не порожнє, додатне, значення до 100
date_to	Дата завершення	Не порожнє, не менше поточної дати
Score (Оцінка)		
task_id	Ідентифікатор завдання	Зовнішний ключ, не порожнє
student_id	Ідентифікатор студента	Зовнішний ключ, не порожнє
teacher_id	Ідентифікатор викладача	Зовнішний ключ, не порожнє
score_value	Кількість балів	Не порожнє, додатне
evaluation_date	Дата оцінювання	Не порожнє, не менше поточної дати, за замовчуванням поточна дата
		Складений первинний ключ (‘task_id’, ‘student_id’)
Schedule (Розклад)		
schedule_id	Ідентифікатор розкладу	Первинний ключ, не порожнє
group_id	Ідентифікатор навчальної групи	Зовнішний ключ, не порожнє

Продовження таблиці Б.1

Ім’я атрибуту	Призначення атрибуту	Обмеження цілістності
subject_id	Ідентифікатор предмету	Зовнішний ключ, не порожнє
day	День тижня	Не порожнє, приймає значення від 1 до 6
number	Номер пари	Не порожнє, приймає значення від 1 до 6
class_type	Тип пари	Не порожнє, приймає одне із значень (‘LECTURE, ‘PRACTICAL’)
File (Файл)		
file_id	Ідентифікатор файлу	Первинний ключ, не порожнє
link	Посилання на файл	Не порожнє
filename	Назва файлу	Не порожнє
filetype	Тип файлу	Не порожнє
uploaded_date	Дата завантаження	Не порожнє, не менше поточної дати, за замовчуванням поточна дата
Task_File (Прикріплені файли к завданню)		
task_id	Ідентифікатор завдання	Зовнішний ключ, не порожнє
file_id	Ідентифікатор файлу	Зовнішний ключ, не порожнє

Продовження таблиці Б.1

Ім’я атрибуту	Призначення атрибуту	Обмеження цілістності
		Складений первинний ключ (‘task_id’, ‘file_id’)
Student_Task_File (Файли студента до завдання)		
student_id	Ідентифікатор студента	Зовнішний ключ, не порожнє
task_id	Ідентифікатор завдання	Зовнішний ключ, не порожнє
file_id	Ідентифікатор файлу	Зовнішний ключ, не порожнє
		Складений первинний ключ (‘student_id’, ‘task_id’, ‘file_id’)
Chat (Повідомлення у чаті)		
chat_id	Ідентифікатор повідомлення	Первинний ключ, не порожнє
task_id	Ідентифікатор завдання	Зовнішний ключ, не порожнє
student_id	Ідентифікатор студента	Зовнішний ключ, не порожнє
sender	Ідентифікатор відправника	Зовнішний ключ, не порожнє
sent_date	Дата відправки	Не порожнє, не менше поточної дати, за замовчуванням поточна дата
message	Зміст повідомлення	Не порожнє

ДОДАТОК В

Діаграма класів моделей та репозиторіїв

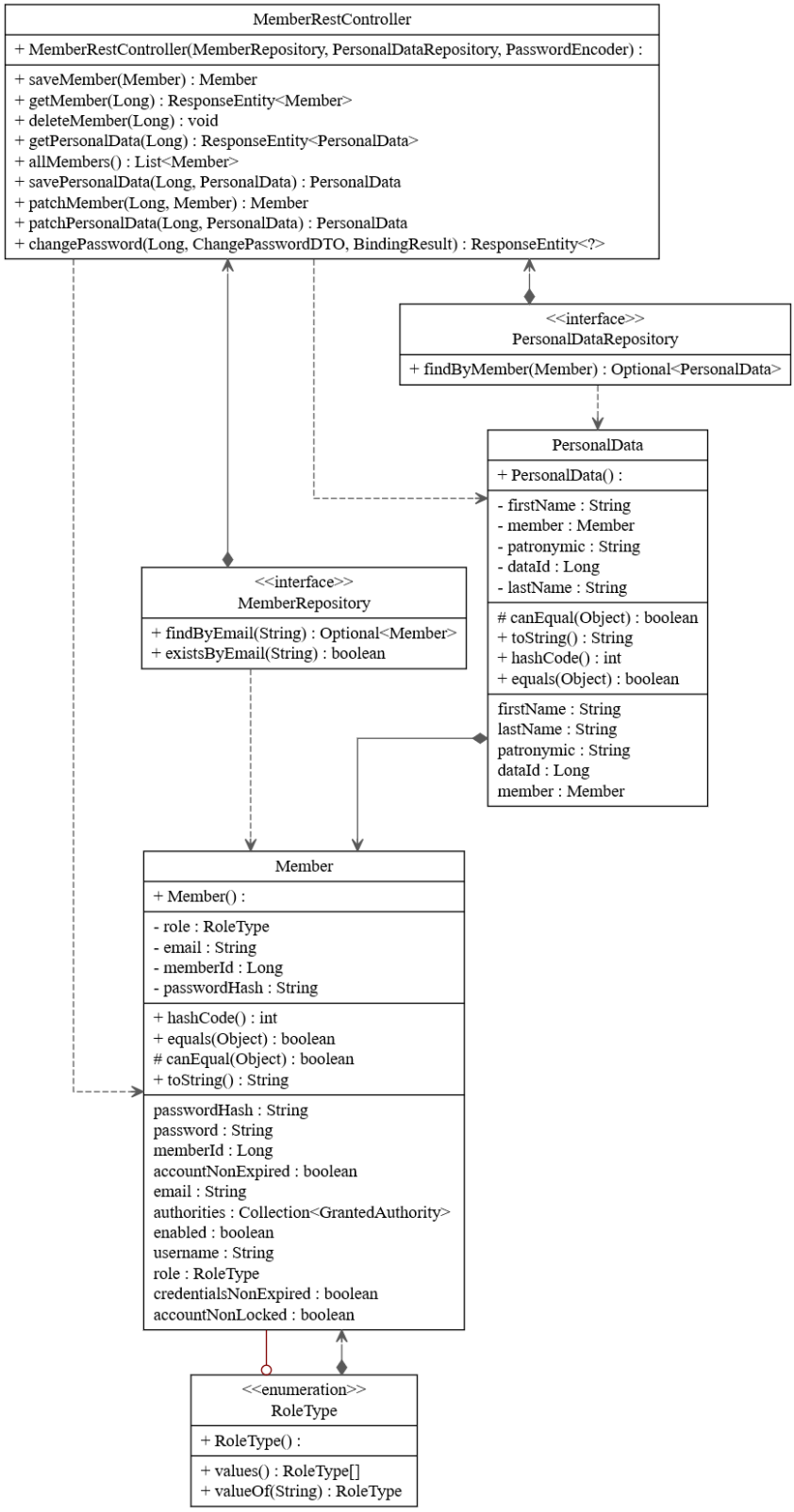

Рисунок В.1 – UML діаграма контролера користувача, об'єктів користувача та персональних даних та їх репозиторіїв

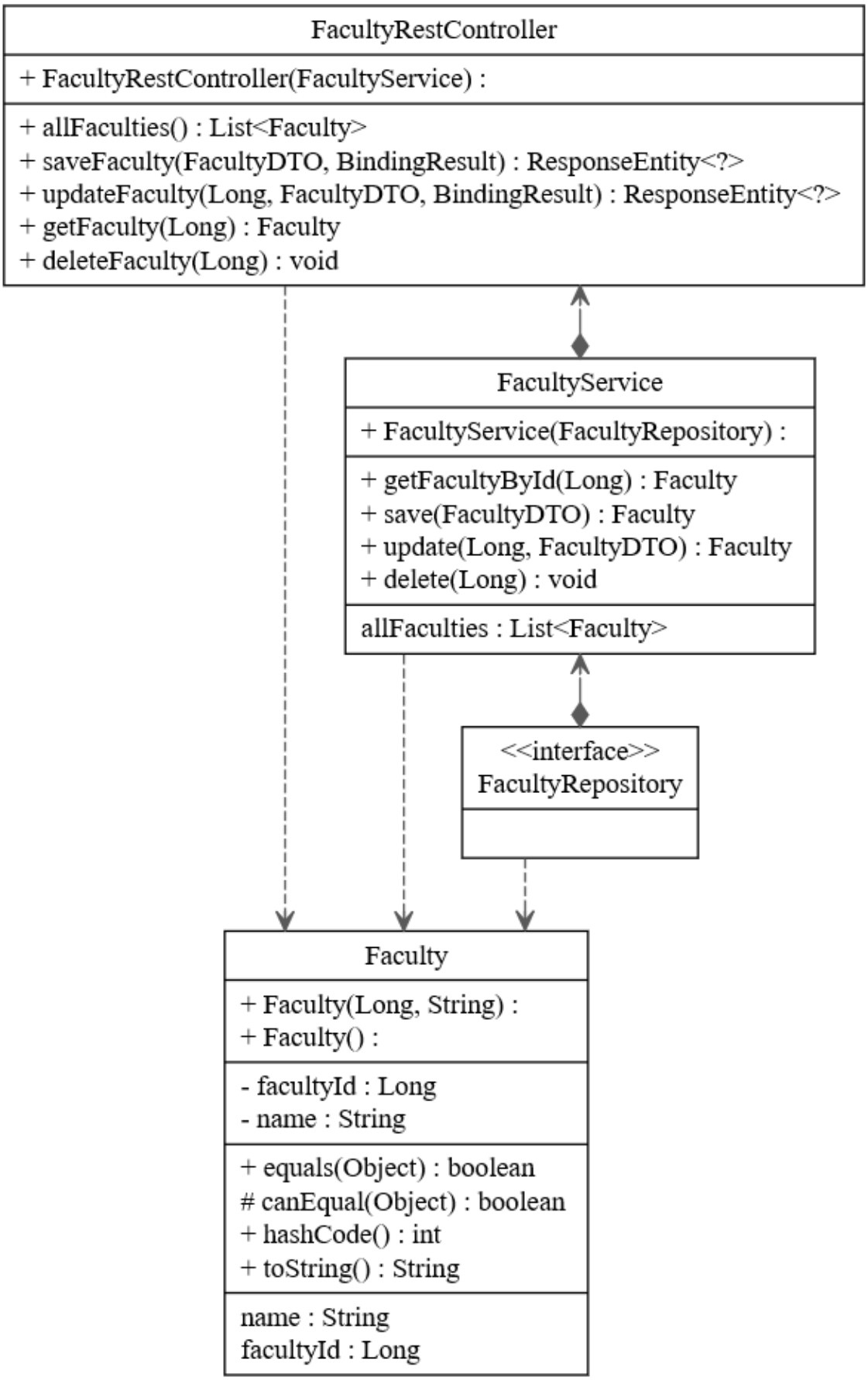

Рисунок В.2 – UML діаграма контролера факультету, об'єкт факультету, його сервіс та репозиторій

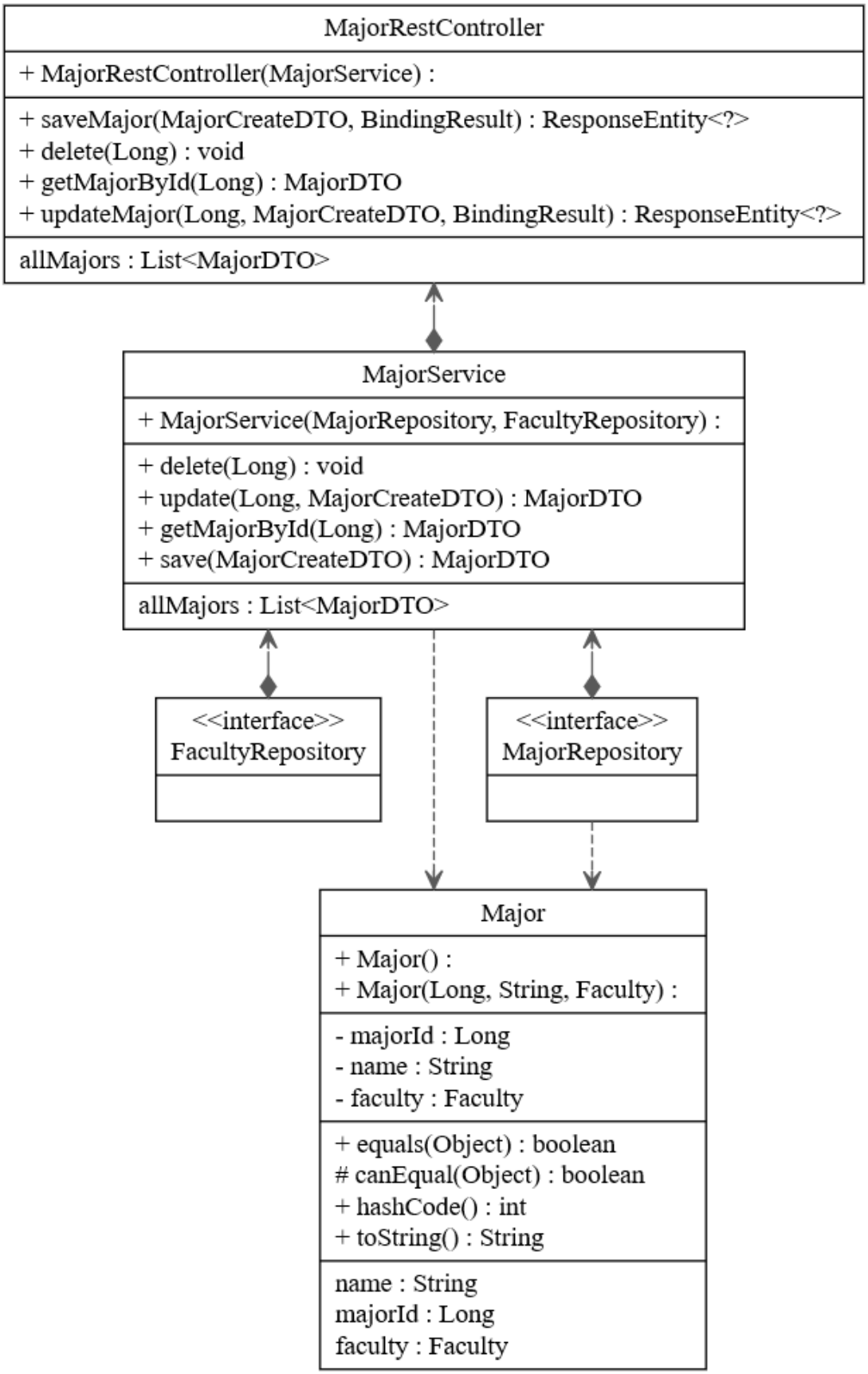

Рисунок В.3 – UML діаграма контролера спеціальності, сервісу спеціальності, об'єкта спеціальності та репозиторіїв спеціальності і факультету

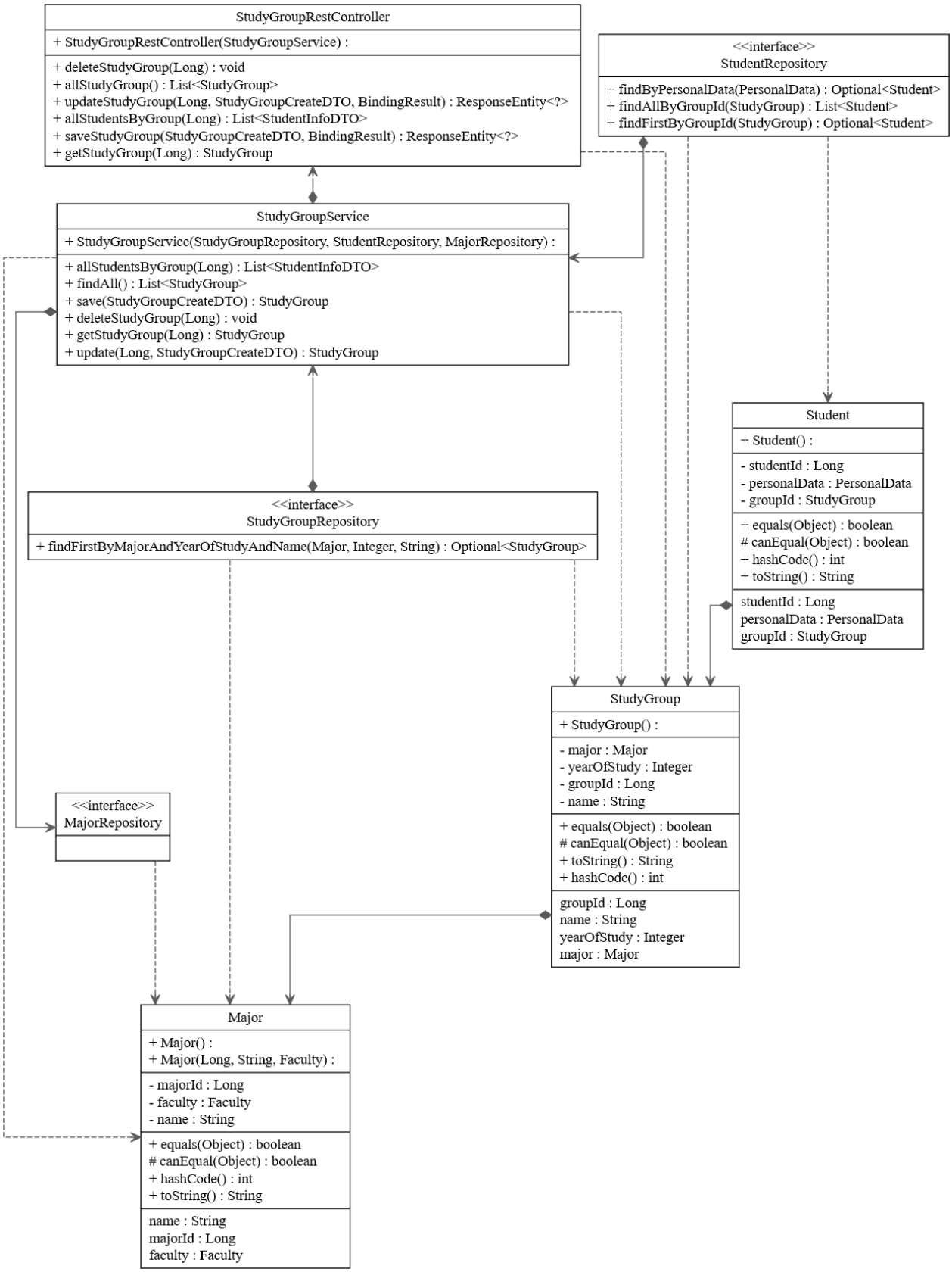

Рисунок В.4 – UML діаграма контролера навчальної групи, її сервіса, об'єктів навчальної групи, студента та спеціальності і використаних репозиторіїв

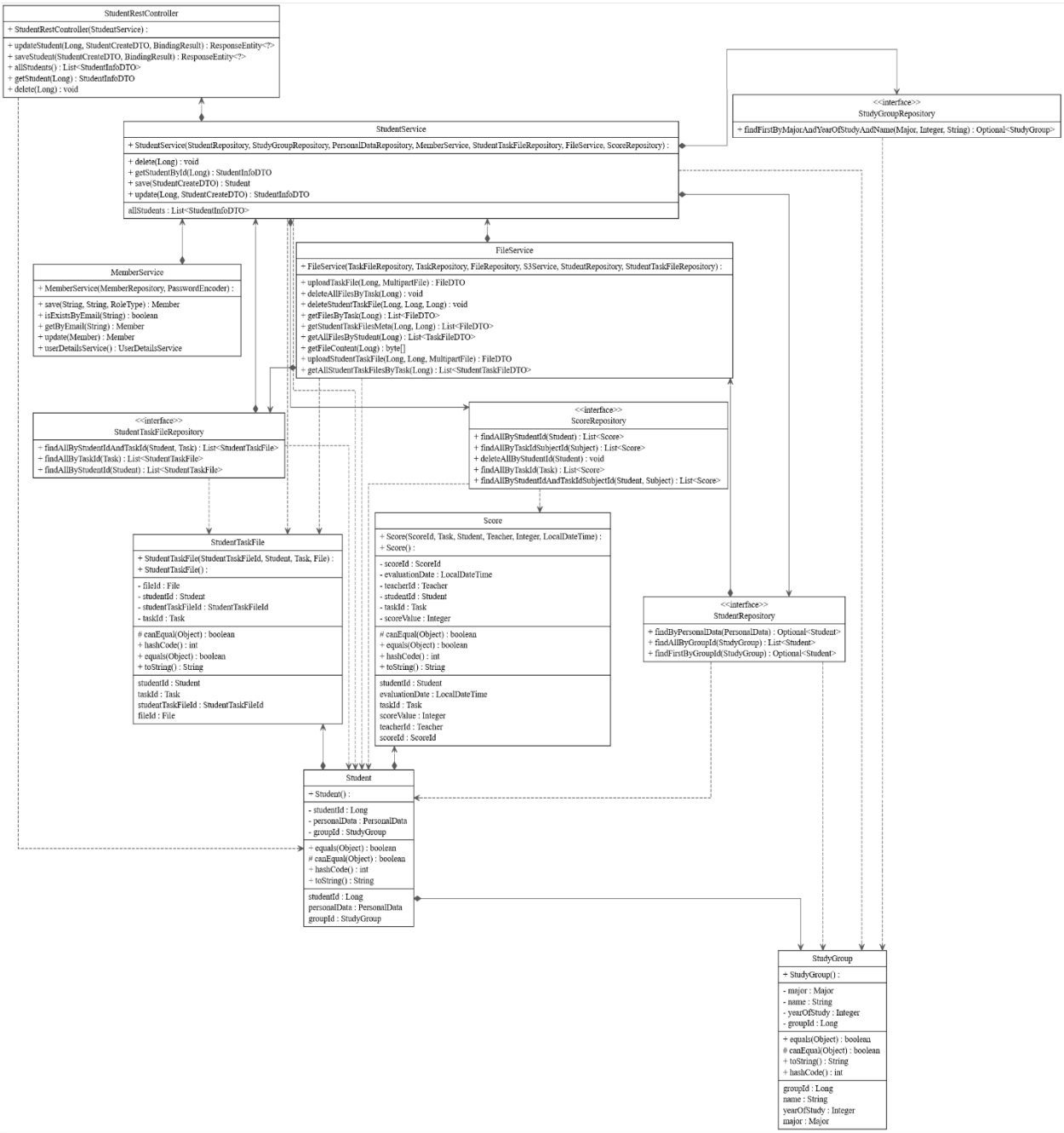

Рисунок В.5 – UML діаграма контролера студента, використаних сервісів та сутностей студента, навчальної групи, оцінки, студентських файлів та їх репозиторіїв

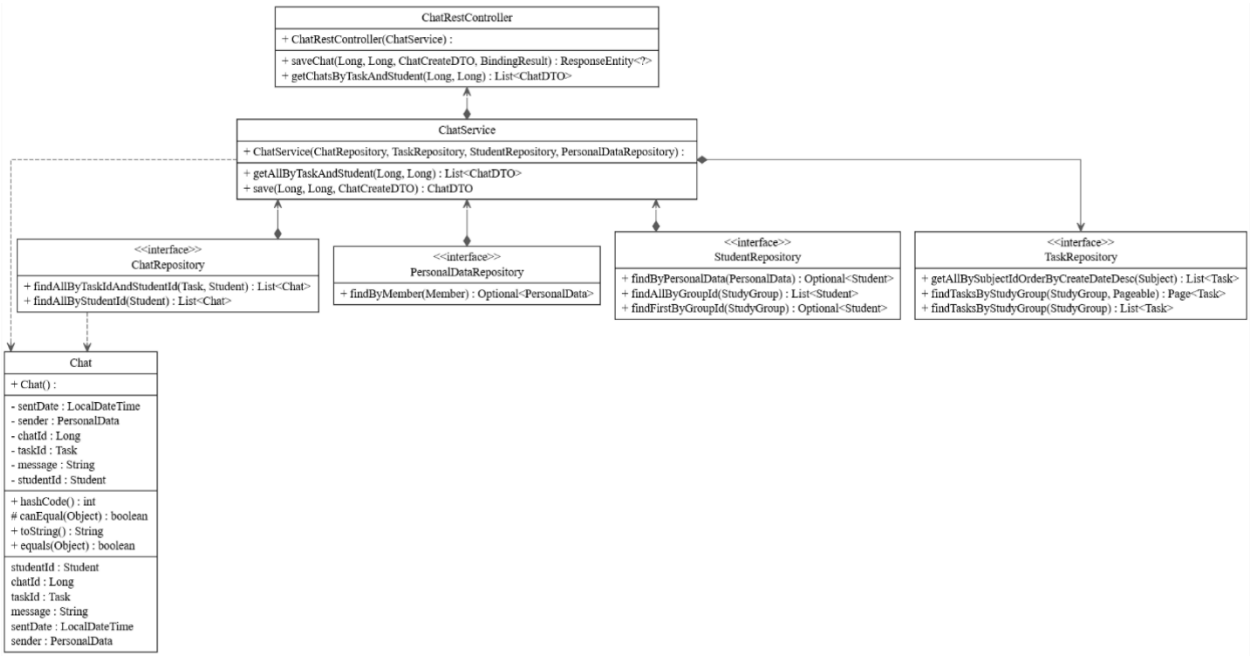

Рисунок В.6 – UML діаграма контролера чату, його об’єкта і використаних сервісів та репозиторіїв

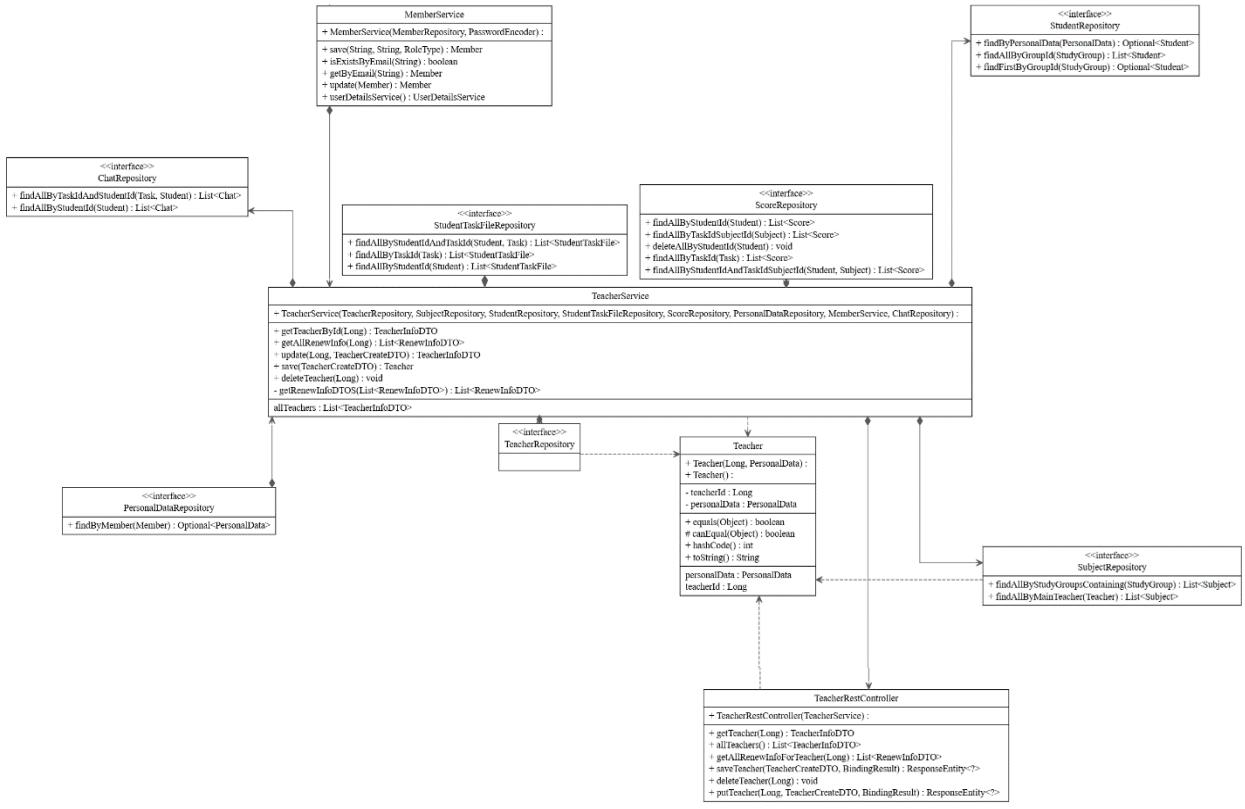

Рисунок В.7 – UML діаграма контролера викладача, його об'єкту і використаних сервісів та репозиторіїв

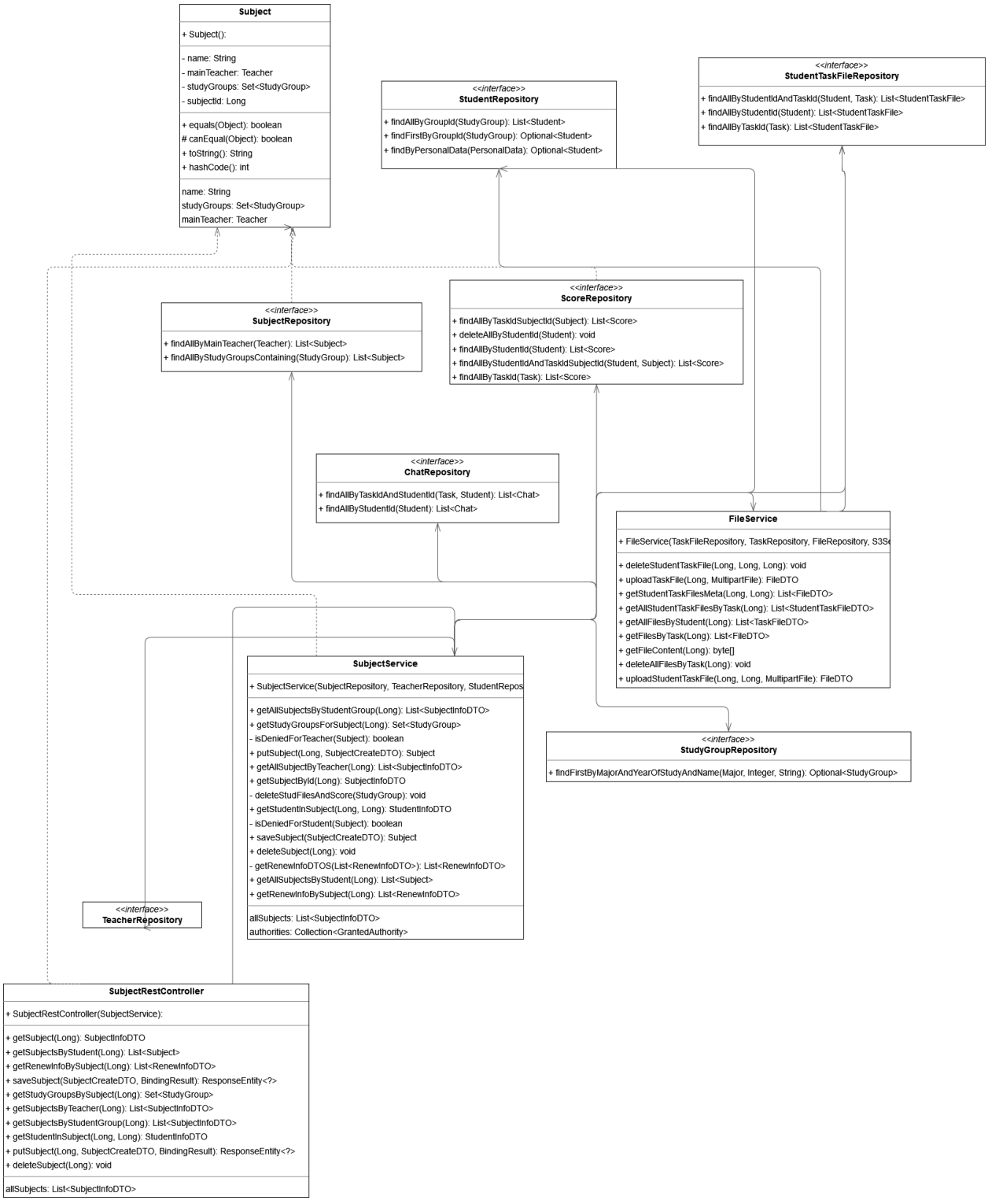

Рисунок В.8 – UML діаграма контролера предмету, його об'єкта і використаних сервісів та репозиторіїв

Рисунок В.9 – UML діаграма контролера завдання, його об'єкту і використаних сервісів та репозиторіїв

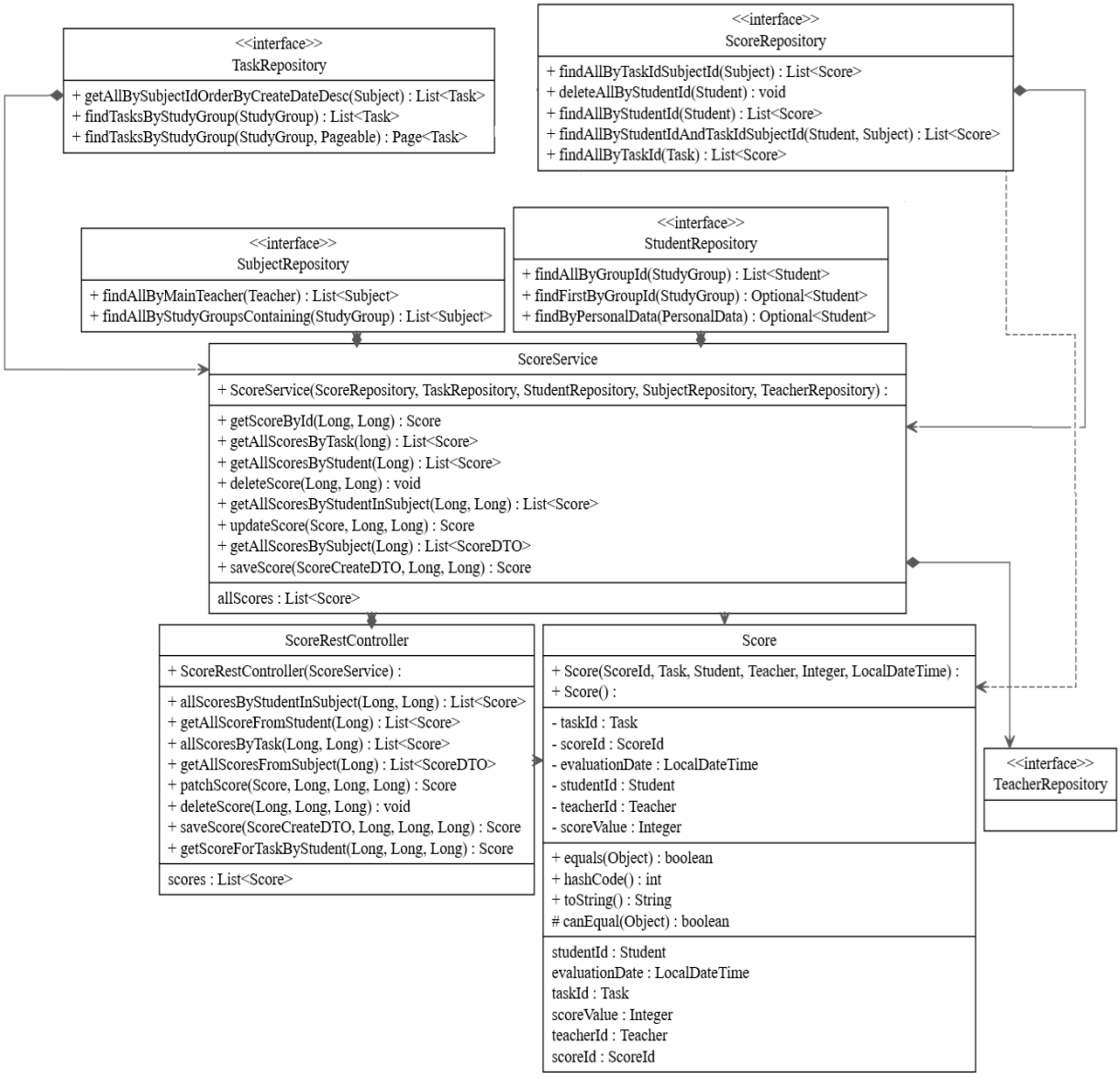

Рисунок В.10 – UML діаграма контролера оцінки, його об'єкта і використаних сервісів та репозиторіїв

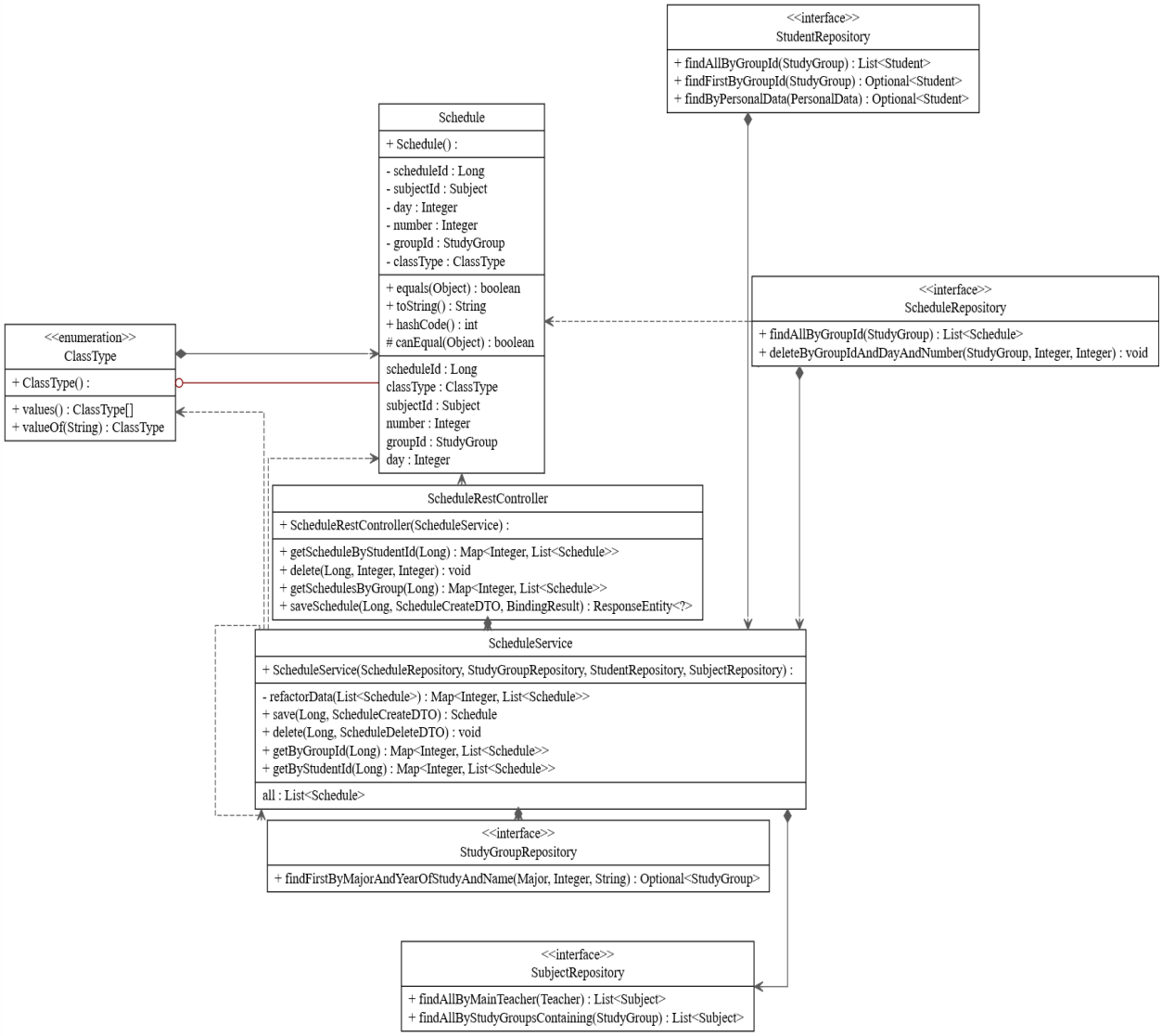

Рисунок В.11 – UML діаграма контролера розкладу, його об'єкта і використаних сервісів та репозиторіїв

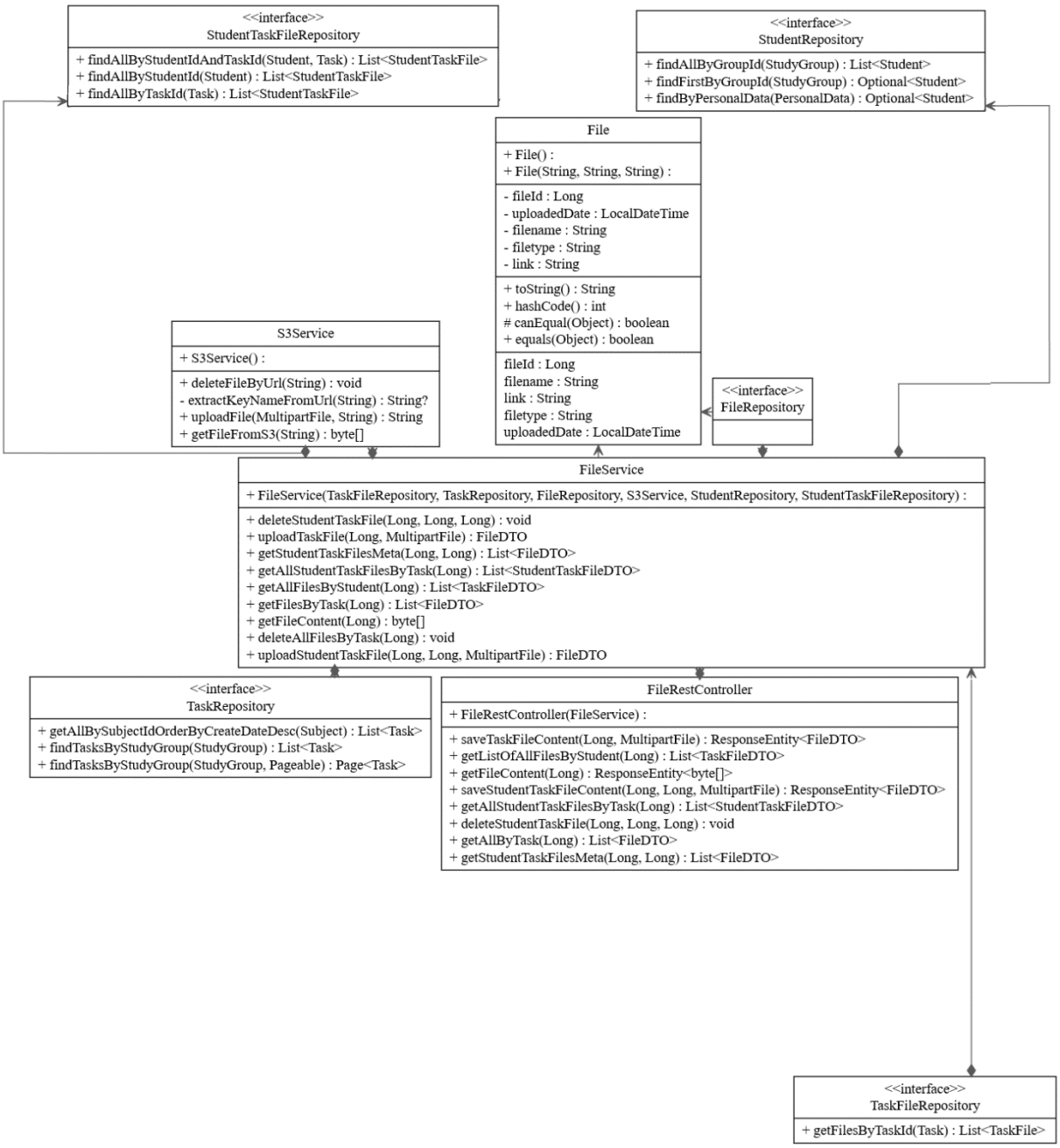

Рисунок В.12 – UML діаграма контролера файлів, його об'єкта і використаних сервісів та репозиторіїв

ДОДАТОК Г

Запити на створення таблиць бази даних

```
CREATE TABLE member
(
    member_id     SERIAL PRIMARY KEY,
    email         VARCHAR(255) UNIQUE NOT NULL,
    password_hash VARCHAR(255)        NOT NULL,
    role          role_type           NOT NULL DEFAULT
('STUDENT'),
    CONSTRAINT proper_email CHECK ( email ~* '^[A-Za-z0-9._%-
]+@[A-Za-z0-9.-]+[.][A-Za-z]+$' )
);

CREATE TABLE personal_data
(
    data_id    SERIAL PRIMARY KEY,
    member     INTEGER      NOT NULL UNIQUE,
    first_name VARCHAR(255) NOT NULL,
    last_name  VARCHAR(255) NOT NULL,
    patronymic VARCHAR(255),
    FOREIGN KEY (member) REFERENCES member (member_id) ON DELETE
CASCADE
);

CREATE TABLE faculty
(
    faculty_id SERIAL PRIMARY KEY,
    name       VARCHAR(255) NOT NULL UNIQUE
);

CREATE TABLE major
(
    major_id   INTEGER PRIMARY KEY,
    name       VARCHAR(255)                              NOT NULL,
    faculty_id INTEGER REFERENCES faculty (faculty_id) NOT NULL
);

CREATE TABLE study_group
(
    group_id      SERIAL PRIMARY KEY,
    name          VARCHAR(255)                          NOT NULL,
    major         INTEGER REFERENCES major (major_id) NOT NULL,
    year_of_study INTEGER                               NOT NULL
CHECK (year_of_study BETWEEN 1 AND 6)
);

CREATE TABLE teacher
(
```

Лістинг Г.1 – Запити на створення таблиць бази даних

```
    teacher_id INTEGER PRIMARY KEY REFERENCES personal_data
(data_id) ON DELETE CASCADE
);

CREATE TABLE student
(
    student_id INTEGER PRIMARY KEY REFERENCES personal_data
(data_id) ON DELETE CASCADE,
    group_id   INTEGER REFERENCES study_group (group_id) NOT
NULL
);

CREATE TABLE subject
(
    subject_id   SERIAL PRIMARY KEY,
    main_teacher INTEGER REFERENCES teacher (teacher_id),
    name         VARCHAR(255) NOT NULL
);

CREATE TABLE subject_study_group
(
    subject_id INTEGER REFERENCES subject (subject_id),
    group_id   INTEGER REFERENCES study_group (group_id),
    PRIMARY KEY (subject_id, group_id)
);

CREATE TABLE task
(
    task_id     SERIAL PRIMARY KEY,
    subject_id  INTEGER      NOT NULL REFERENCES subject
(subject_id) ON DELETE CASCADE,
    task        task_type    NOT NULL,
    title       VARCHAR(255) NOT NULL,
    description TEXT         NOT NULL,
    create_date TIMESTAMP    NOT NULL DEFAULT now()::timestamp
);

CREATE TABLE graded_task
(
    task_id   INTEGER PRIMARY KEY REFERENCES task (task_id) ON
DELETE CASCADE,
    max_score INTEGER   NOT NULL CHECK (max_score BETWEEN 0 AND
100),
    date_to   TIMESTAMP NOT NULL
);

CREATE TABLE score
(
    task_id         INTEGER REFERENCES task (task_id) ON DELETE
CASCADE,
```

Лістинг Г.1, лист 2

```
    student_id      INTEGER REFERENCES student (student_id) ON
DELETE CASCADE,
    teacher_id      INTEGER REFERENCES teacher (teacher_id) ON
DELETE CASCADE,
    score_value     INTEGER   NOT NULL CHECK (score_value
BETWEEN 0 AND 100),
    evaluation_date TIMESTAMP NOT NULL DEFAULT now()::timestamp,
    PRIMARY KEY (task_id, student_id)
);

CREATE TABLE schedule
(
    schedule_id SERIAL PRIMARY KEY,
    group_id    INTEGER    NOT NULL REFERENCES study_group
(group_id),
    subject_id  INTEGER    NOT NULL REFERENCES subject
(subject_id) ON DELETE CASCADE,
    day         INTEGER    NOT NULL CHECK (day BETWEEN 1 AND 6),
    number      INTEGER    NOT NULL CHECK (number BETWEEN 1 AND
6),
    class_type  class_type NOT NULL
);

CREATE TABLE file
(
    file_id       SERIAL PRIMARY KEY,
    link          TEXT         NOT NULL,
    filename      VARCHAR(255) NOT NULL,
    filetype      VARCHAR(255) NOT NULL,
    uploaded_date TIMESTAMP    NOT NULL DEFAULT now()::timestamp
);

CREATE TABLE task_file
(
    task_id INTEGER REFERENCES task (task_id) ON DELETE CASCADE,
    file_id INTEGER REFERENCES file (file_id) ON DELETE CASCADE,
    PRIMARY KEY (task_id, file_id)
);

CREATE TABLE student_task_file
(
    student_id INTEGER REFERENCES student (student_id),
    task_id    INTEGER REFERENCES task (task_id) ON DELETE
CASCADE,
    file_id    INTEGER REFERENCES file (file_id) ON DELETE
CASCADE,
    PRIMARY KEY (student_id, task_id, file_id)
);

CREATE TABLE refresh_token
```

Лістинг Г.1, лист 3

```
(
    token_id    SERIAL PRIMARY KEY,
    token       TEXT                NOT NULL,
    username    VARCHAR(255) UNIQUE NOT NULL,
    expiry_date TIMESTAMP           NOT NULL
);

CREATE TABLE chat
(
    chat_id    SERIAL PRIMARY KEY,
    task_id    INTEGER REFERENCES task (task_id) ON DELETE
CASCADE,
    student_id INTEGER REFERENCES student (student_id) ON DELETE
CASCADE,
    sender     INTEGER REFERENCES personal_data (data_id) ON
DELETE CASCADE,
    sent_date  TIMESTAMP NOT NULL DEFAULT now()::timestamp,
    message    TEXT      NOT NULL
);
```

Лістинг Г.1, лист 4

ДОДАТОК Д

REST запити клієнтського застосунку до сервера

```
export const retrieveAllStudents =
    () => apiClient.get('/students')

export const deleteStudentById =
    (id) => apiClient.delete(`/student/${id}`)

export const retrieveStudentById =
    (id) => apiClient.get(`/students/${id}`)

export const retrieveAllStudyGroups =
    () => apiClient.get('/study_groups')

export const retrieveStudyGroupByGroupId =
    (groupId) => apiClient.get(`/study_groups/${groupId}`)

export const patchStudent =
    (id, student) => apiClient.patch(`/student/${id}`, student)

export const saveStudent =
    (student) => apiClient.post(`/students`, student)

export const retrieveSchedule =
    (studentId) =>
apiClient.get(`/schedules/student/${studentId}`)

export const retrieveSubjects =
    (studentId) =>
apiClient.get(`/subjects/student/${studentId}`)

export const retrieveSubject =
    (subjectId) => apiClient.get(`/subjects/${subjectId}`)

export const retrieveTasksBySubject =
    (subjectId) => apiClient.get(`/subjects/${subjectId}/tasks`)

export const retrieveScoreByStudentInSubject =
    (subjectId, studentId) =>
apiClient.get(`/subjects/${subjectId}/students/${studentId}/scor
es`)

export const retrieveTask =
    (taskId) => apiClient.get(`/tasks/${taskId}`)

export const retrieveScoreByTask =
    (subjectId, taskId, studentId) =>
```

Лістинг Д.1 – Запити клієнтського застосунку до сервера

```
apiClient.get(`/subjects/${subjectId}/tasks/${taskId}/students/$
{studentId}`)

export const retrieveAllTasksByStudent =
    (studentId) => apiClient.get(`/tasks/students/${studentId}`)

export const retrieveAllScoresByStudent =
    (studentId) => apiClient.get(`/scores/${studentId}`)

export const retrieveAllTaskByStudentWithPag =
    (studentId, page, size, sort, direction) =>
apiClient.get(`/tasks/students/${studentId}/pag?page=${page}&siz
e=${size}&sort=${sort}&direction=${direction}`)

export const retrieveFilesMetaByTask =
    (taskId) => apiClient.get(`/tasks/${taskId}/files`)

export const retrieveFileContent =
    (fileId) => apiClient.get(`/files/${fileId}`, {
responseType: 'blob' })

export const uploadStudentTaskFileContent =
    (taskId, studentId, formData) =>
apiClient.post(`/tasks/${taskId}/students/${studentId}/files`,
formData, {
        headers: {
            'Content-Type': 'multipart/form-data'
        }
    })

export const uploadTaskFileContent =
    (taskId, formData) =>
apiClient.post(`/tasks/${taskId}/teach/files`, formData, {
        headers: {
            'Content-Type': 'multipart/form-data'
        }
    })

export const retrieveStudentTaskFileMeta =
    (studentId, taskId) =>
apiClient.get(`/tasks/${taskId}/students/${studentId}/files`)

export const deleteStudentTaskFile =
    (studentId, taskId, fileId) =>
apiClient.delete(`/tasks/${taskId}/students/${studentId}/files/$
{fileId}`)

export const retrieveAllFilesByStudent =
    (studentId) => apiClient.get(`/student/${studentId}/files`)
```

Лістинг Д.1, лист 2

```
export const retrieveAllSubjectsByTeacher =
    (teacherId) =>
apiClient.get(`/subjects/teacher/${teacherId}`)

export const saveInfoTask =
    (subjectId, task) =>
apiClient.post(`/subjects/${subjectId}/tasks/info`, task)

export const saveLabTask =
    (subjectId, task) =>
apiClient.post(`/subjects/${subjectId}/tasks/lab`, task)

export const retrieveStudentsByGroup =
    (groupId) =>
apiClient.get(`/study_groups/${groupId}/students`)

export const retrieveAllScoresInSubject =
    (subjectId) =>
apiClient.get(`/subjects/${subjectId}/scores`)

export const retrieveAllScoreByTask =
    (subjectId, taskId) =>
apiClient.get(`/subjects/${subjectId}/tasks/${taskId}/scores`)

export const retrieveAllStudentTaskFilesByTask =
    (taskId) => apiClient.get(`/tasks/${taskId}/students/files`)

export const deleteTask =
    (taskId) => apiClient.delete(`/tasks/${taskId}`)

export const logoutApi =
    (username) => apiClient.post(`/logout`, {
        username: username
    })

export const saveScore =
    (subjectId, taskId, studentId, score) =>
apiClient.post(`/subjects/${subjectId}/tasks/${taskId}/students/
${studentId}`, score)

export const retrieveRenewInfoBySubject =
    (subjectId) => apiClient.get(`/subjects/${subjectId}/renew`)

export const retrieveRenewInfoByTeacher =
    (teacherId) =>
apiClient.get(`/teachers/${teacherId}/renews`)

export const retrieveStudentInSubject =
    (studentId, subjectId) =>
apiClient.get(`/subjects/${subjectId}/student/${studentId}`)
```

Лістинг Д.1, лист 3

```
export const retrieveAllFaculties =
    () => apiClient.get('/faculties')

export const retrieveFacultyById =
    (facultyId) => apiClient.get(`/faculties/${facultyId}`)

export const saveFaculty =
    (faculty) => apiClient.post(`/faculties`, faculty)

export const updateFaculty =
    (facultyId, faculty) =>
apiClient.put(`/faculties/${facultyId}`, faculty)

export const deleteFaculty =
    (facultyId) => apiClient.delete(`/faculties/${facultyId}`)

export const retrieveMajors =
    () => apiClient.get(`/majors`)

export const retrieveMajorById =
    (majorId) => apiClient.get(`/majors/${majorId}`)

export const saveMajor =
    (major) => apiClient.post(`/majors`, major)

export const updateMajor =
    (majorId, major) => apiClient.put(`/majors/${majorId}`,
major)

export const deleteMajor =
    (majorId) => apiClient.delete(`/majors/${majorId}`)

export const saveStudyGroup =
    (studyGroup) => apiClient.post(`/study_groups`, studyGroup)

export const updateStudyGroup =
    (groupId, studyGroup) =>
apiClient.put(`/study_groups/${groupId}`, studyGroup)

export const deleteStudyGroup =
    (groupId) => apiClient.delete(`/study_groups/${groupId}`)

export const retrieveAllTeachers =
    () => apiClient.get(`/teachers`)

export const saveTeacher =
    (teacher) => apiClient.post(`/teachers`, teacher)

export const retrieveTeacher =
    (teacherId) => apiClient.get(`/teachers/${teacherId}`)
```

Лістинг Д.1, лист 4

```
export const updateTeacher =
    (teacherId, teacher) =>
apiClient.put(`/teachers/${teacherId}`, teacher)

export const deleteTeacher =
    (teacherId) => apiClient.delete(`/teachers/${teacherId}`)

export const updateStudent =
    (studentId, student) =>
apiClient.put(`/students/${studentId}`, student)

export const deleteStudent =
    (studentId) => apiClient.delete(`/students/${studentId}`)

export const retrieveAllSubjects =
    () => apiClient.get(`/subjects`)

export const saveSubject =
    (subject) => apiClient.post(`/subjects`, subject)

export const updateSubject =
    (subjectId, subject) =>
apiClient.put(`/subjects/${subjectId}`, subject)

export const deleteSubject =
    (subjectId) => apiClient.delete(`/subjects/${subjectId}`)

export const retrieveScheduleByStudyGroup =
    (studyGroupId) =>
apiClient.get(`/schedules/${studyGroupId}`)

export const retrieveSubjectsByStudyGroup =
    (studyGroupId) =>
apiClient.get(`/subjects/study_groups/${studyGroupId}`)

export const saveSchedule =
    (studyGroupId, schedule) =>
apiClient.post(`/schedules/${studyGroupId}`, schedule)

export const deleteSchedule =
    (studyGroupId, schedule) =>
apiClient.delete(`/schedules/${studyGroupId}?day=${schedule.day}
&number=${schedule.number}`)

export const retrieveMessages =
    (taskId, studentId) =>
apiClient.get(`/tasks/${taskId}/students/${studentId}/chat`)

export const saveMessage =
```

Лістинг Д.1, лист 5

```
    (taskId, studentId, message) =>
apiClient.post(`/tasks/${taskId}/students/${studentId}/chat`,
message)

export const changePassword =
    (memberId, pass) =>
apiClient.post(`/members/${memberId}/change_password`, pass);

export const executeJwtAuthenticationService = (username,
password) => apiClient.post("/login", {
    "username": username,
    "password": password
})
```

Лістинг Д.1, лист 6

ДОДАТОК Е

Приклад коду сторінки інтерфейса користувача

```
import {useNavigate, useParams} from "react-router-dom";
import {useEffect, useState} from "react";
import NoContentComponent from "../NoContentComponent";
import AdminManageComponent from "./AdminManageComponent";
import {useFormik} from "formik";
import {deleteMajor, retrieveAllFaculties, retrieveMajorById,
saveMajor, updateMajor} from "../../api/MemberApi";

export default function MajorComponent() {
    const {majorId} = useParams()
    const [major, setMajor] = useState(null)
    const [faculties, setFaculties] = useState([])
    const [isLoading, setIsLoading] = useState(true)
    const [isModalOpen, setIsModalOpen] = useState(false)
    const [isModalErrorOpen, setIsModalErrorOpen] =
useState(false)
    const [errorMessage, setErrorMessage] = useState('')
    const navigator = useNavigate()

    useEffect(() => {
        refreshMajor()
    }, []);

    function refreshMajor() {
        setIsLoading(true)
        if (majorId != null) {
            Promise.all([retrieveMajorById(majorId),
retrieveAllFaculties()])
                .then((responses) => {
                    setMajor(responses[0].data)
                    setFaculties(responses[1].data)
                })
                .catch((error) => console.log(error))
                .finally(() => setIsLoading(false))
        } else {
            Promise.all([retrieveAllFaculties()])
                .then((responses) =>
setFaculties(responses[0].data))
                .catch((error) => console.log(error))
                .finally(() => setIsLoading(false))
        }
    }

    const onDeleteHandle = (event) => {
        event.preventDefault()
```

Лістинг Е.1 – Код сторінки для створення або оновлення спеціальності

```
        setIsModalOpen(true)
    }

    const handleDeleteConfirm = () => {
        setIsLoading(true)
        setIsModalOpen(false)
        deleteMajor(majorId)
            .then(() => navigator('/admin/manage/majors'))
            .catch(() => setIsModalErrorOpen(true))
            .finally(() => setIsLoading(false))
    }

    const validate = values => {
        const errors = {};

        if (values.name.length === 0) {
            errors.name = 'Потрібна назва'
        }

        if (values.faculty === '') {
            errors.faculty = 'Потрібен факультет'
        }

        if (majorId == null) {
            if (!Number.isInteger(values.createdMajorId)) {
                errors.createdMajorId = 'Потрібен код
спеціальності'
            } else {
                if (values.createdMajorId < 0 ||
values.createdMajorId >= 1000) {
                    errors.createdMajorId = 'Потрібен валідний
код спеціальності'
                }
            }
        }
        return errors
    }

    const formik = useFormik({
        initialValues: {
            createdMajorId: '', name: '', faculty: ''
        }, validate, onSubmit: values => {
            if (majorId != null) {
                Promise.all([updateMajor(majorId, ({
                    majorId: majorId,
                    name: values.name,
                    facultyId: values.faculty
                }))])
                    .then(() =>
```

Лістинг Е.1, лист 2

```
 navigator('/admin/manage/majors'))
                    .catch(() => setErrorMessage('Помилка'))
            } else {
                Promise.all([saveMajor(({
                    majorId: values.createdMajorId,
                    name: values.name,
                    facultyId: values.faculty
                }))])
                    .then(() =>
navigator('/admin/manage/majors'))
                    .catch(() => setErrorMessage('Спеціальність
з таким кодом вже існує'))
            }

        }
    });

    if (isLoading) {
        return (<div></div>)
    }

    if (!isLoading && !major && majorId != null) {
        return (<NoContentComponent/>)
    }

    if (!isLoading && faculties.length <= 0) {
        return (<div>
            Необхідно спочатку створити факультет
        </div>)
    }

    return (<div className="flex">
        <AdminManageComponent/>
        <div className="w-[77vw] flex flex-col justify-center
items-center mt-10">
            <h1 className="text-3xl font-bold mb-
[10px]">{majorId ? "Зміна спеціальності" : "Створення
спеціальності"}</h1>
            {majorId && <button className="btn btn-danger"
onClick={onDeleteHandle}>
                Видалити спеціальність
            </button>}
            <form
                onSubmit={formik.handleSubmit}
                className="flex flex-col p-8flex flex-col p-8
bg-white shadow-md rounded"
            >
                {majorId == null && (
                    <div className="flex flex-col">
                        <label htmlFor="textField"
```

Лістинг Е.1, лист 3

```
className="font-bold text-gray-700 mb-2">
                            Код спеціальності
                        </label>
                        <input type="number" id="createdMajorId"
                               name="createdMajorId"
                               onChange={formik.handleChange}

value={formik.values.createdMajorId}
                               placeholder="Введіть код нової
спеціальності"
                               min={1}
                               max={1000}
                               className="px-4 py-2 border
rounded focus:outline-none focus:ring-2 focus:ring-blue-400"/>
                        {formik.errors.createdMajorId ? (
                            <div className="mt-2 text-sm text-
red-600">{formik.errors.createdMajorId}</div>
                        ) : null}
                    </div>
                )}
                <label htmlFor="textField" className="font-bold
text-gray-700 mb-2">
                    Назва спеціальності
                </label>
                <input type="text" id="name"
                       name="name"
                       onChange={formik.handleChange}
                       value={formik.values.name}
                       placeholder="Введіть назву"
                       className="px-4 py-2 border rounded
focus:outline-none focus:ring-2 focus:ring-blue-400"/>
                {formik.touched.name && formik.errors.name ? (
                    <div className="mt-2 text-sm text-red-
600">{formik.errors.name}</div>
                ) : null}
                <div className="mb-4">
                    <label
                        htmlFor="faculty"
                        className="font-bold text-gray-700 mb-2"
                    >
                        Факультет
                    </label>
                    <select
                        id="faculty"
                        name="faculty"
                        onChange={formik.handleChange}
                        onBlur={formik.handleBlur}
                        value={formik.values.faculty}
                        className="w-full p-3 border rounded-lg
shadow-sm focus:outline-none focus:ring-2 focus:ring-blue-500">
```

Лістинг Е.1, лист 4

```
                        <option value="" label="Оберіть
факультет"/>
                        {faculties.map(faculty => (<option
                            key={faculty.facultyId}
                            value={faculty.facultyId}
                        >
                            {faculty.name}
                        </option>))}
                    </select>
                    {formik.touched.faculty &&
formik.errors.faculty ? (
                        <div className="mt-2 text-sm text-red-
600">{formik.errors.faculty}</div>
                    ) : null}
                </div>
                {errorMessage && (
                    <div className="text-sm text-red-
600">{errorMessage}</div>
                )}
                <button
                    type="submit"
                    className="mt-2 bg-blue-500 text-white font-
bold py-2 px-4 rounded hover:bg-blue-700">
                    {majorId ? 'Оновити' : 'Створити'}
                </button>
            </form>
            <DeleteConfirmationModal
                isOpen={isModalOpen}
                onClose={() => setIsModalOpen(false)}
                onConfirm={handleDeleteConfirm}
            />
            <DeleteErrorModal
                isOpen={isModalErrorOpen}
                onClose={() => setIsModalErrorOpen(false)}
                onConfirm={handleDeleteConfirm}
            />
        </div>
    </div>)
}

function DeleteConfirmationModal({isOpen, onClose, onConfirm}) {
    return (<>
        {isOpen && (<div
            className="fixed top-0 left-0 w-full h-full bg-gray-
800 bg-opacity-50 flex items-center justify-center z-50">
            <div className="bg-white p-6 rounded-lg shadow-lg">
                <h2 className="text-lg font-semibold mb-
4">Підтвердження видалення</h2>
                <p className="text-sm mb-4">Ви впевнені, що
хочете видалити цю спеціальність?</p>
```

Лістинг Е.1, лист 5

```
                <div className="flex justify-end">
                    <button
                        className="px-4 py-2 mr-2 bg-red-500
text-white rounded hover:bg-red-600"
                        onClick={onConfirm}>
                        Видалити
                    </button>
                    <button
                        className="px-4 py-2 bg-gray-300 text-
gray-800 rounded hover:bg-gray-400"
                        onClick={onClose}>
                        Відміна
                    </button>
                </div>
            </div>
        </div>)}
    </>);
}

function DeleteErrorModal({isOpen, onClose}) {
    return (<>
        {isOpen && (<div
            className="fixed top-0 left-0 w-full h-full bg-gray-
800 bg-opacity-50 flex items-center justify-center z-50">
            <div className="bg-white p-6 rounded-lg shadow-lg">
                <h2 className="text-lg font-semibold mb-
4">Сталися помилка</h2>
                <p className="text-sm mb-4">Можливо деякі
студенти ще пов'язані з цією спеціальністю. Спочатку
                    видалити їх.</p>
                <div className="flex justify-end">
                    <button
                        className="px-4 py-2 bg-gray-300 text-
gray-800 rounded hover:bg-gray-400"
                        onClick={onClose}>
                        Відміна
                    </button>
                </div>
            </div>
        </div>)}
    </>);
}
```

Лістинг Е.1, лист 6

ДОДАТОК Ж

Реалізація безпеки на рівні сервера

```
@Component
public class AuthenticationManagerConfig {
    private final DaoAuthenticationProvider
daoAuthenticationProvider;

    public AuthenticationManagerConfig(DaoAuthenticationProvider
daoAuthenticationProvider) {
        this.daoAuthenticationProvider =
daoAuthenticationProvider;
    }

    @Bean
    public AuthenticationManager authManager(HttpSecurity http)
throws Exception {
        AuthenticationManagerBuilder
authenticationManagerBuilder =

http.getSharedObject(AuthenticationManagerBuilder.class);
        return authenticationManagerBuilder

.authenticationProvider(daoAuthenticationProvider)
                .build();
    }
}
```

Лістинг Ж.1 – Клас AuthenticationManagerConfig

```
@Service
public class AuthenticationService {
    private final AuthenticationManager authenticationManager;
    private final MemberService memberService;
    private final JwtService jwtService;

    public AuthenticationService(AuthenticationManager
authenticationManager, MemberService memberService, JwtService
jwtService) {
        this.authenticationManager = authenticationManager;
        this.memberService = memberService;
        this.jwtService = jwtService;
    }

    public JwtAuthenticationResponse authenticate(LoginDTO
loginDTO) {
```

Лістинг Ж.2 – Клас AuthenticationService

```
        authenticationManager.authenticate(new
UsernamePasswordAuthenticationToken(
                loginDTO.getUsername(),
                loginDTO.getPassword()));

        UserDetails user =
memberService.userDetailsService().loadUserByUsername(loginDTO.g
etUsername());

        return new
JwtAuthenticationResponse(jwtService.generateToken(user),
jwtService.getRefreshToken(user.getUsername()).getToken());
    }

    public Map<String, String> refreshToken(String
refreshTokenStr) {
        return jwtService.findByToken(refreshTokenStr)
                .map(jwtService::verifyExpiration)
                .map(RefreshToken::getUsername)
                .map(username -> {
                    String newAccessToken =
jwtService.generateToken(memberService.userDetailsService().load
UserByUsername(username));
                    Map<String, String> tokens = new
HashMap<>();
                    tokens.put("accessToken", newAccessToken);
                    tokens.put("refreshToken",
jwtService.getRefreshToken(username).getToken());
                    return tokens;
                })
                .orElseGet(() -> null);
    }

    @Transactional
    public void logout(String username) {
        jwtService.deleteByUsername(username);
    }
}
```

Лістинг Ж.2, лист 2

```
@Component
public class CustomDaoAuthenticationProvider {

    private final MemberService memberService;
    private final PasswordEncoder passwordEncoder;

    public CustomDaoAuthenticationProvider(MemberService
```

Лістинг Ж.3 – Клас CustomDaoAuthenticationProvider

```
memberService, PasswordEncoder passwordEncoder) {
        this.memberService = memberService;
        this.passwordEncoder = passwordEncoder;
    }

    @Bean
    public DaoAuthenticationProvider authenticationProvider() {
        DaoAuthenticationProvider provider = new
DaoAuthenticationProvider();

provider.setUserDetailsService(memberService.userDetailsService(
));
        provider.setPasswordEncoder(passwordEncoder);
        return provider;
    }
}
```

Лістинг Ж.3, лист 3

```
@Component
public class JwtAuthenticationFilter extends
OncePerRequestFilter {
    public static final String BEARER_PREFIX = "Bearer ";
    public static final String HEADER_NAME = "Authorization";
    private final JwtService jwtService;
    private final MemberService memberService;

    public JwtAuthenticationFilter(JwtService jwtService,
MemberService memberService) {
        this.jwtService = jwtService;
        this.memberService = memberService;
    }

    @Override
    protected void doFilterInternal(
            @NonNull HttpServletRequest request,
            @NonNull HttpServletResponse response,
            @NonNull FilterChain filterChain
    ) throws ServletException, IOException {

        var authHeader = request.getHeader(HEADER_NAME);
        if (!StringUtils.hasLength(authHeader) ||
!StringUtils.startsWithIgnoreCase(authHeader, BEARER_PREFIX)) {
            filterChain.doFilter(request, response);
            return;
        }

        var jwt = authHeader.substring(BEARER_PREFIX.length());
```

Лістинг Ж.4 – Клас JwtAuthenticationFilter

```
        var username = jwtService.extractUserName(jwt);

        if (StringUtils.hasLength(username) &&
SecurityContextHolder.getContext().getAuthentication() == null)
{
            UserDetails userDetails = memberService
                    .userDetailsService()
                    .loadUserByUsername(username);

            if (jwtService.isTokenValid(jwt, userDetails)) {
                SecurityContext context =
SecurityContextHolder.createEmptyContext();

                UsernamePasswordAuthenticationToken authToken =
new UsernamePasswordAuthenticationToken(
                        userDetails,
                        null,
                        userDetails.getAuthorities()
                );

                authToken.setDetails(new
WebAuthenticationDetailsSource().buildDetails(request));
                context.setAuthentication(authToken);
                SecurityContextHolder.setContext(context);
            }
        }
        filterChain.doFilter(request, response);
    }

    @Override
    protected boolean shouldNotFilter(HttpServletRequest
request) throws ServletException {
        String path = request.getServletPath();
        return path.startsWith("/v1/refresh");
    }
}
```

Лістинг Ж.4, лист 2

```
@Service
public class JwtService {
    private static final long JWT_EXPIRATION = 600000;
    private final RefreshTokenRepository refreshTokenRepository;

    public JwtService(RefreshTokenRepository
refreshTokenRepository) {
        this.refreshTokenRepository = refreshTokenRepository;
    }
```

Лістинг Ж.5 – Клас JwtService

```
    @Value("${jwt.signing.key}")
    private String jwtSigningKey;

    public String extractUserName(String token) {
        return extractClaim(token, Claims::getSubject);
    }

    public String generateToken(UserDetails userDetails) {
        Map<String, Object> claims = new HashMap<>();
        if (userDetails instanceof Member customUserDetails) {
            claims.put("id", customUserDetails.getMemberId());
            claims.put("email", customUserDetails.getEmail());
            claims.put("role", customUserDetails.getRole());
        }
        return generateToken(claims, userDetails);
    }

    private String generateToken(Map<String, Object>
extraClaims, UserDetails userDetails) {
        return
Jwts.builder().setClaims(extraClaims).setSubject(userDetails.get
Username())
                .setIssuedAt(new
Date(System.currentTimeMillis()))
                .setExpiration(new
Date(System.currentTimeMillis() + JWT_EXPIRATION))
                .signWith(getSigningKey(),
SignatureAlgorithm.HS256).compact();
    }

    private String generateRefreshToken(String username,
LocalDateTime expiration) {
        return Jwts.builder()
                .setSubject(username)
                .setIssuedAt(new Date())
                .setExpiration(java.util.Date

.from(expiration.atZone(ZoneId.systemDefault())
                                .toInstant()))
                .signWith(getSigningKey(),
SignatureAlgorithm.HS256).compact();
    }

    private Claims extractAllClaims(String token) {
        return
Jwts.parser().setSigningKey(getSigningKey()).build().parseClaims
Jws(token)
                .getBody();
    }
```

Лістинг Ж.5, лист 2

```
    private <T> T extractClaim(String token, Function<Claims, T>
claimsResolvers) {
        final Claims claims = extractAllClaims(token);
        return claimsResolvers.apply(claims);
    }

    private Key getSigningKey() {
        byte[] keyBytes = Decoders.BASE64.decode(jwtSigningKey);
        return Keys.hmacShaKeyFor(keyBytes);
    }

    private Date extractExpiration(String token) {
        return extractClaim(token, Claims::getExpiration);
    }

    public boolean isTokenValid(String token, UserDetails
userDetails) {
        final String userName = extractUserName(token);
        return (userName.equals(userDetails.getUsername())) &&
!isTokenExpired(token);
    }

    private boolean isTokenExpired(String token) {
        return extractExpiration(token).before(new Date());
    }

    public Optional<RefreshToken> findByToken(String token) {
        return refreshTokenRepository.findByToken(token);
    }

    public void deleteByUsername(String username) {
        refreshTokenRepository.deleteByUsername(username);
    }

    public RefreshToken verifyExpiration(RefreshToken token) {
        if (token.getExpiryDate().isBefore(LocalDateTime.now()))
{
            refreshTokenRepository.delete(token);
            throw new RuntimeException("Refresh token has
expired");
        }
        return token;
    }

    public RefreshToken getRefreshToken(String username) {
        RefreshToken refreshToken = new RefreshToken();
        refreshToken.setUsername(username);
        LocalDateTime expDate =
LocalDateTime.now().plusHours(6);
```

Лістинг Ж.5, лист 3

```
        refreshToken.setExpiryDate(expDate);
        refreshToken.setToken(generateRefreshToken(username,
expDate));
        Optional<RefreshToken> oldToken =
refreshTokenRepository.findByUsername(username);
        oldToken.ifPresent(token ->
refreshToken.setTokenId(token.getTokenId()));
        return refreshTokenRepository.save(refreshToken);
    }
}
```

Лістинг Ж.5, лист 4

```
@RestController
@RequestMapping(path = "/v1", produces = "application/json")
public class LogInRestController {
    private final AuthenticationService authenticationService;

    public LogInRestController(AuthenticationService
authenticationService) {
        this.authenticationService = authenticationService;
    }

    @PostMapping("/login")
    public JwtAuthenticationResponse signIn(@RequestBody
LoginDTO loginDTO) {
        return authenticationService.authenticate(loginDTO);
    }

    @PostMapping("/refresh")
    public ResponseEntity<?> refreshToken(@RequestBody
Map<String, String> request) {
        String refreshTokenStr = request.get("refreshToken");
        Map<String, String> newTokens =
authenticationService.refreshToken(refreshTokenStr);
        return newTokens == null ? new
ResponseEntity<>(HttpStatus.FORBIDDEN) : new
ResponseEntity<>(newTokens, HttpStatus.OK);
    }

    @PostMapping("/logout")
    public ResponseEntity<?> logoutUser(@RequestBody Map<String,
String> request) {
        String username = request.get("username");
        authenticationService.logout(username);
        return ResponseEntity.ok("User logged out
successfully.");
    }
}
```

Лістинг Ж.6 – Клас LogInRestController

```
@Configuration
public class PassEncoderConf {
    @Bean
    public PasswordEncoder passwordEncoder() {
        return new BCryptPasswordEncoder();
    }
}
```

Лістинг Ж.7 – Клас PassEncoderConf

```
@Configuration
public class SecurityConfig {
    private final JwtAuthenticationFilter
jwtAuthenticationFilter;

    public SecurityConfig(JwtAuthenticationFilter
jwtAuthenticationFilter) {
        this.jwtAuthenticationFilter = jwtAuthenticationFilter;
    }

    @Bean
    public SecurityFilterChain filterChain(HttpSecurity http)
throws Exception {
        http.csrf(AbstractHttpConfigurer::disable);
        http.cors(cors -> cors.configurationSource(request -> {
            var corsConfiguration = new CorsConfiguration();

corsConfiguration.setAllowedOriginPatterns(List.of("*"));
            corsConfiguration.setAllowedMethods(List.of("GET",
"POST", "PUT", "DELETE", "OPTIONS"));
            corsConfiguration.setAllowedHeaders(List.of("*"));
            corsConfiguration.setAllowCredentials(true);
            return corsConfiguration;
        }));
        http.sessionManagement(manager ->
manager.sessionCreationPolicy(SessionCreationPolicy.STATELESS));
        http.addFilterBefore(jwtAuthenticationFilter,
UsernamePasswordAuthenticationFilter.class);
        http.authorizeHttpRequests(request ->
                request
                        .requestMatchers("/v1/login",
"/v1/refresh").permitAll()
                        .requestMatchers(HttpMethod.POST,
"/v1/majors", "/v1/members", "/v1/faculties",
                                "/v1/schedules/",
"/v1/schedules/{studentGroupId}", "/v1/students",
"/v1/study_groups",
                                "/v1/subjects", "/v1/teachers")
                        .hasRole("ADMIN")
```

Лістинг Ж.7 – Клас SecurityConfig

```
                        .requestMatchers(HttpMethod.PUT,
"/v1/faculties/{facultyId}", "/v1/majors/{majorsId}",
                                "/v1/students/{studentId}",
"/v1/study_groups/{groupId}", "/v1/subjects/{subjectId}",
                                "/v1/teachers/{teachersId}")
                        .hasRole("ADMIN")
                        .requestMatchers(HttpMethod.GET,
"/v1/faculties", "/v1/faculties/{facultyId}", "/v1/majors",
                                "/v1/majors/{majorsId}",
"/v1/schedules/{groupId}", "/v1/students",
                                "/v1/students/{studentId}",
"/v1/study_groups", "/v1/study_groups/{id}",
                                "/v1/subjects",
"/v1/subjects/study_groups/{studyGroupId}", "/v1/teachers",

"/v1/teachers/{teacherId}").hasRole("ADMIN")
                        .requestMatchers(HttpMethod.DELETE,
"/v1/faculties/{facultyId}", "/v1/majors/{majorsId}",
                                "/v1/schedules/{studyGroupId}",
"/v1/students/{studentId}", "/v1/study_groups/{id}",
                                "/v1/subjects/{id}",
"/v1/teachers/{id}").hasRole("ADMIN")
                        .requestMatchers(HttpMethod.POST,
"/v1/subjects/{subjectId}/tasks/lab",
"/v1/subjects/{subjectId}/tasks/info",

"/v1/subjects/{subjectId}/tasks/{taskId}/students/{studentId}",
"/v1/tasks/{taskId}/teach/files")
                        .hasRole("TEACHER")
                        .requestMatchers(HttpMethod.GET,
"/v1/subjects/teacher/{teacherId}",
"/v1/teachers/{teacherId}/renews")
                        .hasRole("TEACHER")
                        .requestMatchers(HttpMethod.DELETE,
"/v1/tasks/{taskId}").hasRole("TEACHER")
                        .requestMatchers(HttpMethod.GET,
"/v1/study_groups/{id}/students",
"/v1/subjects/{subjectId}/renew",

"/v1/subjects/{subjectId}/scores",
"/v1/subjects/{subjectId}/student/{studentId}",

"/v1/subjects/{subjectId}/tasks/{taskId}/scores",
"/v1/tasks/{taskId}/students/files")
                        .hasAnyRole("ADMIN", "TEACHER")
                        .requestMatchers(HttpMethod.GET,
"/v1/subjects/student/{studentId}",
"/v1/subjects/{subjectId}/students/{studentId}/scores",

"/v1/tasks/students/{studentId}/pag")
```

Лістинг Ж.7, лист 2

```
                        .hasRole("STUDENT")
                        .requestMatchers(HttpMethod.POST,
"/v1/tasks/{taskId}/students/{studentId}/chat").hasAnyRole("TEAC
HER", "STUDENT")
                        .requestMatchers(HttpMethod.POST,
"/v1/tasks/{taskId}/students/{studentId}/files").hasRole("STUDEN
T")
                        .requestMatchers(HttpMethod.DELETE,
"/v1/tasks/{taskId}/students/{studentId}/files/{filesId}").hasRo
le("STUDENT")
                        .anyRequest().authenticated());
        return http.build();
    }
}
```

Лістинг Ж.7, лист 3