Міністерство освіти і науки України
Рівненський державний гуманітарний університет
Кафедра інформаційних технологій та моделювання

**Кваліфікаційна робота**

за освітнім ступенем «бакалавр»

на тему:

# «Е-ЖУРНАЛ АКАДЕМІЧНОЇ ГРУПИ В ОСВІТНЬОМУ ПРОЦЕСІ ЗАКЛАДУ ВИЩОЇ ОСВІТИ»

**Виконав:**
здобувач IV курсу
групи КН-41
спеціальності 122 «Комп’ютерні науки»
Лазарчук Максим Петрович

**Науковий керівник**:
к.п.н., доцент Петренко С. В.

Рівне – 2025

# ЗМІСТ

# ВСТУП

**Актуальність дослідження.** Сучасний розвиток інформаційних технологій дає змогу автоматизувати багато процесів, що попередньо потребували чималих людських зусиль старань і витрат часу. Актуальним вектором є впровадження електронних систем в освітню галузь, зокрема для курування навчальним процесом, відстеження успішності студентів, зберігання інформації про оцінки, відвідуваність та інші показники навчальної діяльності. Автоматизація цих процесів здатна суттєво полегшити роботу адміністрації, викладачів і студентів, зробивши доступ до інформації простішим, швидшим та зручнішим.

У рамках даного дослідження розглядається розробка електронного журналу, який дозволяє забезпечити облік академічних показників студентів на рівні університету. Така система дозволяє адміністратору додавати користувачів та університети, керівникам факультетів контролювати діяльність студентів і викладачів, викладачам – виставляти оцінки, фіксувати відвідуваність, а студентам – переглядати свої результати та завдання.

**Мета дослідження:** розробці та впровадженні веб-додатку для автоматизації управління освітнім процесом у вищих навчальних закладах. Дослідження спрямоване на створення інтерактивної системи, що забезпечує управління користувачами, групами, дисциплінами, розкладом, завданнями, оцінками та звітами, задля підвищення ефективності адміністрування та навчання у сфері освіти.

**Завдання дослідження:**

- Дослідити сучасні підходи до розробки веб-додатків для управління освітнім процесом, включаючи аналіз аналогічних систем і технологій
- визгначити вимоги до функціоналу електроного журналу
- Спроєктувати архітектуру фронтенду та бекенду системи, включаючи REST API, базу даних і клієнт-серверну взаємодію, для забезпечення модульності та масштабованості.
- Розробити веб-додаток для автоматизації управління освітнім процесом у вищих навчальних закладах, включаючи модулі управління користувачами,

групами, дисциплінами, розкладом, завданнями, оцінками, звітами, для підвищення ефективності адміністрування та навчання.

**Об'єкт дослідження:** Процес автоматизації управління навчальною інформацією в закладах вищої освіти.

**Предмет дослідження:** Розробка веб-додатка для ведення електронного журналу з підтримкою різних рівнів доступу користувачів

**Методи дослідження:** аналіз літературних та інформаційних джерел, системний підхід, аналіз існуючих рішень та метод моделювання.

**Структура роботи:** складається зі вступу, трьох розділів, висновків та списку використаних джерел. Перший розділ включає аналіз предметної області, другий проєктування системи, зокрема вибір методів та технологій, архітектурне проєктування та моделювання бази даних, третій розділ присвячений реалізації системи.

# РОЗДІЛ 1
# АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

## 1.1 Огляд існуючих систем для управління навчальним процесом

Сучасні заклади освіти активно впроваджують цифрові рішення для оптимізації навчального процесу та автоматизації обліку успішності студентів. В умовах зростаючих вимог до управління великим обсягом даних про студентів, їхню успішність, відвідуваність та інші аспекти навчання виникає потреба у використанні систем управління навчанням (LMS). LMS дозволяють полегшити процес організації та контролю навчання, забезпечуючи прозорість і доступність даних як для студентів, так і для викладачів.

Цей розділ присвячено огляду існуючих LMS, таких як Moodle, Google Classroom та Blackboard. Метою огляду є виявлення сильних та слабких сторін цих платформ, і також визначення функцій, які потребують поліпшення для створення дієвої та універсальної системи курування навчанням.

**Moodle** (https://moodle.org/) – це одна з найпопулярніших open-source LMS, широко використовувана у навчальних закладах по всьому світу. Вона пропонує великий набір інструментів для організації навчання: інтерактивні курси, тестування, збирання завдань, форуми для обговорення. Moodle є безкоштовною і має відкритий код, що дозволяє адаптувати її до специфічних потреб закладу. Однак, для повного налаштування системи може знадобитися високий рівень технічних знань, а також ресурси для адміністрування та підтримки.

**Google Classroom**(https://classroom.google.com/) – це безкоштовна веб-платформа, інтегрована з інструментами Google, такими як Google Docs, Sheets, та Drive. Вона орієнтована на спрощення управління завданнями та полегшення комунікації між студентами і викладачами. Платформа дозволяє створювати та керувати курсами, перевіряти завдання та обмінюватися повідомленнями. Google Classroom є легкою у використанні і підходить для шкіл та університетів, що прагнуть мінімальних витрат на впровадження LMS. Проте, функціональні можливості цієї платформи обмежені порівняно з більш складними системами,

такими як Moodle. Classroom не дозволяє адаптуватися до складних навчальних процесів, де потрібно враховувати детальний облік успішності та відвідування.

**Нові знання** (https://nz.ua/) - безкоштовні електронні класні журнали та щоденники з можливостями проведення дистанційного навчання. Надають можливості зручного створення уроків, виставлення оцінок та аналізу успішності учнів, класів, школи. Учням та їхнім батькам постійний доступ до всієї історії отриманих оцінок та домашніх завдань. Можливість аналізу успішності за тривалим періодом. Гнучкі механізми аналізу та графічного відображення навчальних досягнень учня, класу, школи, роботи викладача.Можливості працювати дистанційно, задавати, виконувати та перевіряти роботи онлайн.

**Atoms** https://atoms.ua/ - сервіс який може повністю замінити паперовий журнал.Платформа працює таким чином, щоб надати всі можливості паперових журналів + всі переваги цифрового ведення журналу. Також журнали мають повну синхронізацію з електронними щоденниками учнів.Призначений для управління саме для шкіл.

**Єдина школа** https://eschool-ua.com - платний сервіс для шкіл,який рекомендовано Міністерством освіти і науки України, має такі функції: Електроне ведення шкільних журналів,проведення уроків у дистанційному режимі,автоматичне складання звітів,архівне зберігання документації, створювати домашнє завдання та прикріпляти додані файли до нього, та багато іншого.

**Висновок**. Вищезазначені сервісти є ефективними засобами для системи управління навчанням, що надають електроні журнали,щоденники, та інші дистанційні інструменти, проте кожна з них має свої обмеження, плюси та мінуси, які можуть не підійти для певних освітніх завдань.Наприклад **Moodle** чудовий інструмент, проте в нього складних функціонал, який потребує складного налаштування,а серіс **Google Classroom** - не зможе реалізувати такі функції як створення розкладу та завантаження звітів, а інші аналізовані інструменти мають мінуси - що вони або платні, або назначені саме для управління школами.Отже розробка власної системи управління навчальним процесом дозволить врахувати

специфічні потреби закладу освіти включаючи розширену аналітику,облік відвідуваності, керування групами,та інше.

### 1.2 Основні вимоги та модель доступу електронного журналу

#### 1.2.1 Основні вимоги до електроного журналу

Визначемо функціональні та технічні вимоги для розробки електроного журналу

**Функціональні вимоги:**

- управління користувачами. підтримка різних ролей таких як (студент, викладач, адміністратор, супер адміністратор)
- ведення електроного розкладу: створення, перегляд, редагування, видалення розкладу занять
- оцінювання. Надання можливості викладачам виставляти оцінки, студентам переглядати їх
- домашні завдання: для викладачів - створення, редагування, оцінювання, для студентів їх відправка
- Звіти та аналітика: автоматичне формулювання звітів з різними параметрами для різних ролей; онлайн перегляд та можливість завантажити у форматі pdf

**Технічні вимоги:**

- безпека даних. Забезпечення захисту персональних даних та безпеки інформації
- доступність. Забезпечення зручного доступу для платформи з різних пристроїв
- простий та інтуїтивний інтерфейс(UI/UX). Зручний інтерфейс для легкої взаємодії користувачів із системою

**Надійність:**

- програма виконує ту функцію, яку очікував користувач.
- користувач може допускати помилки або використовувати програмне забезпечення несподіваним чином.
- система запобігає будь-якому неавторизованому доступу.

Якщо всі ці речі разом означають «працювати правильно», тоді ми можемо розуміти надійність означає, приблизно, «продовжувати працювати правильно, навіть коли все йде неправильно» [1].

### 1.2.2 Рольова модель доступу

Даний застосунок буде використовувати контроль доступу на основі ролей (RBAC - role-based access control), тобто для різних користувачів - буде наданий відповідний інтерфейс та відповідна функціональність [2].

Планується організувати 4 ролі доступу:

1. Студент(Student)
2. Викладач(Teacher)
3. Адміністратор(Manager)
4. Супер Адміністратор(SuperAdmin)

Розглянемо детальніше кожну роль:

**Студент**. користувачам з цією ролю буде надано можливість переглядати розклад занять, завдання з кожної дисципліни, свою відвідуваність та оцінки, зможе надсилати виконані завдання на перевірку викладача, а також формувати звіти за заданими параметрами

**Викладач**. Для викладача також будуть надані дозволи на перегляд вже його занять, буде надано доступ на виставлення відвідування та оцінок для студентів, змога створювати завдання, та переглядати і оцінювати роботи студентів

**Адміністратор**. Буде мати можливість керувати всіма функціями у факультеті, а саме - створення розкладів для викладачів, створення дисциплін та привязування до них відповідних викладачів, підтвердження реєстрації студентів та викладачів до групи та факультету відповідно, та формування звітів по групам, студентам

**Супер Адміністратор**. Роль призначена для розробників, щоб повністю контролювати систему та мати змогу робити будь-які можливі зміни, а також головне - це створювати університети та факультети і підтверджувати реєстрацію адміністраторів на конкретний факультет.

При розробці системи важливо застосувати обмеження на основі ролей, щоб гарантувати, що ролі нижчого рівня не можуть виконувати функції, зарезервовані для ролей вищого рівня. Наприклад, учитель не повинен мати дозволу на зміну розкладу, а учень не повинен мати доступу до оцінок. Ці обмеження мають суворо дотримуватись як через інтерфейс користувача, так і шляхом перевірки запитів на сервер. Це забезпечує надійну безпеку та цілісність електронного журналу, запобігаючи неавторизованим діям незалежно від використовуваного методу доступу.

# РОЗДІЛ 2
# ПРОЄКТУВАННЯ СИСТЕМИ

## 2.1 Технічний стек

Для розробки інформаційної системи використовується сучасний фронтед(front-end) та бекенд(back-end) стек (набір технологій)

**Фронтенд**:

– Бібліотека/Фреймворк розробки користувацьктх інтерфейсів

*Reactjs* - біблітека яка забезпечує ефективну розробку інтерфейсу на основі компонентів, що ідеально підходить для складних додатків, тоді як TypeScript запезпечить безпеку типів, зменшуючи кількість помилок від час написання коду та підвищує зручність обслуговуваня коду [3].

*Vuejs* - фреймворк, зручний для побудови як малих так і великих додатків, має власно екосистему з бібліотекою vuex для менеджменту стану і vue router для маршррутизації. Проте він менш поширений серед додатків у порівнянні з React

Angular - потужний фреймворк від Google, який має всі інструменти для розробки великих додатків з багатофункціональним інтерфейсом. Однак, він більш складний у вивчені.

*Обрано: Reactjs + Typescript*, оскільки він дозволяє швидко створювати гнучкі, масштабрвані інтерфейси, підтримка Typescript покращує зручність підтримки й скорочує кількість помилок. React також дозволяє розробляти рішення в будь-який спосіб, а не як Vuejs чи Angular які мають правила для виконання, оскільки Reactjs - це бібліотека, а Vuejs та Angular фреймворки.

– Менеджер Стану

*Redux* – популярний інструмент для керування станом додатку, який дозволяє централізувати всі дані й забезпечує зручність відстеження змін. Redux Toolkit (RTK) спрощує роботу з Redux завдяки скороченню шаблонного коду та вбудованій підтримці асинхронних запитів.

*MobX* – альтернатива Redux, яка пропонує реактивний підхід до управління станом, що може бути простішим для невеликих додатків. Проте MobX менш популярний у великих проєктах і не має такої підтримки.

*Zustand* – легкий, швидкий менеджер стану, який не потребує великої кількості конфігурацій, але менш придатний для великих додатків із складним станом.

*Обрано: Redux Toolkit (RTK),* оскільки він забезпечує структуроване управління станом у великих проєктах, а також *RTK Query* для роботи з API й кешуванням.

– Бібліотеки для роботи з формами:

*React Hook Form* – бібліотека, оптимізована для управління станом форм у React додатках. Вона зменшує ререндери, підтримує валідацію та дозволяє легко інтегрувати додаткові інструменти, такі як *Zod*.

*Formik* – інша популярна бібліотека, яка пропонує функціонал для роботи з формами. Вона добре підходить для великих додатків, але має менше функцій у порівнянні з React Hook Form.

*Обрано: React Hook Form* через його ефективність і *Zod* для зручної валідації.

– Бібліотека для стилізації

Оскільки немає строгого визначеного дизайну для застосунку, використаємо бібліотеку для написанна стилів, що пришвидшить розробку користувацького інтерфейсу

*Material UI (MUI)* – бібліотека з великою кількістю готових компонентів, що дозволяє швидко створювати адаптивні інтерфейси. Вона має інтеграцію з темами та може адаптуватися до будь-якого дизайну.

*Ant Design* – інша бібліотека, яка також пропонує готові компоненти. Популярна у корпоративному секторі, але менш розповсюджена у поєднанні з React.

*Bootstrap* – загальновідома бібліотека для стилізації, яка має обмежені можливості для кастомізації.

*Обрано: Material UI*, оскільки вона написана саме для React, та забезпечує швидкий старт і покращує користувацький досвід (UX)

**Бекенд**:

– База даних

*MySQL* – одна з найпопулярніших реляційних баз даних. Вона підтримує складні запити, транзакції та чудово підходить для структурованих даних з відносинами.

*PostgreSQL* – потужна реляційна база даних, яка підтримує JSON, що дозволяє працювати як з реляційними, так і з документними даними. PostgreSQL має більше функціональних можливостей у порівнянні з MySQL, але складніша в налаштуванні.

*MongoDB* – нереляційна база даних, яка підходить для додатків з динамічними структурами даних, однак менш ефективна у реляційних запитах.

*Обрано: MySQL* як надійне та розповсюджене рішення для реляційних структур.

– Фреймворк:

*NestJS* – фреймворк для Node.js, побудований на TypeScript і розроблений для створення масштабованих серверних додатків. NestJS використовує модульну архітектуру, що забезпечує чітке розділення логіки [4].

*Express* – легкий фреймворк для створення веб-додатків і API. Він більш гнучкий, але не має вбудованої підтримки TypeScript і структурованої архітектури.

*Fastify* – швидкий фреймворк, оптимізований для продуктивності, але менш поширений у розробці великих проєктів.

*Обрано: NestJS* через зручну архітектуру, підтримку TypeScript та можливістю легко інтегруватися з іншими інструментами

– ORM:

*Sequelize* – ORM для Node.js, який підтримує реляційні бази даних і забезпечує вбудовану валідацію та асоціації між моделями [5].

*TypeORM* – ORM, розроблений для TypeScript, що забезпечує гнучкі можливості для налаштування запитів та підтримку різних баз даних. Підходить для складних проєктів, але може бути менш оптимізованим.

*MikroORM* – проста ORM, призначений для роботи з NoSQL базами даних, але також підтримує реляційні бази.

*Обрано: Sequelize* за його стабільність та широку документацію для MySQL.

– Реалізація авторизації та автентифікації:

*Auth0* - auth0 готове рішення для автентифікації (процес підтвердження особикористувача, щоб переконатися, що він є дійсно ти, за кого себе видає), та авторизації (процес надання користувачеві доступу до певних функцій чи ресурсів у системі на основі його прав або ролей), auth0 надійний варіант оскільки має

захист від різних видів хакерських атак. Також ця технологія надає особистий кабінет для створення додатків, управління користувачами та їхніми ролями, та багато іншого [6].

*Firebase Authentication* – сервіс від Google, який підтримує автентифікацію з соціальних мереж і по електронній пошті, але менш гнучкий у порівнянні з Auth0.

*Власна реалізація* – можна створити кастомну систему автентифікації з JWT, однак це вимагає більше часу і може бути менш захищеним.

*Обрано: Auth0* як готове та безпечне рішення, що надає всі необхідні функції без потреби розробки кастомної логіки.

**2.2 База даних: структура та взаємозв'язки таблиць**

Перше, що потрібно визначити при проєктуванні бази даних, – які моделі створюватимуться. Важливо виділити основні сутності, використовуючи принципи нормалізації для забезпечення структурованості й ефективності бази даних [7]. У нашому випадку це таблиці: Користувач, Університет, Факультет, Група, Дисципліна, Завдання, Оцінка, Відвідуваність, Розклад, тощо

Для кожної моделі як індентифікатор будемо використовувати UUID (Universal Unique Identifier) що підвисть безпеку в проєкті від здогадування значень [8].

Зв'язки будемо позначати так:

(1 - 1) - зв'язок один до одного

(1 - n) - зв'язок одни до багатьох

(n - n) - зв'язок багато до багатьох

**Модель Університет(University):**

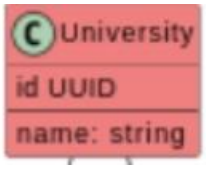


Рисунок. 2.1. Модель Університет

Для цієї моделі достатньо буде мати 1 поле - ім’я

**Модель Факультет(Faculty):**

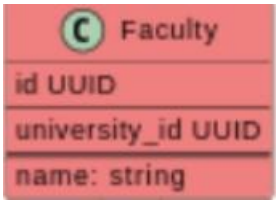


Рисунок. 2.2. Модель Факультет

Зв’язки: University - Faculty (1 - n)

**Модель Група(Group):**

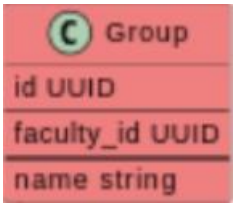


Рисунок. 2.3. Модель Група

Зв’язки: Faculty – Group (1 - n)

**Модель Користувач(User):**

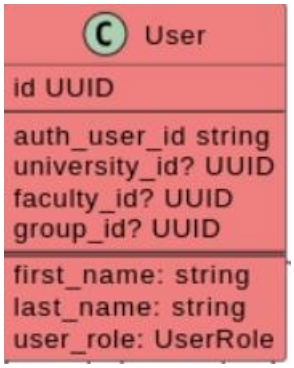


Рисунок. 2.4. Модель Користувач

Зв’язки: University - User (1 - n), Faculty – User(1 - n), Faculty - User(1 - n)

Тип **UserRole** – перерахування ролей (Student, Teacher, Manager, SuperAdmin)

У цій моделі ми не будемо зберігати пароль та пошту, оскільки це буде зроблено на стороні auth0, на його стороні також буде збережено і роль користувача, проте для подальшої зручності збережемо це і в базі даних, **auth_user_id** - це id користувача в системі auth0 тому це поле потрібно для звязування користувача в базі даних та в базі auth0.Зауважимо що поля **university_id**?: UUID, **faculty_id**?: UUID, **group_id**?: UUID є необовязковими, оскільки для різних ролей вони не потрібні, наприклад для Супер Адміністратора - не потрібні ніякі з цих полей, проте для Студента потрібні всі.

**Модель Дисципліна (Subject):**

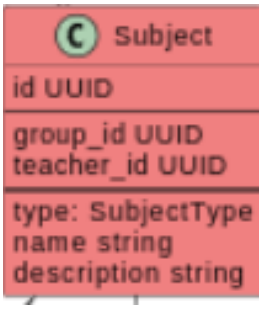


Рисунок. 2.5. Модель Дисципліна(Предмет)

Зв’язки: Group – Subject (1 - n)

**CourseType** - тип оцінювання (EXAM, CREDIT) - екзамен або залік

**Модель Завдання (Assigment):**

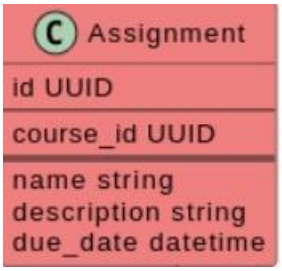


Рисунок. 2.6. Модель Завдання

Зв’язки: Subject- Assigment (1 - n)

**Модель РоботаСтудента(StudentSubmission):**

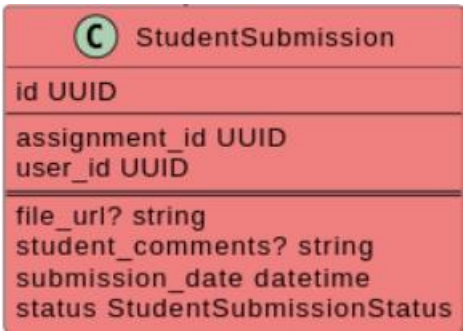


Рисунок. 2.7. Модель Робота Студента

Зв’язки: Assigment - StudentSubmission (1 - 1), User – StudentSubmission (1 - n)

**StudentSubmissionStatus** - статус відправленої роботи (PENDING- в очікувані, REVIEWED - переглянуто, GRADED-оцінено, RESUBMISSION_REQUESTED - запит на допрацювання, LATE_SUBMISSION - пізнє надсилання)

На даний момент імплементувати можливість завантаження файлів немає, тому для відправки завдання додамо можливітьс вставити посилання **file_url** на виконану роботу, наприклад на гугл диск.

**Модель Оцінка(Grade):**

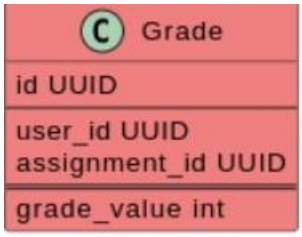


Рисунок. 2.8. Модель Оцінка

Зв’язки: User - Grade(1-n), Assigment - Grade (1- 1)

**Модель Фінальна оцінка за предмет(FinalGrade):**

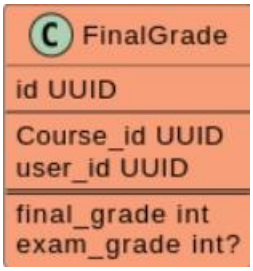


Рисунок. 2.9. Модель Фінальна оцінка

Зв’язки: User - FinalGrade(1-n), Subject- Grade (1-n)

**exam_grade** - оцінка за екзамен не є обов'язково, оскільки форма оцінювання у предмета може бути залік - де екзамен не проводиться.

**Модель Екземпляр Розкладу (SubjectInstance):**

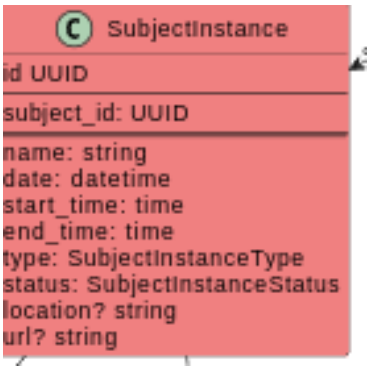


Рисунок. 2.10. Модель Екземпляр Розкладу (Заняття)

Зв'язки: Subject – SubjectInstance (1-n)

**SubjectInstanceType** - тип заняття (LECTURE - лекція, PRACTICE - практика)

**SubjectInstanceStatus** - статус занняття (PENDING-очікується, COMPLETED – завершено, CANCELED - скасовано)

**location**: у випадку очного проведення заннятя дозволяється можливість вказати місце проведення

**url**: у раза дистанційного навчання дозволяється вказати інше посилання для проведення заняття у випадку коли посилання **default_url** в розкладі стало неактуальним

**Модель Відвідуваність (Attendance):**

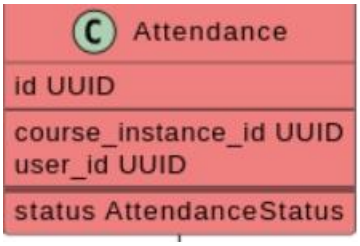


Рисунок. 2.11. Модель Відвідування

Зв'язки: CourseInstance- Attendance(1-n), User - Attendance(1-n)

**AttendanceStatus** - статус відвідування (PRESENT-присутній, ABSENT-відсутній, LATE-запізнився)

Додамо також модель сповіщень для користувача

Зобразимо остаточну модель на uml діаграмі

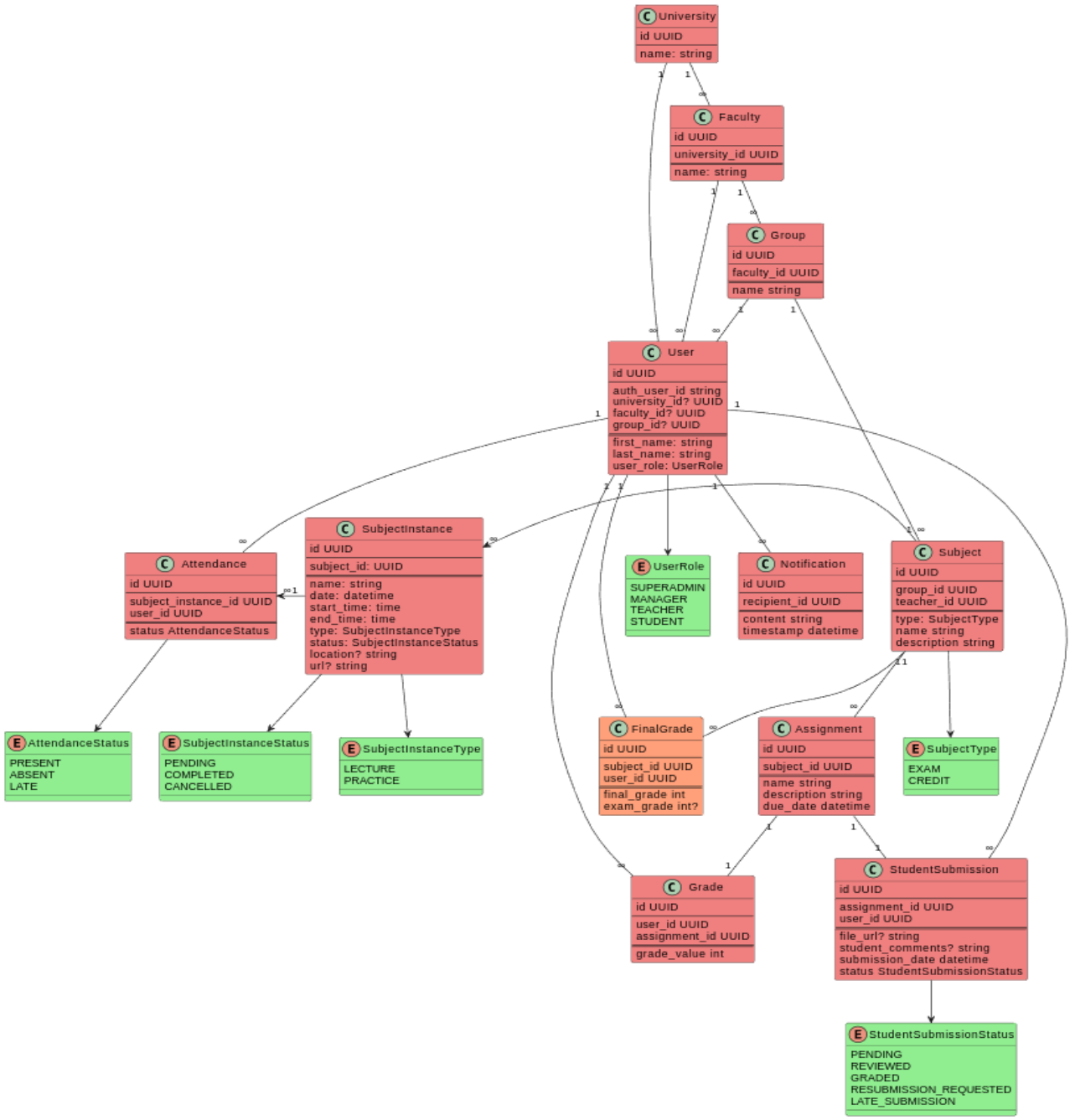


Рисунок. 2.12. Діаграма моделі бази даних

## 2.3 Архітектура системи та її компоненти

### 2.3.1 Загальна архітектура додатку

Користувач може взаємодіяти з інтерфейсом через веб-застосунок, та оскільки мобільний додаток розробляти не планується, дозволимо користувачу завантажувати додаток через PWA. PWA (Progressive Web App) - вебзастосунок, який є гібридом звичайної вебсторінки та мобільного застосунку. Сайт запитає користувача чи бажає він додати застосунок на головний екран, і в разі дозволу

буде мати іконку сайту, та зможе перейти в 1 клік.PWA - також має перевагу - кешуваня даних, що дозволить відкрити застосунок без інтернету [9].

Бекенд буде написаний на Nestjs який буде ‘спілкуватися’ із базою даних та api яке надає auth0.

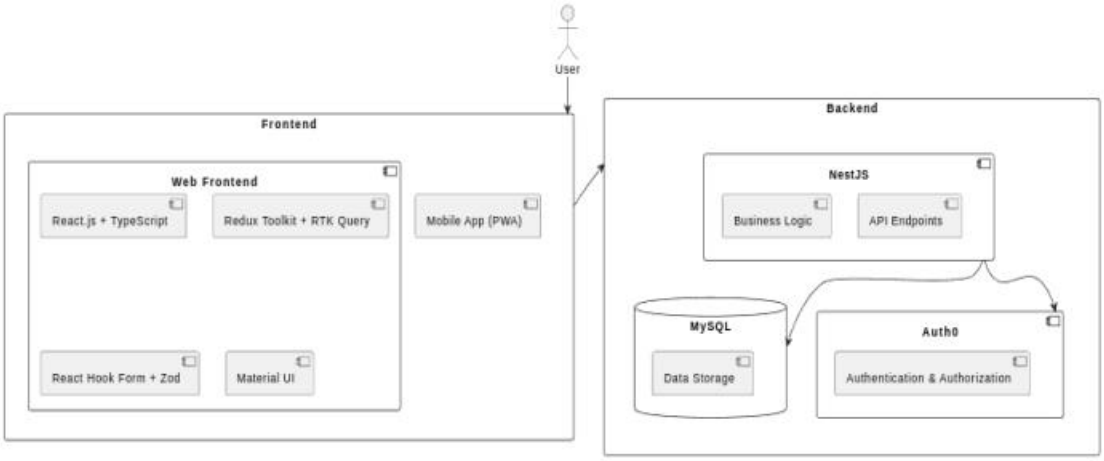


Рисунок. 2.13. Діаграма архітектури додатку

Архітектуру проєкта можна зобразити вищезазначеною діаграмою

### 2.3.2 Front-end Архітектура додатку

Ініцілізуємо Реакт додаток за допомогою Vite.

Vite - це інструмент для створення, який має на меті забезпечити швидшу та ефективнішу розробку сучасних веб-проєктів [10].

Використовуючи команду: npm create vite@latest. Далі в командному рядку вводимо імя проєкту, обираємо біблеотеку React, та обираємо TypeSctipt.

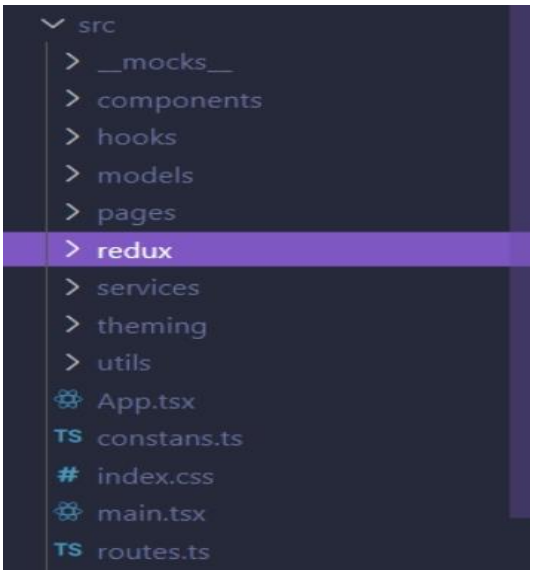


Рисунок. 2.14. Структура react додатку

Папка src в Reactjs містить всі файли та папки з вихідним кодом проєкту [11], розберемо детальніше:

**main.tsx** – Основний файл, з якого починається додаток. Він ініціалізує кореневий компонент і підключає провайдери, такі як Redux і контекст провайдери, що можуть бути необхідні для управління станом додатку. Зазвичай саме тут визначається кореневий елемент, який використовує DOM-елемент для відображення всього інтерфейсу.

**App.tsx** – Головна компонента, що виконує роль "оболонки" для всього додатку. У цьому файлі як праивло налаштовуються основні маршрути для навігації між сторінками, а також компоненти, які будуть постійно присутні, такі як шапка, футер або панель навігації. App.tsx визначає, як виглядатиме структура додатку для користувача.

**Routes.ts** – Файл, де налаштовуються маршрути, щоб додаток міг змінювати сторінки залежно від URL. Це може включати шляхи до сторінок (наприклад, /home, /login, /profile) та прив’язки до компонентів, які будуть рендеритись на цих шляхах. Також тут можна налаштувати назви ендпоінтів для звернення до API.

**Constants.ts** – файл в якому будуть зберігатися константи, для того щоб звертатися за цими даними в одному місці а не дублювати їх в коді

**Pages** – Папка, що містить всі сторінки додатку. Кожна сторінка є окремим компонентом і відповідає за рендеринг основного контенту для певного URL. Наприклад, HomePage, LoginPage, ProfilePage. Завдяки цьому структура додатку стає зрозумілою, а кожна сторінка має свій окремий компонент із відповідним кодом і стилями.

**Components** – Папка для компонентів, які використовуються на сторінках, але не є повноцінними сторінками. Це можуть бути кнопки, форми, модальні вікна, картки або інші повторювані елементи інтерфейсу, які застосовуються на кількох сторінках. Виділення компонентів у цю папку спрощує повторне використання і підтримку коду.

**Models** – Папка для опису моделей і типів, які використовуються у додатку. Завдяки TypeScript можна точно визначити структуру даних, яка передається між

компонентами або отримується від API. Наприклад, тут можна описати структуру об’єктів User, Course, Schedule, що дозволить уникнути помилок, пов’язаних з неправильним використанням даних.

**Redux** – Папка, де зберігається вся логіка з управлінням станом додатку. Використовуючи Redux Toolkit, можна організувати централізоване управління станом додатку, що забезпечує легкість доступу до даних із будь-якої компоненти. Також тут розміщуються запити, створені за допомогою RTK Query, що автоматизують взаємодію з API.

**Services** – Папка для коду, що відповідає за взаємодію з API, якщо RTK Query використати недоцільно, тут можна визначити деякі функції, що виконують запити до сервера.

**__mocks__** - Папка для створення мок-даних, які використовуються при тестуванні компонентів. Моки допомагають симулювати реальні дані для компонентів і перевіряти їхню роботу без необхідності взаємодії з реальним API.

**Hooks** – папка яка буде містити власні написані реакт хуки.

**Utils** – Папка для утиліт і допоміжних функцій. Тут зберігаються функції, які можуть бути використані в різних частинах додатку, наприклад, форматування дати, валідатори даних або інші загальні алгоритми.

**Index.css** – файл який описує глобальні стилі додатку.

### 2.3.3 Back-end Архітектура додатку

Створимо таку структуру проєкту

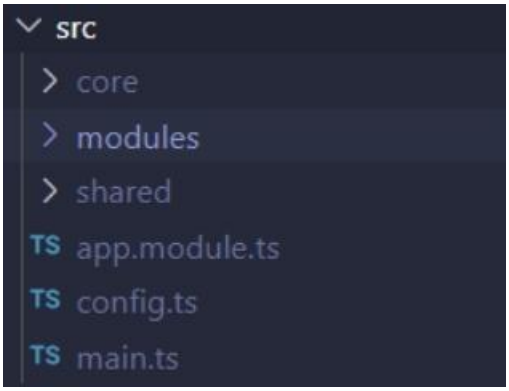


Рисунок. 2.15. Структура nestjs додатку

**Main.ts** – Головний файл, який запускає додаток. Він ініціалізує кореневий модуль і починає обробку запитів.

**App.module.ts** – файл в якому підключають та налаштовують всі інші модулі

**Config.ts** – файл, який містить всі необхідні зміні середовища, для запуску проєкта, такі як: токени від auth0, дані бази даних (паролі, назва), та інше

**Core** – папка яка буде містити логіку стосувно осовного налаштування проєкту, а саме авторизація та автентифікація, підключення бази даних.

**Shared** – Каталог із допоміжними функціями, які використовуються в усьому проєкті. Це можуть бути константи, декоратори, інтерфейси та інші повторно використовувані елементи.

**Modules** – оновний каталог для модулів додатку. Кожен модуль (наприклад, UserModule, ScheduleModule) містить бізнес-логіку, контролери, сервіси та моделі, що відповідають за певні функції, як-от CRUD операції.

Тепер розглянемо архітектуру моделей яку визначає сам nestjs.

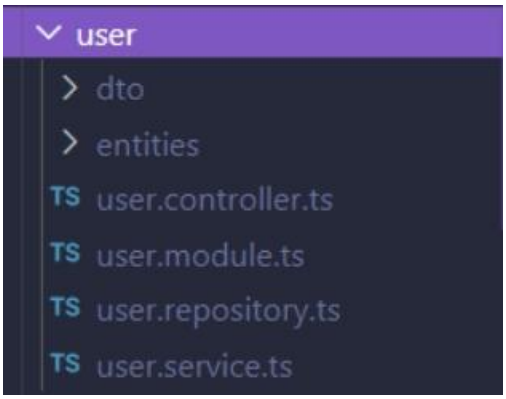


Рисунок. 2.16. Структура рхітектури моделі nestjs

**Dto** – папка яка містить файли які визначають що повинен отримати сервер, для того щоб їх обробити [12].

**Entities** – папка яка містить файли які повністю описують модель – поля та її тип даних, зв’язки між моделями – один до одного, один до багатьох чи багагато до багатьох.

**Модуль (Module):** Основний елемент структури додатку, що групує логічно пов’язані частини коду, такі як контролери, сервіси та репозиторії, встановлює імпорти та експорти.

**Контролер (Controller)**: Відповідає за обробку HTTP-запитів, таких як GET, POST, PUT, DELETE, і викликає відповідні методи сервісів. Контролер

отримує дані від користувача або клієнта, передає їх на обробку у сервіс і повертає відповідь. Наприклад, контролер UserController обробляє запити, пов'язані з користувачами, і передає їх у UserService.

**Сервіс (Service)**: Відповідає за бізнес-логіку, обробку даних, перевірку та підготовку до збереження в базі даних. Сервіси зазвичай використовуються для розділення логіки додатку від обробки запитів, що дозволяє легко їх тестувати та змінювати. Наприклад, UserService може обробляти інформацію про користувачів та виконувати всі операції з даними, необхідні для контролера.

**Репозиторій (Repository)**: Спеціальний клас або файл, який безпосередньо взаємодіє з базою даних [13]. Репозиторій виконує операції з базою даних, такі як збереження, оновлення, видалення та отримання даних, і використовує ORM для комунікації з базою даних. Наприклад, UserRepository взаємодіє з таблицею користувачів і виконує запити на отримання або зміну даних.

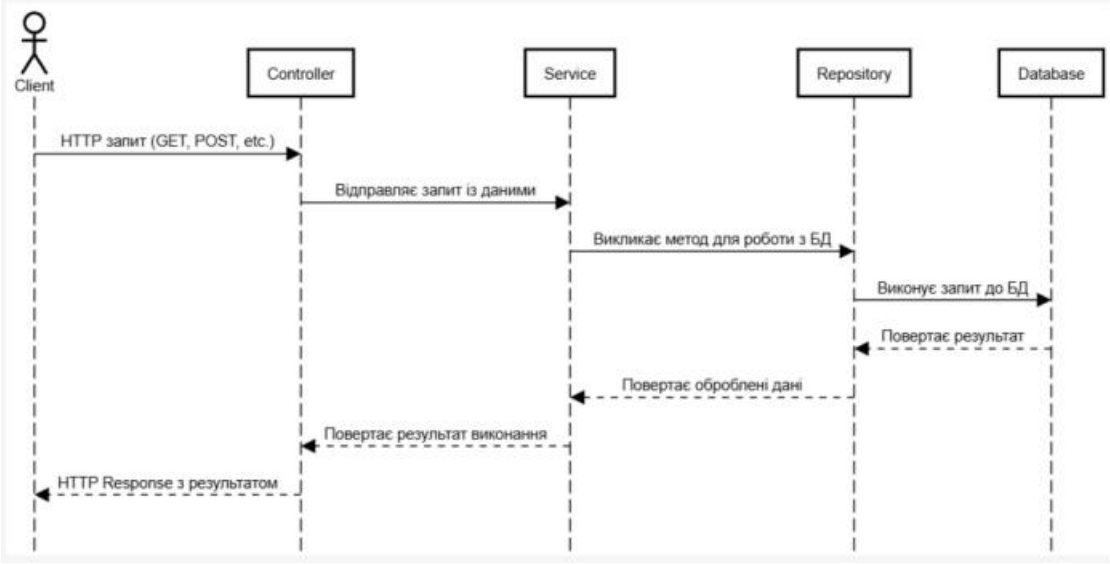


Рисунок. 2.17. Діаграма послідовності обробки HTTP запиту

# РОЗДІЛ 3
# РЕАЛІЗАЦІЯ СИСТЕМИ

## 3.1 Серверна частина

Весь код серверної частини додатку знаходиться за цим посиланням https://github.com/Maks262626/LearnLog-be/tree/v1

**Початкова ініціалізація**

Бекенд проєкту ініціалізовано за допомогою NestJS CLI, що створює базову структуру. Для запуску проєкту використовується файл main.ts, який налаштовує сервер і підключає необхідні модулі.

- **NestFactory.create(AppModule)**: Створює екземпляр додатка на основі головного модуля AppModule.
- **ValidationPipe**: Використовується для автоматичної валідації вхідних даних на основі DTO (Data Transfer Objects), що підвищує безпеку та зменшує ризик помилок.

**Налаштування змінних середовища**

Для безпечного керування конфігурацією, такою як параметри підключення до бази даних MySQL і Auth0, використовується файл .env. Цей файл містить приватні дані, такі як хост, порт, ім’я користувача, пароль бази даних, а також параметри аутентифікації Auth0. Модуль @nestjs/config дозволяє завантажувати ці змінні в додаток, забезпечуючи їх доступність у коді.

Приклад вмісту файлу .env:

DB_HOST=localhost

DB_PORT=3306

DB_USER=root

DB_PASSWORD=password

DB_NAME=learnlog

AUTH0_DOMAIN=your-domain.auth0.com

AUTH0_AUDIENCE=your-audience

Для подальшого доступу до всіх змінних середовища створено файл конфігурації:


Рисунок. 3.1. Файл конфігурації config.ts

В головному файлі app.module.ts ці змінні підключаються для всього проєкту:

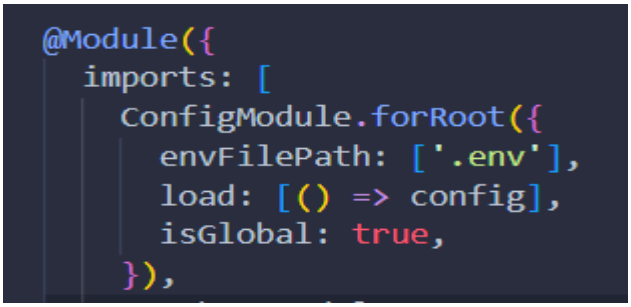


Рисунок. 3.2. Файл app.module.ts

- **ConfigModule.forRoot**: Робить змінні середовища доступними глобально у всьому додатку.
- **envFilePath**: Вказує шлях до файлу .env.
- **isGlobal: true**: Усуває необхідність імпортувати ConfigModule у кожен модуль, спрощуючи конфігурацію.

**Інтеграція Swagger**

Swagger, реалізований через модуль @nestjs/swagger, використовується для створення інтерактивної документації API. Це дозволяє розробникам переглядати доступні ендпоінти, їх параметри та формати відповідей [14]. Налаштування

Swagger виконується в main.ts, де визначається назва API, опис і підтримка автентифікації через JWT.

**Підключення до бази даних**

Для роботи з базою даних MySQL у проєкті використовується Sequelize – ORM для Node.js. Конфігурація підключення налаштовується в файлі database.service.ts:

```
export const databaseProviders = [
  {
    provide: 'SEQUELIZE',
    useFactory: async (configService: ConfigService) => {
      const sequelize = new Sequelize({
        dialect: 'mysql',
        host: configService.get<string>('db.host'),
        port: configService.get<number>('db.port'),
        username: configService.get<string>('db.username'),
        password: configService.get<string>('db.password'),
        database: configService.get<string>('db.name'),
      });
      sequelize.addModels([
        User,
        University,
        Faculty,
        Group,
        Subject,
        FinalGrade,
        Assignment,
        StudentSubmission,
        Grade,
        SubjectInstance,
        Attendance,
      ]);
      return sequelize;
    },
    inject: [ConfigService],
  },
];
```

Рисунок. 3.3. Файл підключення бази даних database.service.ts

- **Sequelize**: Налаштовує підключення до MySQL, використовуючи параметри з .env.
- **sequelize.addModels**: Реєструє моделі (наприклад, User, University, Faculty) для роботи з базою даних.
- **Ін’єкція ConfigService**: Забезпечує доступ до конфігураційних змінних.

Sequelize дозволяє визначати моделі як класи TypeScript, що спрощує роботу з даними та забезпечує типізацію.

**Інтерцептор**

Для забезпечення однакового формату відповідей API і централізованої обробки помилок у проєкті використано власний інтерцептор ResponseInterceptor [15]. Інтерцептори в NestJS дозволяють перехоплювати запити

та відповіді, трансформувати дані або обробляти помилки до їх відправлення клієнту.

Метод responseHandler приймає результат обробки запиту (res) і повертає об’єкт із ключем data, який містить ці дані. Наприклад, якщо контролер повертає об’єкт { id: "123", name: "Example" }, то відповідь виглядатиме так:

```
{
  "data": {
    "id": "123",
    "name": "Example"
  }
}
```

Це забезпечує однакову структуру для всіх успішних відповідей API.

Метод обробляє помилки, отримуючи екземпляр помилки (exception) і контекст запиту (context), отримує об’єкт HTTP-відповіді через context.switchToHttp().getResponse().

Визначає статус помилки: якщо це HttpException, береться її статус (наприклад, 400 для BadRequestException), інакше використовується 500 (Internal Server Error). Формує JSON-відповідь із назвою помилки (error) та її повідомленням (message).

Приклад відповіді на помилку:

```
{
  "error": "BadRequestException",
  "message": "Invalid input data"
}
```

**Розробка модулів**

Код бекенду системи побудовано на основі модульної архітектури NestJS, що забезпечує чіткий поділ відповідальностей і полегшує підтримку та масштабування проєкту. Для кожної сутності системи (User, University, Faculty, Group, Subject, Assignment, Grade, Attendance, Schedule) створено окремий модуль, такий як UserModule, UniversityModule тощо.

Кожен модуль складається з:

- **Контролерів** (@Controller): Обробляють HTTP-запити,визначають маршрути, отримують дані та викликають методи сервісів.
- **Сервісів** (@Injectable): Містять бізнес-логіку, таку як створення чи оновлення даних, і взаємодіють із базою через репозиторії.
- **Репозиторіїв**: Інкапсулюють логіку доступу до бази даних, забезпечуючи абстракцію над Sequelize.
- **Моделей** (@Table): Визначають структуру даних для кожної сутності за допомогою Sequelize.
- **DTO** (Data Transfer Object) – об’єкт, який використовується для передачі даних між клієнтом і сервером

Така організація відповідає принципу єдиної відповідальності, що робить код структурованим, читабельним і зручним для тестування.

Маршрути до кожного контролера винесено в окремий файл, що підвищує модульність. Наприклад: user.route.ts

```ts
export const USER_CONTROLLER = 'user';

export const USER_ROUTES = {
  REGISTER: 'register',
  ME: 'me',
  FIND_ALL_USERS: '',
  GET_TEACHERS_BY_FACULTY_ID: 'teachers/:id',
  GET_USERS_IN_MY_GROUP: 'in-my-group',
  GET_USERS_IN_MY_FACULTY: 'in-my-faculty',
  FIND_USER: ':id',
  FIND_USERS_FROM_UNIVERSITY: 'university/:id'
  FIND_USERS_FROM_FACULTY: 'faculty/:id',
  FIND_USERS_FROM_GROUP: 'group/:id',
  UPDATE_USER: ':id',
  SET_USER_ROLE: 'role/:id',
  DELETE_USER: ':id'
} as const;
```

Рисунок. 3.4. Файл шляхів для ендпоінтів користувачів user.routes.ts

У проєкті активно застосовуються декоратори NestJS для спрощення роботи з маршрутами, валідацією даних і безпекою:

- **Маршрути**: Декоратори @Controller, @Get, @Post, @Put, @Delete визначають шляхи та методи HTTP. Наприклад, @Controller('users') задає

базовий шлях /users, а @Get(':id') – маршрут для отримання користувача за ідентифікатором.

- **Валідація**: Декоратори @Body і @Param використовуються разом із DTO (Data Transfer Objects) для типізації та перевірки вхідних даних.
- **Безпека**: Декоратор @UseGuards застосовується для захисту маршрутів. Наприклад, AuthGuard перевіряє JWT-токен, а RoleGuard – роль користувача для реалізації RBAC.

**Політика доступу через UserPolicyService**

Для надання доступу користувачам конкретної ролі виконувати запит здійснюєься за допомогою декоратора @UseGuards, та цього в деяких випадках недостатньо, наприклад: є метод для підтвердження користувача, який приймає ідентифікатор користувача, та роль яку потрібно присвоїти, цей метод дозволений для ролей: superadmin, manager. Але це обмеження на ролі, не завадить будь-якому менеджеру присвоїти роль будь-якому користувачу маючи його ідентифікатор, саме тому, для реалізації складної логіки RBAC у проєкті створено сервіс UserPolicyService, який централізує перевірку прав доступу до сутностей (User, Assignment, Grade, Attendance тощо). Нижче наведено метод в сервісі UserPolicyService:

```
private async isFacultyMatch(faculty_id: string, user: User): Promise<boolean> {
  return this.isSuperAdmin(user) || user.faculty_id === faculty_id;
}

async canAssignRole(
  caller: UserRoleName,
  target: UserRoleName,
  userId: string,
  callerUser: User,
): Promise<boolean> {
  if (caller === UserRoleName.MANAGER && ![UserRoleName.TEACHER, UserRoleName.STUDENT].includes(target)) {
    return false;
  }
  const user = await this.userRepository.findUser(userId);
  if (!user) return false;

  return this.isFacultyMatch(user.faculty_id, callerUser);
}
```

Рисунок. 3.5. Метод canAssignRole у файлі user-policy.service.ts

canAssignRole: правило, яке визначає чи може користувач призначити роль іншому користувачу. Тобто користувач може змінити роль іншого, якщо він має

роль superadmin або ж має роль менеджера та факультет користувача співпаде із факультетом користувача роль якого присвоюється.

**Налаштування Auth0**

Для використання Auth0 у проєкті спочатку необхідно налаштувати обліковий запис і застосунок на офіційному сайті Auth0.

Створивши новий обліковий запис, після реєстрації створюється «tenant» – основний простір для вашого проєкту, який об’єднує всі налаштування.

Створення нового застосунку

У панелі керування Auth0 переходимо в розділ «Applications» (Застосунки)

Натискаємо Create Application (Створити застосунок)

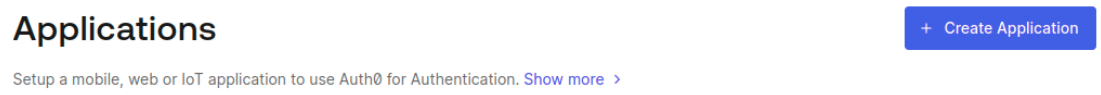


Рисунок. 3.6. Створення застосунку в Auth0

Обираємо тип «Single Page Application» (SPA), оскільки фронтенд проєкту побудовано на React.

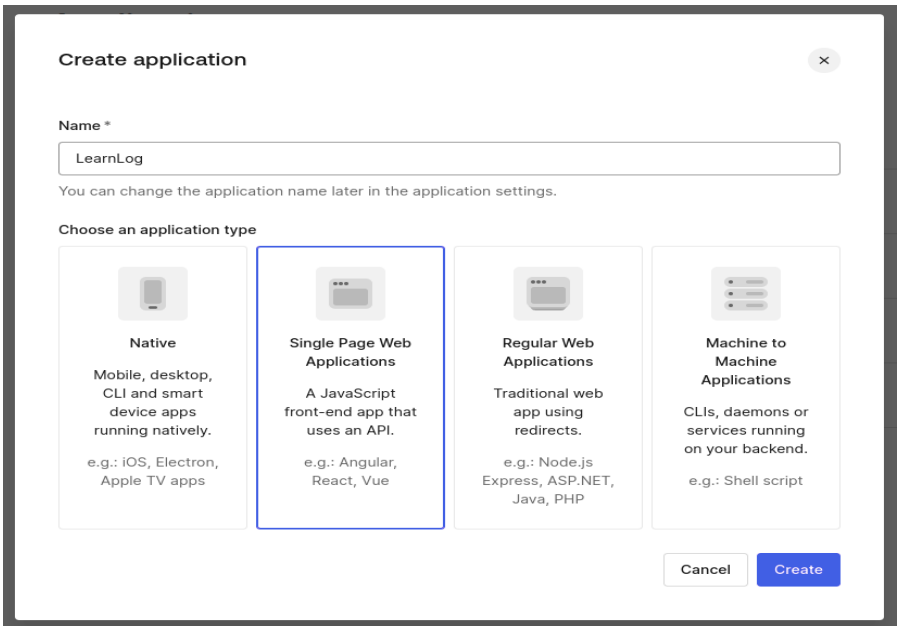


Рисунок. 3.7. Вибір типу SPA застосунку в Auth0

Далі у налаштуваннях застосунку потрібно вказати такі опції:

- Allowed Callback URLs: URL, на який користувач буде перенаправлений після входу
- Allowed Logout URLs: URL для виходу з системи, якщо потрібно.

- Allowed Web Origins: Домен нашого фронтенду для CORS.

У нашому випадку це все буде http://localhost:5173 де відбувається розробка клієнтської частини застосунку

Також потрібно створити ще додаток у тенанті auth0 з типом Machine to Machine, що дозволить бекенду виконувати серверні операції незалежно від користувача,наприклад, отримувати дані чи управляти користувачами через API.

Далі потрібно додати наші визначені ролі у вкладці User Managment → Roles.

Для кожної ролі створюється, свій ідентифікатор, який ми можемо побачити в нашалтуванях кожної ролі.

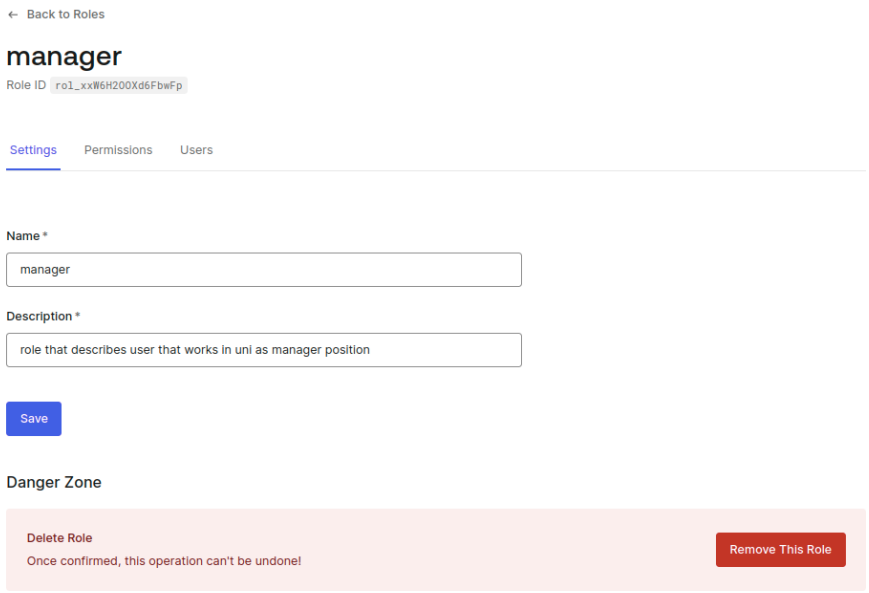


Рисунок. 3.8. Сторінка Налаштування ролі в Auth0

Їх ми будемо використовувати в коді, для встановлення ролей користувачам.

Також потрібно в Auth0 Management API(застосунок в тенанті який є за замовчуванням), надати потрібні дозволи, які зображені на рисунку.

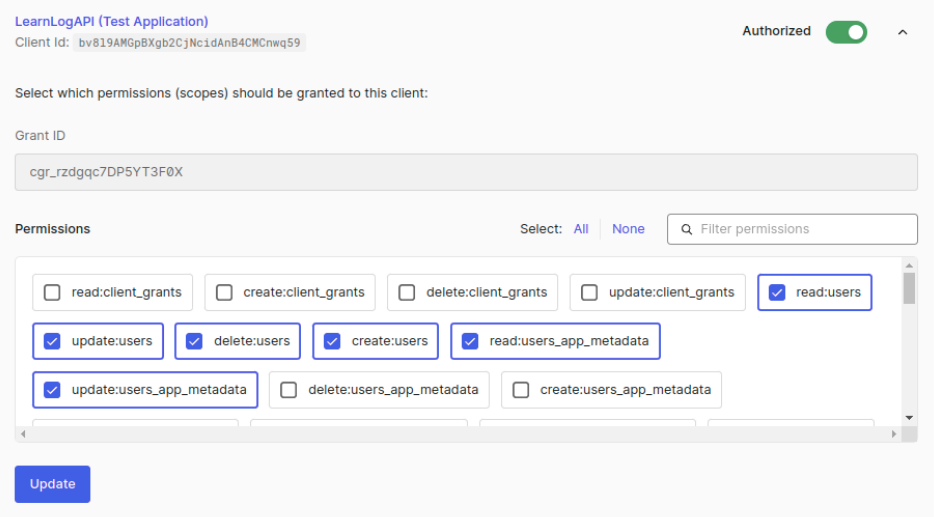


Рисунок. 3.9. Сторінка налаштування Auth0 Management API в Auth0

На цьому налаштування auth0 в особистому кабінеті завершується.

Реалізуємо сервіс який міститиме наступні функції: присвоєння ролі користувачу, отримання ролей, видалення ролей, видалення користувача.

Для перевірки токена, визначена стратегія JWT-аутентифікації у файлі jwt.strategy.ts, яка в методі validate формує об’єкт користувача для подальшої обробки в системі [16].

```
async validate(payload: any): Promise<any> {
  const auth0_user_id = payload.sub;
  const roles = await this.authzService.getUserRolesAuth0(auth0_user_id);
  const user = await this.userService.getMe(auth0_user_id);
  if (user) {
    return {
      id: user.id,
      auth0_user_id,
      role: roles[0],
      faculty_id: user.faculty_id,
      university_id: user.university_id,
      group_id: user.group_id,
      first_name: user.first_name,
      last_name: user.last_name,
    };
  }
  return { auth0_user_id, role: roles[0] };
}
```

Рисунок. 3.10. Метод validate у файлі jwt.strategy.ts

Тепер можна реалізувати потрібні декоратори та гарди, які необхідні для обробки запитів: Декоратор CurrentUser (поточний користувач) – повертає дані користувача який викликав запит, гард Role який дозволяє передати в нього потрібні ролі, і він поверне результат чи є така роль у користувача.

Система підтримує базові CRUD-операції (Create, Read, Update, Delete) [17] для всіх сутностей бази даних, таких як User, Faculty, Group, Subject, Assignment, Grade, FinalGrade, SubjectInstance і Attendance. Після забезпечення базової функціональності розроблено спеціалізовані методи, що виходять за межі CRUD і надають додаткові можливості для управління освітнім процесом. Ці методи враховують ролі користувачів (студент, викладач, менеджер, супер-адміністратор) і забезпечують контекстно-залежний доступ через RBAC.

Для кожної сутності бази даних реалізовано модуль із п'ятьма компонентами: сутність (entity), DTO, контролер, сервіс і репозиторій. Ця структура, яка описана вище, застосовується до всіх моделей.

Приклад базових CRUD-операцій, для моделі Group, контролер (GroupController) містить ендпоінти:

- POST /groups: Створення групи.
- GET /groups: Отримання всіх груп.
- GET /groups/:id: Отримання групи за ідентифікатором.
- PATCH /groups/:id: Оновлення групи.
- DELETE /groups/:id: Видалення групи.

**Спеціалізовані методи для моделей**

Окрім базових CRUD-операцій, для різних моделей реалізовані спеціалізовані методи, які враховують потребу різних ролей користувачів і забезпечують розширену функціональність. Нижче наведено опис цих методів для ключових моделей.

**Модель User**

- *userMe*: Повертає інформацію про поточного користувача, отриману з JWT-токена через декоратор @CurrentUser.
- *getTeachersByFacultyId:* Повертає список викладачів за ідентифікатором факультету.
- *findUsersFromUniversity, findUsersFromFaculty, findUsersFromGroup:* Повертають користувачів, пов'язаних із університетом, факультетом або групою за відповідними ідентифікаторами.

- *findUsersFromMyFaculty, findUsersInMyGroup*: Повертають користувачів із факультету або групи поточного користувача, використовуючи його faculty_id або group_id.
- *setUserRole*: Призначає роль користувачу з перевіркою прав через UserPolicyService.

**Модель Faculty**

- *findFacultiesInMyUniversity*: Повертає всі факультети університету, до якого належить поточний користувач, використовуючи його university_id.
- *findFacultiesByUniversityId:* Повертає факультети за заданим ідентифікатором університету.

**Модель Group**

- *findGroupsInMyFaculty*: Повертає всі групи з факультету поточного користувача.
- *findGroupsByFacultyId:* Повертає групи за ідентифікатором факультету.

**Модель Subject**

Модель Subject включає методи для роботи з предметами залежно від ролі користувача:

- *getTeacherSubjects*: Повертає всі предмети, які викладає поточний викладач.
- *getMyCourses*: Для менеджерів повертає всі предмети на їхньому факультеті.
- *getSubjectsInMyGroup*: Для студентів повертає предмети, які викладаються в їхній групі.
- *getSubjectsByGroupId*: Повертає предмети за ідентифікатором групи.

Особливість створення: При створенні предмета автоматично створюються записи моделей FinalGrade для всіх студентів групи з оцінкою null. Це реалізовано через транзакцію яка гарантує, що якщо створення FinalGrade не вдасться, предмет не буде створено, зберігаючи цілісність даних.

```
async create(createSubjectDto: CreateSubjectDto): Promise<Subject> {
  const transaction = await this.sequelize.transaction();
  try {
    const subject = await this.subjectRepository.createSubject(createSubjectDto, transaction);
    const { group_id } = subject;
    const users = await this.userRepository.findUsersFromGroup(group_id);
    const finalGradeInstances: CreateFinalGradeDto[] = [];
    users.forEach((user) => {
      const finalGradeDto: CreateFinalGradeDto = {
        user_id: user.id,
        subject_id: subject.id,
        final_grade: null,
      };
      finalGradeInstances.push(finalGradeDto);
    });
    await this.finalGradeRepository.bulkCreate(finalGradeInstances, transaction);
    await transaction.commit();
    return subject;
  } catch (err) {
    await transaction.rollback();
    throw err;
  }
}
```

Рисунок. 3.11. Метод створення дисципліни

**Модель Assignment**

- *findBySubjectId:* Повертає всі завдання за ідентифікатором предмета.

Особливість створення: При створенні завдання автоматично створюються записи в моделі Grade для всіх студентів групи з оцінкою null, також із використанням транзакції

```
async create(createAssignmentDto: CreateAssignmentDto): Promise<Assignment> {
  const transaction = await this.sequelize.transaction();
  try {
    const assignment = await this.assignmentRepository.createAssignment(
      createAssignmentDto,
      transaction,
    );

    const { subject_id } = assignment;
    const subject = await this.subjectRepository.findSubject(subject_id);
    const { group_id } = subject;

    const users = await this.userRepository.findUsersFromGroup(group_id);
    const gradeInstances: CreateGradeDto[] = [];
    users.forEach((user) => {
      const gradeDto: CreateGradeDto = {
        user_id: user.id,
        assignment_id: assignment.id,
        grade_value: null,
      };
      gradeInstances.push(gradeDto);
    });

    await this.gradeRepository.bulkCreate(gradeInstances, transaction);

    await transaction.commit();
    return assignment;
  } catch (err) {
    await transaction.rollback();
    throw err;
```

Рисунок. 3.12. Метод створення завдання

**Модель Grade**

- *getGradeByUserIdAndAssignmentId:* Повертає оцінку користувача за конкретним завданням.

**Модель SubjectInstance**

Модель SubjectInstance відповідає за розклад занять і підтримує методи для різних ролей:

- *getByGroup:* Для менеджерів повертає розклад занять для заданої групи у вказаний період.
- *getTeacherSchedule*: Для викладачів повертає їхній розклад занять.
- *getInMyGroup*: Для студентів повертає розклад занять їхньої групи.

Форматування розкладу: Метод mapLecturesByDay у сервісі SubjectInstance-Service групує заняття за днями тижня для зручного відображення на фронтенді:

```
mapLecturesByDay(subjectInstances: SubjectInstance[]): Record<Day, SubjectInstance[]> {
  const daysMap: Record<number, Day> = {
    0: 'Sun',
    1: 'Mon',
    2: 'Tue',
    3: 'Wed',
    4: 'Thu',
    5: 'Fri',
    6: 'Sat',
  };
  const lecturesByDay: Record<Day, SubjectInstance[]> = {
    Mon: [],
    Tue: [],
    Wed: [],
    Thu: [],
    Fri: [],
    Sat: [],
    Sun: [],
  };
  for (const instance of subjectInstances) {
    const date = dayjs(instance.date);
    const dayNumber = date.day();
    const dayName = daysMap[dayNumber];

    lecturesByDay[dayName].push(instance);
  }

  return lecturesByDay;
}
```

Рисунок. 3.13. Метод mapLecturesByDay

Цей метод відображає розкладу у форматі тижневого календаря, що відповідає потребам фронтенду.

Особливість створення: При створенні заняття автоматично створюються записи моделей Attendace для всіх студентів групи із значенням за замовчуваням –відсутній, також із використанням транзакції:

```
async create(createSubjectInstanceDto: CreateSubjectInstanceDto): Promise<SubjectInstance> {
  const transaction = await this.sequelize.transaction();
  try {
    const subjectInstance = await this.subjectInstanceRepository.createSubjectInstance(
      createSubjectInstanceDto,
      transaction,
    );

    const { subject_id } = createSubjectInstanceDto;
    const subject = await this.subjectRepository.findSubject(subject_id);
    const { group_id } = subject;
    const users = await this.userRepository.findUsersFromGroup(group_id);
    const attendances: CreateAttendanceDto[] = [];
    for (const user of users) {
      const attendance: CreateAttendanceDto = {
        subject_instance_id: subjectInstance.id,
        user_id: user.id,
        status: AttendanceStatus.ABSENT,
      };
      attendances.push(attendance);
    }
    await this.attendancerepository.bulkCreate(attendances, transaction);
    await transaction.commit();
    return subjectInstance;
  } catch (err) {
    await transaction.rollback();
    throw err;
  }
}
```

Рисунок. 3.14. Метод mapLecturesByDay

**Модуль звітів**

Модуль звітів складається з контролера (ReportsController) і сервісу (ReportsService), які взаємодіють із репозиторіями інших моделей для збору даних. Контролер обробляє HTTP-запити, а сервіс реалізує логіку формування звітів і генерації файлів. Оскільки звіт не є сутністю в базі даних, реалізація модуля не потребує Репозиторія, Моделі, DTO.

Бібліотеки jsPDF і ExcelJS обрано для генерації PDF і XLSX звітів завдяки їхній детальній документації, простоті використання та сумісності з NestJS для серверної генерації [18,19].

Контролер визначає ендпоінти для створення індивідуальних і групових звітів, захищених через AuthGuard і RoleGuard. Основні ендпоінти включають:

**Індивідуальні звіти для студентів**

- GET /reports/student-grades: Повертає оцінки студента (JSON).

- GET /reports/student-individual-attendances: Повертає відвідуваність студента (JSON).
- GET /reports/student-individual-grades-pdf: Генерує PDF-звіт з оцінками.
- GET /reports/student-individual-attendances-pdf: Генерує PDF-звіт із відвідуваністю.

**Групові звіти для менеджерів**

- GET /reports/student-grades/:id: Повертає оцінки студента за ID.
- GET /reports/student-individual-attendances/:id: Повертає відвідуваність студента за ID.
- GET /reports/student-group-attendance-summary/:id: Повертає зведений звіт відвідуваності групи.
- GET /reports/student-group-grade-summary/:id: Повертає зведений звіт оцінок групи.
- GET /reports/student-group-attendance-individual-pdf/:id: Генерує PDF-звіт відвідуваності всіх студентів групи.
- GET /reports/student-group-attendance-summary-xlsx/:id: Генерує XLSX-звіт відвідуваності групи.
- GET /reports/student-group-grade-summary-xlsx/:id: Генерує XLSX-звіт оцінок групи.

Сервіс ReportsService реалізує логіку збору даних і генерації звітів. Він взаємодіє з репозиторіями моделей (UserRepository, SubjectRepository, GradeRepository тощо) і використовує UserPolicyService для перевірки прав доступу. Основні методи:

- *studentGrades*: Формує звіт про оцінки студента, включаючи фінальні оцінки (FinalGrade) і оцінки за завдання (Grade).
- *individualStudentAttendances*: Формує звіт про відвідуваність студента за всіма предметами.
- *studentGroupGradesSummaryReport*: Генерує зведений звіт оцінок для всіх студентів групи.

- *studentGroupAttendanceSummary*: Генерує зведений звіт відвідуваності для групи.
- *generateStudentAttendanceReportPdf*, *generateIndividualStudentGradesReportPdf*: Створюють PDF-звіти для індивідуальних даних, використовуючи jsPDF.
- *generateIndividualAttendanceReportsByGroupPdf*: Генерує PDF-звіт відвідуваності для всіх студентів групи.
- *generateStudentGroupAttendanceSummaryXlxs*, *generateStudentGroupGradesSummaryXlxs*: Створюють XLSX-звіти для груп, використовуючи ExcelJS.

## 3.2 Клієнтська частина

Реалізація фронтенду базується на сучасних веб-технологіях, які відповідають вимогам до зручності, безпеки та масштабованості. Цей розділ описує використані технології, ключові аспекти реалізації, інтеграцію з бекендом.

Весь код клієнтської частини додатку знаходиться за цим посиланням https://github.com/Maks262626/LearnLog-fe/tree/v1

**Підключення Auth0 на клієнтській стороні**

Бібліотека @auth0/auth0-react, забезпечує безпечний вхід і захист маршрутів. Головний компонент App.tsx обгорнуто в Auth0Provider, що налаштовує домен, Client ID і redirect URI для Auth0, які дістаються і файла змінних середовища .env

**Організація маршрутів**

Файл App.tsx є основним входом фронтенду системи, де налаштовується маршрутизація за допомогою React Router. Він визначає набір маршрутів, кожен з яких може бути доступний лише для певних ролей користувачів, що реалізовано через компоненти ProtectedRoute та RoleProtectedRoute.

**Захист доступу**

Компонент ProtectedRoute забезпечує, що лише аутентифіковані користувачі можуть отримати доступ до захищених сторінок. Він використовує хук useAuth0 для перевірки статусу аутентифікації та завантаження, а також викликає

useGetMeQuery для отримання даних користувача з бекенду, щоб переконатися, що користувач завершив реєстрацію та був схвалений. Наприклад, якщо користувач не аутентифікований, його перенаправляють на сторінку входу, а якщо реєстрація не завершена – на сторінку реєстрації.

Компонент RoleProtectedRoute додатково обмежує доступ на основі ролі, перевіряючи, чи містить масив ролей користувача, отриманий з токена Auth0, необхідну роль. Якщо ні, користувач перенаправляється на сторінку профілю.

**Динамічний вибір змісту**

У App.tsx використовуються функції getHomeComponent та getUsersComponent, які динамічно повертають різні компоненти залежно від ролі користувача. Наприклад, для ролі 'superadmin' домашня сторінка відображає <Home />, для 'manager' – <ManagerHome />, для 'teacher' – <TeacherHome />, а для 'student' – <StudentHome />. Це дозволяє показувати різні компоненти на одному й тому самому url шляху відповідно до різних ролей користувача

Для забезпечення ефективної взаємодії фронтенду системи із бекендом використано RTK Query – інструмент із пакету Redux Toolkit, який спрощує виконання HTTP-запитів, управління кешем, забезпечує типізовану взаємодію з REST API, автоматичне оновлення даних і обробку помилок.

Файл apiSlice.ts визначає базовий API-клієнт для всіх запитів до бекенду. Він налаштовує fetchBaseQuery із базовим URL, авторизацією через JWT-токен і обробкою помилок 401.

Для кожного модуля бекенду, (користувачі, групи, предмети, розклад, відвідуваність, завдання, оцінки, звіти), створено APISlice, який реалізовує ендпоінти. Кожен файл використовує apiSlice.injectEndpoints для визначення запитів (queries) і мутацій (mutations), що відповідають CRUD-операціям і спеціалізованим методам бекенду.

**Загальна структура API Slice для модулів:**

- **Queries**: запити типу GET, для отримання даних.
- **Mutations**: POST, PATCH, DELETE запити для створення, оновлення чи видалення даних.

- **Теги**: Використовуються providesTags для позначення кешованих даних і invalidatesTags для їх оновлення після мутацій.
- **Хуки**: RTK Query генерує хуки, такі як useGetUsersQuery чи useUpdateGradeMutation, для використання в компонентах.

**Реєстрація**

Реєстрація має 2 етапи, перший – форма реєстрації Auth0, яка зберігає чутливі дані користувача на своїй стороні, другий етап – форма реєстрації RegisterForm, яка дозволяє користувачам завершити процес реєстрації після входу через Auth0, вказавши такі дані як: ім'я, прізвище, університет, факультет і, за необхідності, групу. Компонент реалізовано з використанням React Hook Form, Zod для валідації та RTK Query для отримання даних і відправки запиту на реєстрацію [20].

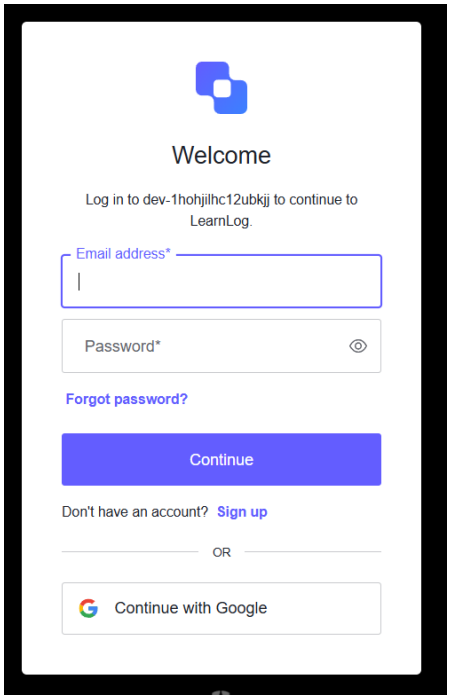


Рисунок. 3.15. Форма Реєстрації першого етапу Auth0

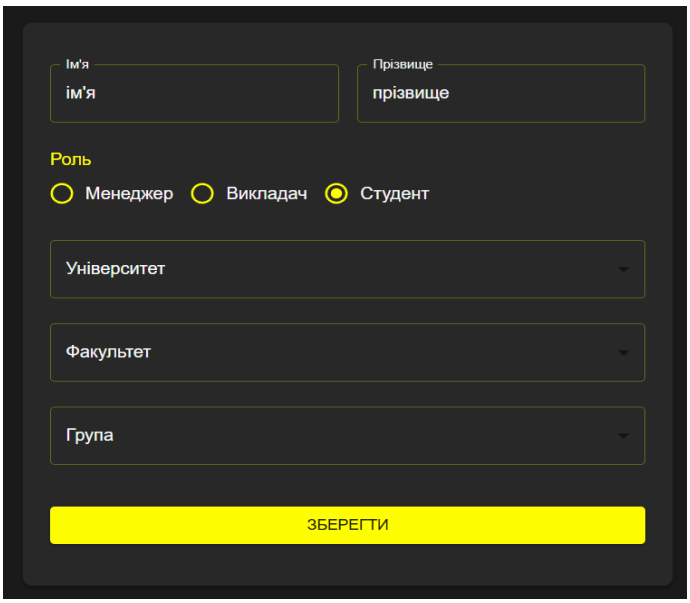


Рисунок. 3.16. Форма Реєстрації другого етапу RegisterForm

**Профіль**

Сторінка профілю користувача (Profile) у системі є важливим компонентом фронтенду, який дозволяє авторизованим користувачам переглядати свої персональні дані, отримані через Auth0, виходити з системи та змінювати мову інтерфейсу.

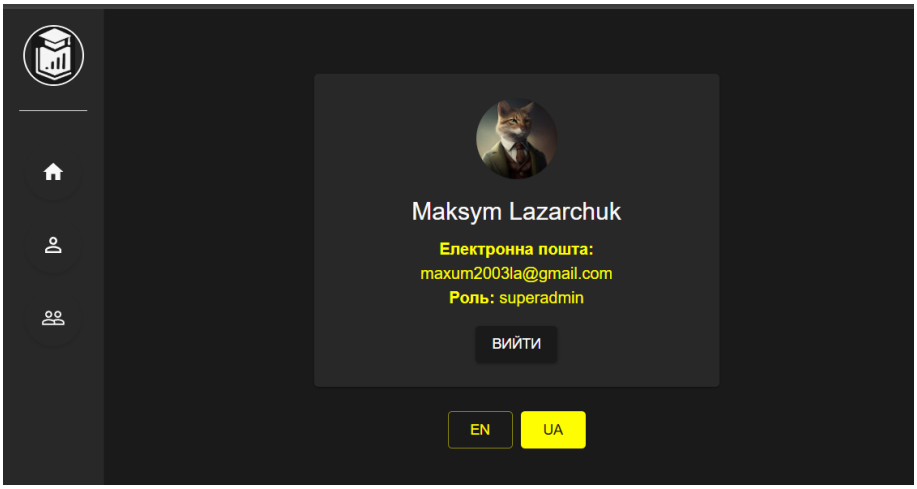


Рисунок. 3.17. Сторінка Профілю

Для забезпечення безпеки та контролю якості даних у системі реалізовано обмеження доступу для непідтверджених користувачів. Користувачі, які завершили реєстрацію через форму RegisterForm, але ще не були схвалені менеджером або супер-адміністратором, мають доступ виключно до сторінки профілю (Profile). Усі інші функції системи, включаючи перегляд розкладів,

оцінок, звітів чи управління користувачами, залишаються недоступними до моменту підтвердження.

**Підтвердження користувачів**

Сторінка підтвердження користувача (UserManage) у системі призначена для менеджерів або супер-адміністраторів, які управляють обліковими записами користувачів. Вона дозволяє переглядати деталі користувача, змінювати його роль та видаляти обліковий запис.

Компонент використовує хуки useParams і useNavigate з react-router-dom для отримання ID користувача з URL та навігації. Для взаємодії з API застосовуються RTK Query хуки (useGetUserByIdQuery, useUpdateUserRoleMutation, useDeleteUserMutation) для отримання даних, оновлення ролі та видалення користувача

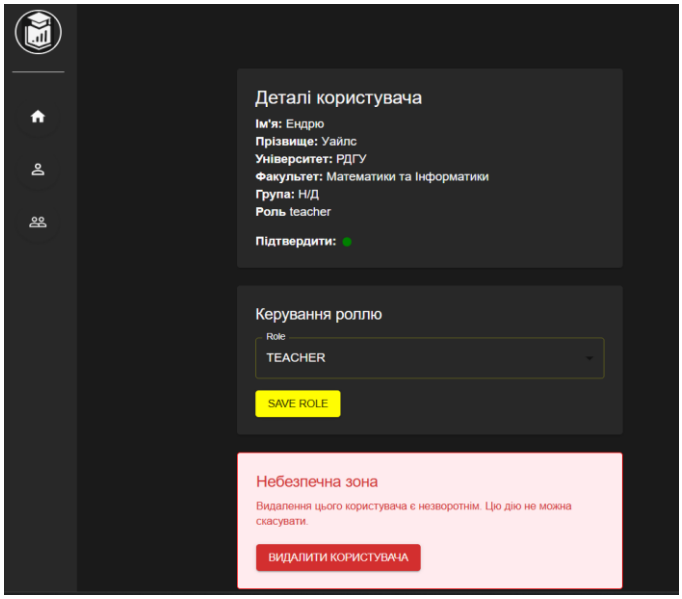


Рисунок. 3.18. Сторінка Підтвердження користувачів UserManage

**Управління групами**

Функціонал для менеджерів включає дві основні можливості:

1. **Створення груп**: Менеджери можуть створювати нові навчальні групи, вказуючи назву. Групи прив’язуються до факультету, в якому знаходиться менеджер.

2. **Перегляд користувачів у групах**: Менеджери можуть переглядати студентів, які належать до певної групи, із деталями, такими як ім’я, прізвище, роль і статус підтвердження.

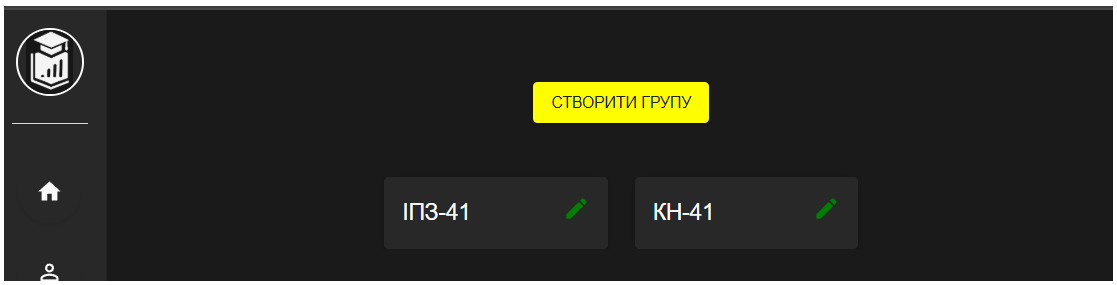


Рис. 3.19. Сторінка Груп

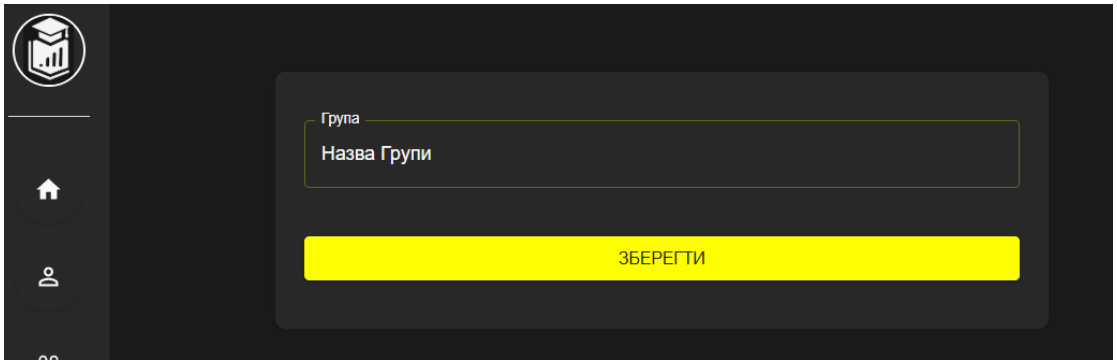


Рисунок. 3.20. Сторінка Створення Груп

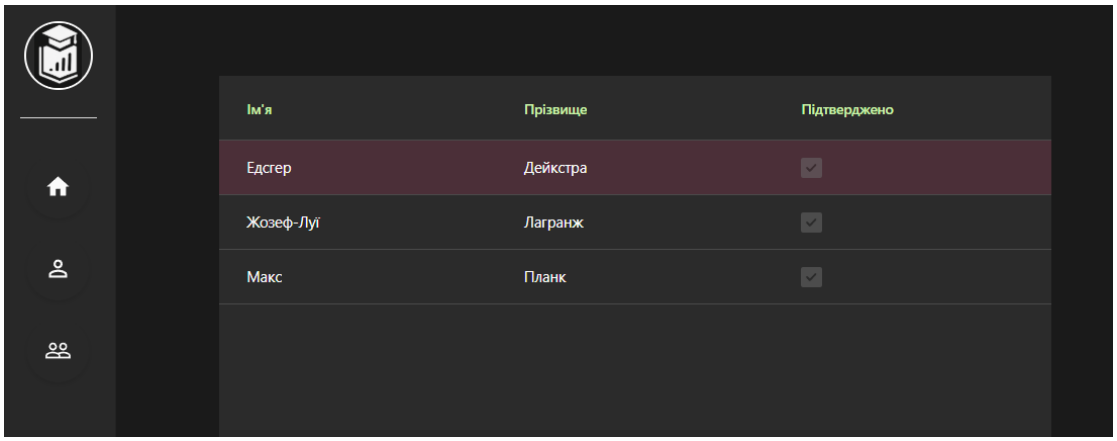


Рисунок. 3.21. Сторінка Перегляду користувачів у групі

Модуль використовує RTK Query (useGetGroupsInMyFacultyQuery, useGetUsersFromGroupQuery, useCreateGroupMutation, useUpdateGroupMutation, useDeleteGroupMutation) для взаємодії з API. Компоненти побудовані з Material-UI та Ag-Grid (AgGridReact) для відображення таблиць із підтримкою клікабельних рядків. Форма створення/оновлення груп (GroupForm) валідується через Zod (GroupValidationType)

**Дисципліни**

Функціонал для сторінки дисциплін включає наступні можливості:

1. **Перегляд списку дисциплін**: Сторінка доступна для всіх ролей, менеджери можуть переглядати всі дисципліни у факультеті, студенти – у своїй

групі, викладачі –дисципліни які викладають. Сторінка містить інформацією про назву предмета, викладача та групу. Список підтримує навігацію до редагування існуючих дисциплін для менеджерів.

2. **Створення нової дисципліни**: Менеджери можуть додавати нові предмети, вказуючи назву, факультет, за яким дисципліна викладається, та викладача, відповідального за курс.

3. **Призначення викладача**: Система дозволяє призначити викладача з будь-якого факультету, що забезпечує гнучкість у розподілі викладацького складу. Ця можливість є важливою для міжфакультетських курсів або залучення спеціалістів із інших підрозділів.

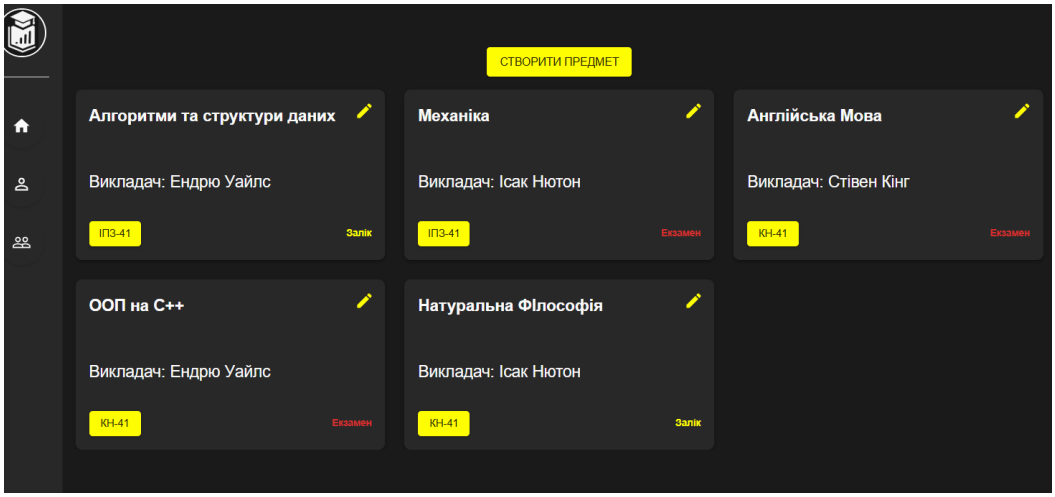


Рисунок. 3.22. Сторінка дисциплін

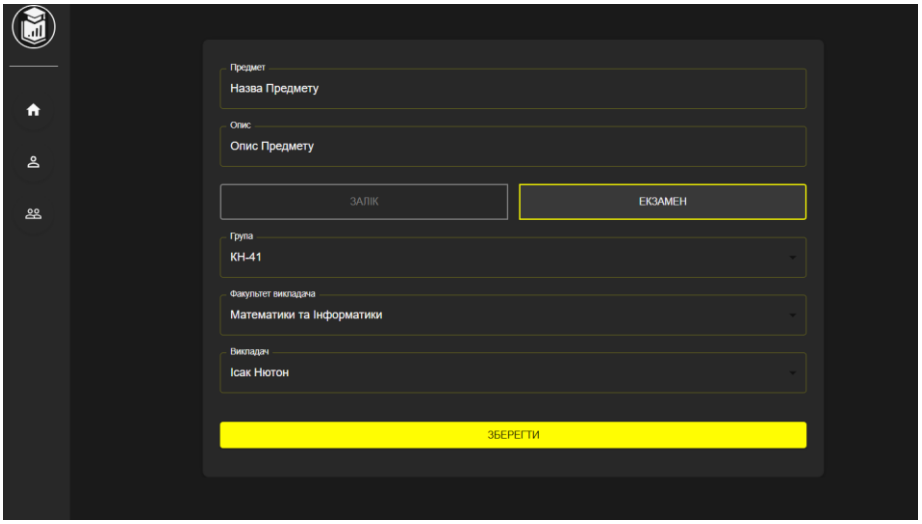


Рисунок. 3.23. Сторінка Створення дисципліни

Цей модуль як і попередній використовує ендпоінти CRUD операцій для моделі Subject. Форма створення/оновлення дисциплін (SubjectForm) валідується через Zod (SubjectValidationType).

**Розклад**

Функціонал створення розкладу включає наступні можливості:

1. **Перегляд тижневого розкладу**: Менеджери можуть переглядати розклад для обраної групи на конкретний тиждень, із відображенням занять по днях (понеділок–неділя). Кожне заняття показує назву дисципліни, час і статус.
2. **Створення нового заняття**: Менеджери можуть додавати нові заняття через модальне вікно, вказуючи дисципліну, групу, дату, час початку та закінчення, а також статус заняття.
3. **Редагування існуючих занять:** При кліку на заняття в тижневому розкладі відкривається модальне вікно для редагування його параметрів
4. **Фільтрація за групами та тижнями**: Менеджери можуть обирати групу зі списку та переглядати розклад для певного тижня, використовуючи календар для навігації між тижнями.

Перегляд розкладу доступний для всіх ролей: менеджери бачать розклади всіх груп, тоді як викладачі та студенти – лише свої.

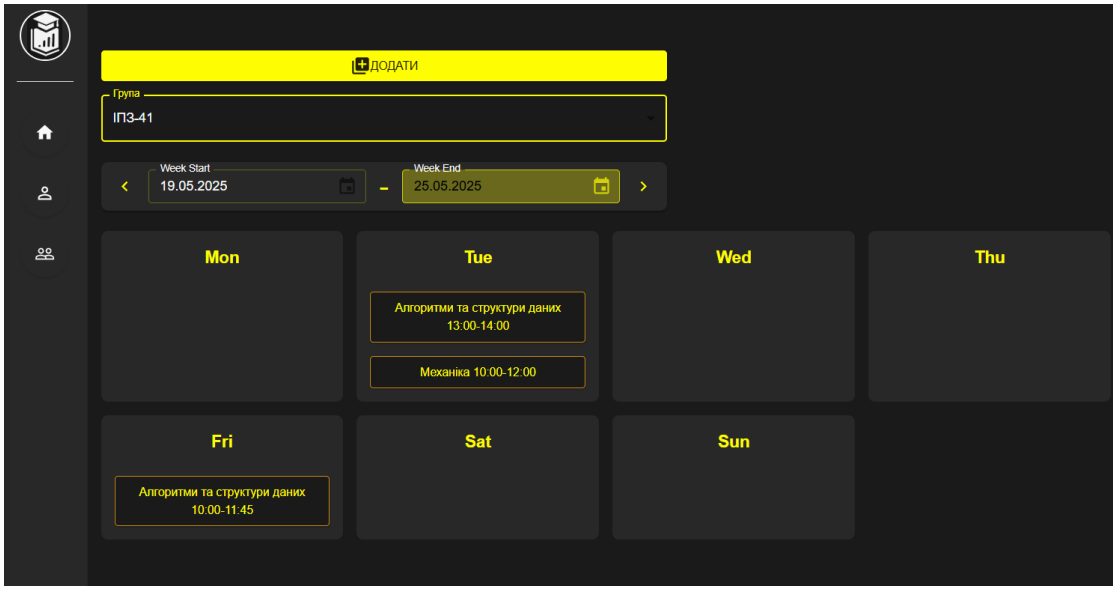


Рисунок. 3.24. Сторінка Перегляду розкладу

Модуль використовує CRUD-операції для моделі SubjectInstance через RTK Query. Головні компоненти модуля WeekView - який відповідає за вигляд тижне-

вого розкладу, та показування занять в конкретні дні, WeekDateRangePicker – що обирає інтервал тижня startDate, endDate, для того щоб правильно надіслати запит на ендпоінти для отримання занять, ScheduleModal – для модального вікна через яке оновлюється, створюється, видаляється заняття.

**Формування звітів**

Функціонал звітів включає наступні ключові можливості:

1. **Перегляд звітів про оцінки**: Менеджери можуть аналізувати оцінки студентів за групою Звіт показує оцінки за дисципліни для всіх студентів у групі.
2. **Перегляд звітів про відвідуваність**: Дозволяє переглядати дані про відвідування занять групою, включаючи кількість пропущених і відвіданих занять.
3. **Індивідуальні звіти**: Менеджери можуть отримати детальну статистику для окремого студента, включаючи його оцінки та відвідуваність за всіма дисциплінами.
4. **Експорт звітів**: Доступний експорт звітів у PDF (для індивідуальних звітів про відвідуваність) і XLSX (для підсумкових звітів про оцінки та відвідуваність), що полегшує обмін даними.

Модуль використовує CRUD-операції для звітів через RTK Query (useGetStudentGroupGradeSummaryQuery, useGetStudentGroupAttendanceSummaryQuery, useGetStudentAttendancesReportByUserIdQuery, useGetStudentGradesReportByUserIdQuery). Форми фільтрації реалізовані через Material-UI (Select, Button) з підтримкою вибору типу звіту, групи та користувача. Експорт у PDF і XLSX забезпечується функціями downloadGroupAttendanceIndividualPdf, downloadGroupAttendanceSummaryXlsx, downloadGroupGradeSummaryXlsx які використовують Fetch API замість RTK Query через потребу в обробці бінарних даних (Blob) для завантаження файлів. Візуалізація даних здійснюється через компоненти SubjectsGradesView, SubjectsGradesTableView, AttendanceReport, GradesView, AttendancesView із таблицями, акордеонами та діаграмами (PieChart, GradeLineChart) за допомогою Material UI.

**Створення завдань та виставлення оцінок**

Функціонал для викладачів включає наступні ключові можливості:

1. **Перегляд завдань**: Викладачі можуть переглядати список завдань для обраної дисципліни, включаючи назву завдання, дедлайн і пов’язану групу студентів.

2. **Створення завдань**: Викладачі можуть створювати нові завдання через модальне вікно, вказуючи назву, опис, дедлайн і дисципліну. Завдання автоматично прив’язуються до обраної дисципліни та групи.

3. **Редагування та видалення завдань**: Викладачі можуть змінювати параметри завдання (наприклад, дедлайн) або видаляти його, якщо воно більше не актуальне.

4. **Оцінка студентських робіт**: Викладачі можуть переглядати подані студентами роботи (наприклад, файли чи текстові відповіді) і виставляти оцінки за кожне завдання або ж просто натиснути на завдання та виставити оцінку якщо подання роботи є необов’язкове.

5. **Виставлення підсумкових оцінок**: Викладачі мають можливість встановлювати підсумкові оцінки за дисципліну для студентів через окремий інтерфейс, доступний за кнопкою “Створити підсумкову оцінку”.

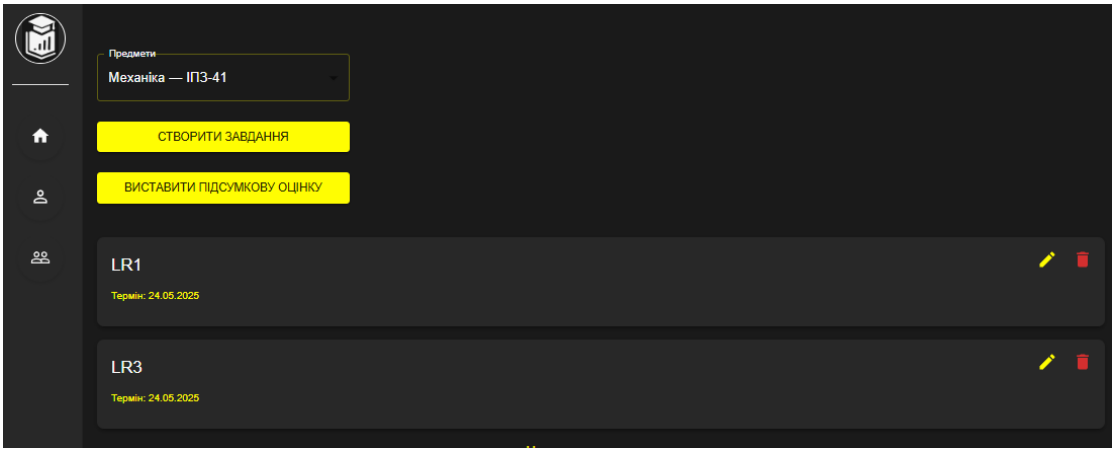


Рисунок. 3.25. Сторінка викладача для завдань


Рисунок. 3.26. Сторінка викладача для виставлення оцінок за конкретне завдання

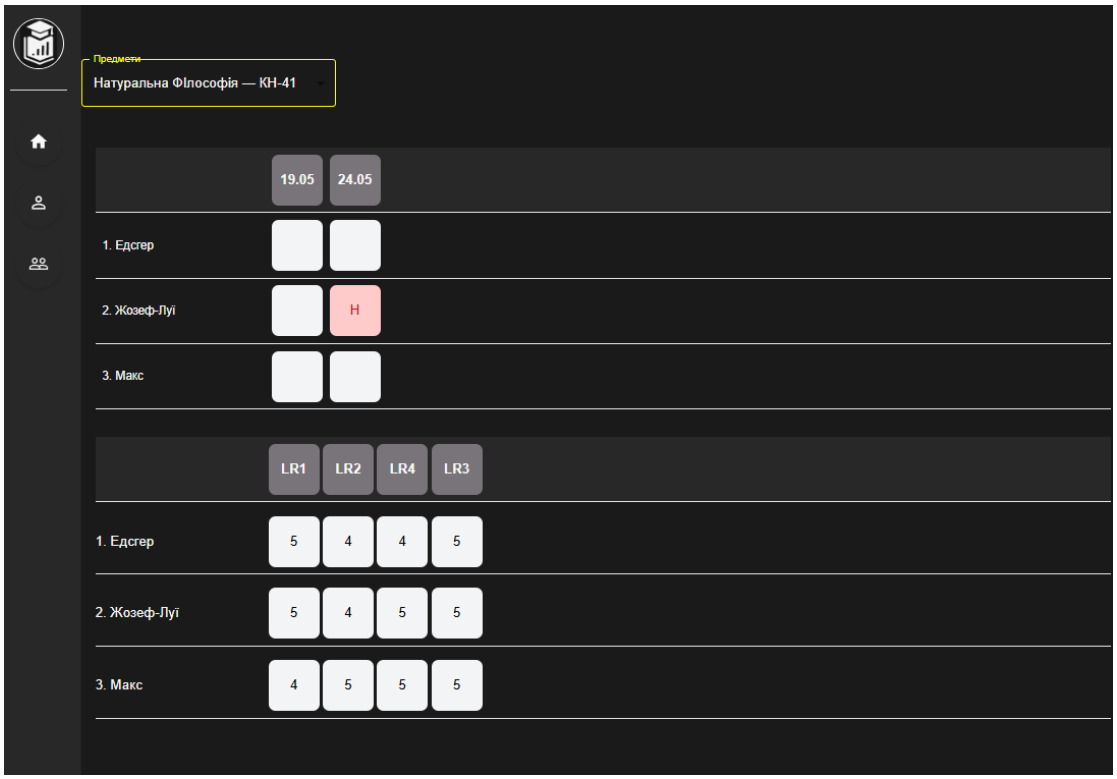


Рисунок. 3.27. Сторінка викладача журнал для загального виставлення оцінок та відвідувань

Сторінка для завдань (TeacherAssignments): Реалізована для перегляду, створення, редагування та видалення завдань викладачем. Дозволяє обирати дисципліну через випадаючий список, відображати список завдань (AssignmentCardList) та поданих студентами робіт (StudentSubmissionList). Нові завдання додаються через модальне вікно (AssignmentModal) з полями для назви, опису, дедлайну та дисципліни.

Журнал (TeacherJournal): Реалізований для відображення оцінок студентів за дисципліну. Включає вибір дисципліни через Select, відображення занять (Jour-

nalView) та оцінок за завдання (AssignmentGradesView). Дозволяє викладачу переглядати відвідуваність і оцінки для студентів обраної групи.

Оцінка завдань (AssignmentEvaluation): Реалізована для перегляду та оцінки поданих студентами робіт. Відображає список студентів із їхніми оцінками в сітці, де клік на оцінку відкриває модальне вікно (GradeModal) для редагування. Опис завдання рендериться через @uiw/react-markdown-preview.

**Подання рішень**

Функціонал подання рішень студентами включає наступні можливості:

1. **Перегляд завдання**: Студенти можуть переглядати деталі завдання, включаючи його опис, який відображається у форматі Markdown для зручного читання.

2. **Подання рішення**: Студенти можуть надіслати рішення через модальне вікно, вказавши посилання на зовнішній ресурс (наприклад, Google Drive) та додавши коментар. Подання є формальним і необов'язковим, що дозволяє студентам вирішувати, чи подавати відповідь.

3. **Гнучкість формату**: Оскільки система не підтримує завантаження файлів, студенти можуть використовувати будь-які зовнішні ресурси для зберігання своїх робіт, надаючи лише URL-адресу та, за потреби, текстове пояснення.

Викладачі зможуть бачити надіслані рішення у себе на сторінці, та оцінити їх

Сторінка перегляду завдання (AssignmentView): Реалізує відображення деталей завдання для студентів, кнопка відкриває модальне вікно (StudentSubmissionModal) для подання рішення з полями для URL і коментарями.

Сторінка подання рішення (Submission): Дозволяє студентам переглядати деталі поданого рішення (статус, коментарі, URL, дата) та, для викладачів, оцінювати роботи. Відображає статус через Chip із кольоровим кодуванням і дозволяє копіювати URL. Викладачі можуть запросити доопрацювання або затвердити роботу через GradeModal для виставлення оцінки.

**Реалізація двомовності інтерфейсу**

Модуль двомовності на фронтенді реалізований за допомогою бібліотеки react-i18next, яка інтегрується з React для підтримки української та англійської мов [21]. Налаштування включає створення файлів перекладів (uk.json, en.json) у папці locales, де зберігаються ключі та відповідні тексти для інтерфейсу. У файлі i18n.ts ініціалізується i18next із ресурсами перекладів

Хук useTranslation використовується в компонентах для доступу до функції t для перекладу текстів. Перемикання мови реалізовано через кнопку в профілі користувача, яка викликає метод i18n.changeLanguage('uk' | 'en').

**Реалізація Progressive Web App (PWA)**

Модуль Progressive Web App (PWA) у системі реалізований для того, щоб користувачі могли працювати з додатком навіть без підключення до інтернету, а також отримувати швидкий доступ за допомогою іконки, яка появляється в разі дозволу завантажити додаток. Для цього використано Vite-плагін @vite-pwa, який інтегрується з React – популярною бібліотекою для створення інтерфейсів.

Для його налаштування потрібно описати manifest - це об’єкт, який містить метадані додатку, які потрібні для його роботи як PWA, та підключається у файл конфігурації Vite.

Тепер коли користувач зайде на вебсайт, через браузер, з’являється сповіщення, яке пропонує встановити додаток на пристрій. Це схоже на встановлення звичайної програми, але не потребує магазину додатків. Після цього PWA стає доступним прямо з робочого столу чи меню телефона, що значно підвищує зручність.

# ВИСНОВКИ

У результаті проведеного дослідження було досягнуто мети –веб-додаток для автоматизації управління освітнім процесом у вищих навчальних закладах. Система забезпечує інтерактивне управління користувачами, групами, дисциплінами, розкладом, завданнями, оцінками та звітами, що сприяє підвищенню ефективності адміністрування та навчання

Розробка веб-додатку охопила створення всіх запланованих модулів. Фронтенд, реалізований на React і TypeScript, забезпечує адаптивний і багатомовний інтерфейс із підтримкою української та англійської мов через react-i18next. Використання RTK Query оптимізувало взаємодію з API, а Material-UI забезпечило зручний і сучасний дизайн. Бекенд, побудований на NestJS, реалізує CRUD-операції та спеціалізовані методи, такі як формування звітів у PDF і XLSX, із захистом через RBAC і Auth0. Інтеграція з Auth0 гарантує безпеку автентифікації та авторизації, а PWA-підхід забезпечує офлайн-доступ і кешування.

Значення отриманих результатів полягає у створенні універсального інструменту, який автоматизує ключові аспекти освітнього процесу, зменшує адміністративне навантаження та підвищує прозорість даних. Система може бути впроваджена у вищих навчальних закладах для управління навчанням, що сприяє цифровізації освіти. Наукова цінність роботи полягає у комплексному підході до проєктування та реалізації веб-додатку, який поєднує сучасні технології та принципи безпеки.

Перспективи подальших досліджень включають:

- Розширення функціоналу, наприклад, додавання можливість для завантажування файлів, для викладачів – навчальні матеріали, для студентів – виконанні роботи
- Додання сповіщень, щоб користувач отримував інформацію про зміну в розкладі, виставлення оцінок, тощо
- Інтеграція із різними сервісами – Google Drive, Google Calendar, Zoom,тощо

- Додання чату або форуму для комунікації

Таким чином, розроблена система є ефективним рішенням для автоматизації освітнього процесу, що відповідає сучасним вимогам цифровізації освіти та відкриває можливості для подальшого розвитку.

## СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ


1. Kleppmann, Martin. *Designing Data-Intensive Applications: The Big Ideas Behind Reliable, Scalable, and Maintainable Systems //*. O'Reilly Media, 2017./ - С. 6 (Дата звернення: 30.09.2024).
2. Frontegg. Role-Based Access Control (RBAC) Guide. URL: https://frontegg.com/guides/rbac (Дата звернення: 01.02.2025).
3. Reactjs – Documentation. URL: https://react.dev/. (Дата звернення: 18.10.2024).
4. NestJS - Documentation. URL: https://docs.nestjs.com/. (Дата звернення: 3.11.2024).
5. Sequelize Documentation. URL: https://sequelize.org/ (Дата звернення: 02.03.2025).
6. Auth0 - Authorization and Authentication Documentation. URL: https://auth0.com/docs (Дата звернення: 20.09.2024).
7. Основні етапи проектування баз даних. URL: https://javarush.com/ua/quests/lectures/ua.questhibernate.level17.lecture01 (Дата звернення: 13.10.2024).
8. IETF. UUID Version 4 Specification. URL: https://www.ietf.org/archive/id/draft-ietf-uuidrev-rfc4122bis-11.html (Дата звернення: 15.01.2025).
9. Mozilla Developer Network (MDN) - Progressive Web Apps (PWA). URL: https://developer.mozilla.org/en-US/docs/Web/Progressive_web_apps. (Дата звернення: 07.10.2024).
10. Vite - Getting Started Guide. URL: https://vite.dev/guide/. (Дата звернення: 18.10.2024).
11. Bacancy Technology - React Architecture Patterns and Best Practices. URL: https://www.bacancytechnology.com/blog/react-architecture-patterns-and-best-practices. (Дата звернення: 05.11.2024).
12. Fowler, M. (2003). Patterns of Enterprise Application Architecture. Addison-Wesley (Дата звернення: 12.04.2025).
13. Repository Pattern. URL: https://blog.mnavarro.dev/the-repository-pattern-done-right (Дата звернення: 13.04.2025).

14. Swagger. Documenting APIs with Swagger. URL: https://swagger.io/resources/articles/documenting-apis-with-swagger/ (Дата звернення: 11.02.2025).
15. Mcuslu. What is an Interceptor in Software Development. URL: https://mcuslu.medium.com/what-is-an-interceptor-in-software-development-012ebba031e3 (Дата звернення: 12.02.2025).
16. GeeksforGeeks. JSON Web Token (JWT). URL: https://www.geeksforgeeks.org/json-web-token-jwt/ (Дата звернення: 13.04.2025).
17. CrowdStrike. Understanding CRUD Operations. URL: https://www.crowdstrike.com/en-us/cybersecurity-101/observability/crud/ (Дата звернення: 27.05.2025).
18. jsPDF Documentation. URL: https://artskydj.github.io/jsPDF/docs/index.html (Дата звернення: 6.05.2025).
19. ExcelJS Documentation. URL: https://github.com/exceljs/exceljs#readme (Дата звернення: 6.05.2025).
20. Zod Documentation. URL: https://zod.dev/ (Дата звернення: 12.04.2025).
21. React-i18next Documentation. URL: https://react.i18next.com/ (Дата звернення: 3.05.2025).