МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ХЕРСОНСЬКИЙ ДЕРЖАВНИЙ УНІВЕРСИТЕТ
Факультет комп'ютерних наук, фізики та математики
Кафедра комп'ютерних наук та програмної інженерії

ПРОГРАМНЕ СЕРЕДОВИЩЕ НАВЧАЛЬНОГО ПРИЗНАЧЕННЯ «ЗАДАЧА ПОБУДОВИ НАЙКОРОТШИХ ШЛЯХІВ У ОРІЄНТОВАНОМУ ГРАФІ, ТА ЇЇ УЗАГАЛЬНЕННЯ ОБМЕЖЕННЯМ РЕСУРСІВ ВИКОНАВЦЯ»

Кваліфікаційна робота (проєкт)
на здобуття ступеня вищої освіти «бакалавр»

Виконав: здобувач
спеціальності: 122 Комп'ютерні науки
Освітньо-професійної програми: Комп'ютерні науки
Нетьосов М. В.
Керівник: д.ф.-м.н. проф. Львов М.С.
Рецензент: Котова О.В., доцент кафедри алгебри, геометрії та математичного аналізу, ХДУ

ЗМІСТ

ПЕРЕЛІК УМОВНИХ СКОРОЧЕНЬ

БД	база даних
JSON	JavaScript Object Notation, текстовий формат даних, який базується на об’єкті JavaScript
NoSQL	non SQL, тобто вид баз даних відмінний від реляційних
SQL	Structured Query Language (мова структурованих запитів) використовується для взаємодії з реляційними базами даних

ВСТУП

Актуальність теми. У зв'язку з розвитком комп'ютерних технологій, великий попит мають веб-сервіси для дистанційного навчання, тому що це дозволяє економити значну кількість часу, скласти зручний для себе графік навчання, а головне – доступність, завдяки можливості перебувати в будь-якій точці світу. Також цікавою та важливою темою для вивчення є графи та алгоритми їх оптимізації, бо значна частина сучасних технологій використовує саме графи для вирішення повсякденних проблем, з якими стикаються люди.

Метою роботи є розробка навчальної платформи, а також початкового набору матеріалів на обрану тему.

Завданнями роботи є:

1. Повна розробка навчальної платформи:
    - проектування бази даних;
    - серверна складова;
    - графічний інтерфейс користувача;
2. Створення набору навчальних матеріалів.

Об’єкт дослідження – процес створення веб-сервісів на основі клієнт-серверної архітектури, а також процес систематизації інформації для педагогічних цілей.

Предмет дослідження – навчальна веб-платформа.

РОЗДІЛ 1
ТЕОРЕТИЧНІ ВІДОМОСТІ ТА ВИБІР ТЕХНОЛОГІЙ

1.1 База даних. PostgreSQL

Для зберігання даних було обрано об'єктно-реляційну базу даних PostgreSQL. Серед причин (переваг) цього вибору варто зазначити наступні:

- Формат зберігання даних. Дані зберігаються у пов’язаних між собою таблицях з фіксованим шаблоном. На відміну від NoSQL баз даних, це дає меншу гнучкість, але знижує ймовірність помилок до мінімуму.
- Транзакції. Це механізм, який дозволяє одночасно вносити декілька змін. У такому разі або всі зміни будуть прийняті, або відхилені, якщо хоч в одному виникне помилка.
- Типи даних. Функціонал PostgreSQL постійно розширюється та оновлюється. На даний момент є підтримка деяких NoSQL можливостей (наприклад JSON, XML) та усіх інших необхідних типів (документи, примітиви, геометрія, структури тощо).
- Пошук інформації. Відповідь на практично будь-яке питання можна знайти в офіційній документації. Крім того, існує безліч освітніх ресурсів як на офіційному сайті так і на інших.

Отже, маємо SQL базу даних з додатковими NoSQL можливостями, зручним пошуком необхідної інформації та деякими перевагами над іншими SQL орієнтованими БД [33, 47].

1.2 Серверна складова

1.2.1 Фреймворк NestJS на базі платформи NodeJS. Як середовище для розробки серверної складової проєкту було обрано технологію NodeJS [4], яка має такі переваги:

- Модель неблокуючого введення-виведення, заснована на подіях. Це дозволяє витрачати менше ресурсів, ніж у моделі паралелізму з використанням потоків операційної системи. А також спрощує розробку та дозволяє обробляти більшу кількість запитів одночасно.
- Єдина мова програмування для серверної та клієнтської розробки, що дозволяє не відволікатися на особливості мови при переході від однієї частини проєкту до іншої.

На базі NodeJS вибрано фреймворк NestJS [16], який побудований з використанням TypeScript [41] - надбудови над JavaScript з використанням типів даних, що дозволяє раніше виявляти або зовсім уникати типових помилок та значно спрощує процес розробки. NestJS поєднує об’єктно-орієнтоване програмування з функціональним та функціонально-реактивним [3] (підхід з асинхронними запитами та реагуванням на певні дії), за замовчуванням використовує надійний мінімалістичний фреймворк Express [18] (також є можливість заміни на Fastify [19]), додає свій рівень абстракції над цими платформами з можливістю взаємодіяти з базовим функціоналом. Серед можливостей, які надає NestJS, варто зазначити такі:

- Контролери - механізм маршрутизації, який відповідає за обробку запитів та повернення результатів [9].
- Провайдери - будь-яка логіка програми (сервіси, репозиторії для роботи з базою даних, фабрики тощо). Один провайдер (екземпляр) можна використовувати у різних частинах програми за допомогою нативної підтримки шаблону проектування dependency injection

[13]. Для цього потрібно лише вказати тип провайдера в конструкторі іншого провайдера і NestJS автоматично зробить решту роботи [34].

- Модулі - сукупність контролера, провайдерів та інших складових, що відповідають за окрему частину програми. Модулі ізольовані один від одного і за потреби можуть експортувати власні провайдери або імпортувати чужі. Цей підхід дозволяє розробляти велики та легко масштабовані додатки [29].
- Проміжне програмне забезпечення (middleware) - функція, що викликається перед обробкою маршруту і має доступ до об’єктів запиту (request) та відповіді (response) [28].
- Фільтри винятків. За замовчуванням, NestJS автоматично обробляє всі необроблені винятки за допомогою вбудованого рівня помилок. Фільтри винятків дозволяються розробити та використовувати власні типи помилок для обробки конкретної поставленої задачі [17].
- Труби (Pipes). Отримують доступ до вхідних даних і мають два основні завдання: перетворення типів даних (наприклад, рядок у число) або перевірка дійсності даних. У другому випадку, якщо дані відповідають вимогам, то проходять далі, інакше програма генерує помилку та припиняє обробку поточного запиту [32].
- Охоронці (Guards). Єдиним завданням є визначення, чи буде поточний запит оброблений чи ні. Типовим прикладом є охоронці, які перевіряють чи авторизований користувач або чи має він необхідну роль для доступу до певного ресурсу. Як і другий варіант застосування pipes, або пропускає запит далі, або генерує помилку та припиняє обробку запиту [23].
- Перехоплювачі (interceptors) можуть використовуватися як до так і після обробки запиту. Дозволяють додати логіку, замінити

результат або виключення, розширити або повністю перевизначити поведінку обробки запиту [25].

- CLI дозволяє швидко створювати та модифікувати як окремі елементи програми, так і цілі модулі з готовими шаблонами під конкретні поставлені завдання та набором тестів [31].

Загалом, процес обробки запиту виглядає таким чином:

1. Користувач надсилає запит.
2. У порядку застосування у коді, виконуються middlewares, pipes, guards та interceptors.
3. Контролер отримує потрібні дані із запиту та відправляє їх на обробку до сервісу.
4. Сервіс обробляє дані та повертає результат до контролера.
5. Контролер відправляє отриманий результат клієнту.

Якщо під час цього ланцюжка виникає помилка, то вона обробляється створеним обробником виключень якщо вдалося її виявити та обробити, інакше вбудованим глобальним фільтром. Завдяки цьому сервер не перестає працювати після неочікуваних помилок.

1.2.2 Робота з базою даних. Sequelize. Для підключення та взаємодії з базою даних використовується ORM [46] (Object Relational Mapping) - “міст” між БД та програмою, який дозволяє у звичному синтаксисі мови програмування керувати вмістом бази даних. А саме реалізація Sequelize [37] для роботи з TypeScript [38]. Sequelize дозволяє як заповнити нову базу даних потрібними таблицями, так і підключитися до бази з існуючими таблицями та продовжити розробку. Для опису таблиць використовуються класи, властивостями яких є поля таблиці. За допомогою декораторів, полям додаються певні параметри (ключі, тип даних, значення за замовчуванням тощо).

1.2.3 Інші поняття та техніки. DTO (data transfer object) - об’єкт що описує дані, які передаються по мережі. Часто використовується для перевірки коректності вхідних даних або отримання лише потрібної інформації для конкретного завдання [12].

Хешування – односторонній процес перетворення даних з метою їхнього захисту. Відновити дані, знаючи хеш, неможливо. Можна лише повторно хешувати та порівняти між собою хеші (у такому випадку вони мають бути однаковими). Часто використовується для зберігання паролів у базі даних [2].

JWT (JSON Web Token) - стандарт для передачі зашифрованої інформації у вигляді об’єкта JSON. JWT складається з трьох частин, які розділені крапками:

- Заголовок (header). Містить загальну інформацію про токен: тип токена та алгоритм підпису. Кодується у формат Base64Url.
- Корисне навантаження (payload). Будь-яка інформація для передачі. Кодується у формат Base64Url.
- Підпис (signature). Використовується для перевірки оригінальності переданої інформації та розшифровки самої інформації за допомогою закритого ключа. Отримується шляхом використання алгоритму хешування, до якого передається header, payload та закритий ключ (secret).

Найчастіше JWT використовують для авторизації користувача [1, 27].

1.3 Клієнтська складова

1.3.1 Бібліотека React. React - це найпоширеніша бібліотека для створення веб-інтерфейсів користувача. Для написання коду використовується синтаксис JSX [26] - розширення JavaScript для поєднання з HTML. Для оптимізації React

використовує віртуальне DOM-дерево, накопичує необхідні зміни і потім сам визначає коли треба оновити оригінальне DOM-дерево браузера [36]. Додаток на React складається з компонентів - своєрідних функцій, які приймають набір параметрів на повертають елементи, з яких будується кінцева розмітка веб-сторінки [8]. Серед найчастіше використовуваних можливостей React слід виділити такі:

- useState - хук для додавання до компоненту деякого стану, при зміні якого автоматично оновлюється частина компоненту, яка залежить від цього стану [45].
- useContext - хук схожий на useState, але використовується для зберігання не локального стану окремого компонента, а глобально для всієї програми [42].
- useEffect - хук для роботи з побічними ефектами, тобто діями, зона відповідальності яких виходить за рамки компонента (наприклад, отримання даних з сервера, підписка на події, оновлення, очищення тощо) [43].
- Router - підтримує маршрутизацію на стороні користувача. Це дозволяє заощадити значну кількість часу завдяки тому, що не потрібно щоразу при оновленні сторінки або переході на іншу сторінку, завантажувати нову HTML розмітку з CSS стилями та кодом JavaScript, які браузер повинен проаналізувати [20].
- useNavigate - хук для переходу на іншу сторінку за допомогою Router [44].

1.3.2 Менеджер стану. MobX. Для роботи з глобальним станом (сховищем даних) програми використовуватиметься бібліотека MobX, яка дозволяє писати мінімалістичний код без використання шаблонів (тобто у зручному форматі) та автоматично виявляє внесені зміни з подальшим їх поширенням у необхідні місця. MobX оптимізує обчислення, гарантуючи що вони будуть виконані тільки тоді, коли необхідні, що дозволяє уникнути ручних оптимізацій, які

схильні до помилок. Відмінною особливістю від інших менеджерів станів є простота та прозорість функціоналу [35].

1.3.3 Стилізація. Bootstrap. Щоб власноруч не розробляти з нуля дизайн сайту, а також його адаптацію до пристроїв з різним розміром екрану, було використано бібліотеку `bootstrap-react`, яка побудована на базі фреймворку Bootstrap з великим набором готових компонентів та стилів. Також, за замовчуванням підтримуються спеціальні можливості (наприклад, управління за допомогою клавіатури та програми для зчитування інформації з екрана), що дозволяє користуватися веб-додатком більшій кількості користувачів [6].

РОЗДІЛ 2
ОПИС РЕАЛІЗАЦІЇ ПРОГРАМНОГО ПРОДУКТУ

2.1 Серверна складова

2.1.1 Налаштування. Визначимо базові змінні оточення для роботи сервера за допомогою `.env-development` та `.env-production` файлів у корені проєкту:

- PORT - ціле число, відповідає за номер порту, на якому працюватиме сервер;
- POSTGRES_HOST - адреса, на якій працює база даних;
- POSTGRES_PORT - номер порту, на якому працює БД;
- POSTGRES_USERNAME - логін власника БД;
- POSTGRES_PASSWORD - пароль власника БД;
- POSTGRES_DB - назва БД;
- PRIVATE_KEY - закритий ключ, необхідний для JWT;
- STORAGE_PATH - шлях для зберігання файлів, які будуть завантажені на сервер;
- MAIN_ADMIN_EMAIL – пошта головного адміністратора;
- MAIN_ADMIN_PASSWORD – пароль головного адміністратора.

Пошта та пароль головного адміністратора використовуються лише при першому запуску сервера, потім їх краще видалити.

Встановимо модуль `cross-env` [11], який дозволяє при запуску сервера визначити додаткові параметри оточення. У файлі `package.json` зробимо команди для запуску у режимах розробки та робочої версії:

- “start”: "cross-env NODE_ENV=production nest start"
- "start:dev": "cross-env NODE_ENV=development nest start --watch"

Далі у файлі `app.module.ts` до списку імпортів (imports) додаємо модуль `ConfigModule` з бібліотеки `@nestjs/config`, передаючи до нього назву файлу з

параметрами оточення, яка визначається динамічно за допомогою раніше визначеної змінної `NODE_ENV`. Пізніше до цього списку імпортів потрібно буде додати кожен створений модуль.

Переходимо до “точки входу” до програми - файлу `main.ts`. Додаємо наступні вирази:

- const PORT = process.env.PORT || 5000. Отримуємо номер порту для роботи сервера із змінних оточення. Якщо не вдалося отримати - визначаємо значення за замовчуванням 5000;
- app.enableCors(). Для можливості робити запити до сервера з інших доменів [10];
- app.setGlobalPrefix(‘/api’). Визначаємо глобальний префікс для всіх запитів.

Та змінюємо число на визначену константу `PORT` у функції `app.listen`.

2.1.2 Проектування бази даних. Sequelize. Основними складовими освітньої платформи є користувачі та дисципліни. Кожна дисципліна має кілька модулів, а кожен модуль має кілька навчальних матеріалів – документів. Також є декілька типів користувачів: учні, вчителі та адміністратори. Тому додаємо таблицю ролей. Останньою складовою буде сутність "групи" для об'єднання кількох учнів, вчителів та дисциплін. У результаті отримуємо схему бази даних наведеної на рис. 2.1.

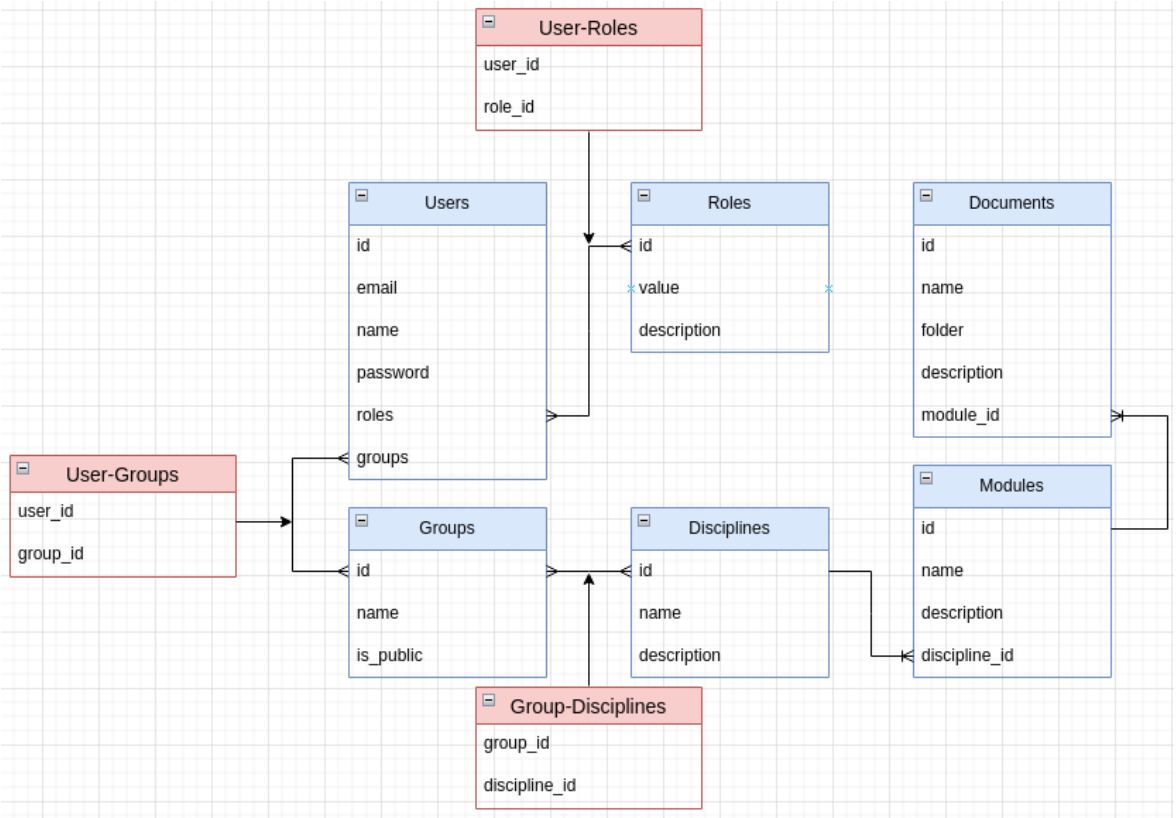

Рисунок 2.1 – Схема бази даних

Блакитним кольором виділено основні таблиці, червоним - додаткові, необхідні для реалізації зв'язку багато-до-багатьох.

Для початку роботи з Sequelize треба встановити пакети `sequelize` і `sequelize-typescript`, а також `pg` і `pg-hstore` для роботи з БД PostgreSQL. Далі у файлі `app.module.ts` до списку імпортів додаємо модуль `SequelizeModule` з пакету `@nestjs/sequelize` та викликаємо метод `forRoot`, передаючи до нього наступні параметри:

- dialect: ‘postgres’. Так як Sequelize підтримує різні популярні БД, потрібно вибрати з якою саме будемо працювати.
- `host`, `port`, `username`, `password` і `database` отримуємо із змінних оточення за допомогою `process.env.VARIABLE`. Варто зазначити, що значення змінних зберігається у вигляді рядка, тому номер порту необхідно перетворити до числового вигляду, інакше виникне помилка.
- масив `models`. Сюди потрібно буде додавати всі створені моделі.
- autoLoadModels: true. Дозволяє автоматично створити відсутні таблиці в БД.

Для кожної основної сутності (таблиць позначених блакитним кольором) створюємо окремий модуль та моделі відповідних таблиць (додаткові таблиці для зв’язку багато-до-багатьох додаємо лише до одного з відповідних модулей). Модель являє собою клас, успадкований від `Model` з пакету `sequelize-typescript`, позначений декоратором `@Table` з того самого пакета. Кожна властивість позначається декоратором `@Column`, до якого передаються параметри відповідного стовпця таблиці. За допомогою декораторів: `@HasMany`, `@HasOne`, `@BelongsTo`, `@BelongsToMany` та `@ForeignKey`, позначаються поля, які відповідають за зв’язок між таблицями [38, 39].

2.1.3 Сервіси. Сервіс у NestJS - це провайдер, який являє собою клас позначений декоратором `@Injectable` та реалізує певну логіку роботи з даними [34]. Розроблені сервіси з набором методів наведено у табл. 2.1.

Таблиця 2.1 – Розроблені сервіси

Модуль	**Методи**
Users	createUser, getUserById, getAllUsers, addRole, removeRole, addGroup, removeGroup
Roles	getRoleByValue, getAllRoles
Groups	createGroup, getGroupById, getAllGroups, updateGroup, deleteGroup, addDiscipline, removeDiscipline
Disciplines	createDiscipline, getDisciplineById, getAllDisciplines, updateDiscipline, deleteDiscipline
Modules	createModule, getModuleById, getAllModules, updateModule, deleteModule
Documents	createDocument, getDocumentById, deleteDocument
Додаткові модулі	
Auth	registration, login, generateToken, validateUser
Files	saveFile, getFile

Для отримання доступу із сервісу до методів моделі для роботи з базою даних потрібно:

- додати модель до масиву `models` у файлі `app.module.ts`;

- додати модель до списку імпортів поточного модуля за допомогою метода `forFeature` з `SequelizeModule`;
- за допомогою ін'єкції залежностей (dependency injection) та декоратора `@InjectModel` додати модель через конструктор сервісу.

Для отримання доступу із сервісу до методів іншого сервісу потрібно:

- додати необхідний сервіс до списку експортів (exports) відповідного модуля;
- додати необхідний модуль до списку імпортів (imports) поточного модуля;
- за допомогою ін'єкції залежностей додати необхідний сервіс через конструктор поточного сервісу.

Під час розробки Auth сервісу для реєстрації та аутентифікація користувачів слід звернути увагу на:

- Щоб захистити особисті дані користувачів від злому бази даних, паролі повинні зберігатися в хешованому вигляді. Для цього використовуються бібліотеки `bcrypt` або `bcryptjs` [2, 5].
- Під час реєстрації або входу в обліковий запис, користувачеві слід повертати jwt токен в якому зберігається лише потрібна про нього інформація для роботи сайту (наприклад, ім'я, список ролей та груп). При цьому, конфіденційну інформацію (наприклад, пароль чи адресу) не можна передавати, оскільки підпис ключа необхідний лише для ідентифікації його справжності, але не захищає від зчитування інформації [27].
- Для захисту від крадіжки токенів слід використовувати пару токенів: access token з невеликим часом дії для доступу до ресурсів та refresh token з великим часом дії для оновлення поточної пари токенів. Більш детально з цією методикою можна ознайомитись у статті [1].

2.1.4 Контролери. Контролер у NestJS – це клас позначений декоратором `@Controller()`. У конструкторі обов'язково необхідно за допомогою dependency injection додати відповідний сервіс. Над кожним методом класу додається декоратор, що відповідає необхідному методу HTTP (`@Get()`, `@Post()`, `@Put()`, `@Delete()`, `@Patch()`, `@Options()` і `@Head()`, а також `@All()` для обробки будь-якого типу запиту). Усі вище перелічені декоратори приймають необов'язковий параметр - частина адреси ресурсу. Таким чином, остаточна адреса, що обробляється конкретним методом контролера, складається з:

- адреси комп'ютера, на якому запускається сервер;
- глобальний префікс, визначений за допомогою `setGlobalPrefix` у файлі `main.ts`;
- параметра, що передається до декоратора `@Controller()`;
- параметра, що передається до декоратора метода HTTP.

До параметрів застосовуються такі декоратори:

- для отримання інформації із запиту: `@Body()`, `@Query()`, `@Param()`, `@Session()`, `@Headers()`, `@Ip()`, `@HostParam()`;
- для отримання доступу до об’єктів з базових платформ (Express, Fastify): `@Request()` або `@Req()`, `@Response()` або `@Res()`, `@Next()`.

Варто звернути увагу, що при використанні `@Res` або `@Next` відключається стандартний механізм NestJS і потрібно власноруч продовжувати виконання запиту. Якщо необхідно внести маленькі доповнення (наприклад, встановити файли cookie), то слід встановити параметр `passthrough: true` в даному декораторі [9].

Щоб точно знати яким має бути тіло (body) конкретного запиту та виключити обробку некоректних даних, для цього запиту розробляється DTO - клас, який містить необхідні властивості з описом типу даних та набором декораторів з пакету `class-validator` [7] для більш чіткої валідації даних (наприклад, `@IsEmail()`, `@Length()` та інші). Якщо тіло запиту не відповідає певному DTO, NestJS припиняє виконання запиту і повертає помилку на клієнт.

Для обмеження доступу до деяких ресурсів сервера було розроблено таких охоронців (Guard):

- JwtAuthGuard. З об'єкта запиту намагається отримати токен авторизації. Якщо токена немає або він недійсний, доступ заборонений і NestJS повертає помилку на клієнт.
- RolesGuard. Перевіряє, чи має користувач необхідну роль для доступу до певного ресурсу. Для роботи цього охоронця необхідно додатково створити декоратор `@Roles`, за допомогою якого визначатиметься список ролей, яким дозволено доступ до певного ресурсу.

Розроблених охоронців можна застосовувати або глобально, або до певного контролера чи методу [23].

2.1.5 Документація. Swagger. Для початку роботи з Swagger потрібно встановити пакет `@nestjs/swagger`. Далі у файлі `main.ts` створити конфіг за допомогою класу `DocumentBuilder` до якого додати назву, опис, версію та іншу інформацію про сервер. На основі конфігу та екземпляра програми (app) створюється документ і додається адреса, за якою буде доступна документація. Swagger автоматично додає всі маршрути, що обробляються контролерами, в документацію, але для її повного заповнення необхідно над кожним контролером, методом контролера, властивостями моделі і DTO додати відповідні декоратори з описом [30]. Приклади документації наведено у додатку А.

2.2 Клієнтська складова

2.2.1 Зв’язок із сервером. Axios. Як HTTP-клієнт було обрано Axios, за допомогою якого можна створювати окремі клієнти-екземпляри для надсилання запитів зі своїми конфігами. Це дозволяє уникати повторів даних, характерних для певної групи запитів. Створимо два екземпляри: для авторизованих та неавторизованих користувачів. Конфіги обох екземплярів міститимуть адресу сервера (baseURL). Додатково конфіг для авторизованих користувачів міститиме властивість `withCredentials: true` для автоматичної передачі Cookies на сервер.

Також, Axios може перехоплювати запити і відповіді. Використовуємо це для автоматичного додавання jwt токена до кожного запиту, який відправляється за допомогою екземпляра axios для авторизованого користувача. Для цього створюємо функцію-перехоплювач, яка отримує токен з локального сховища та поміщає його у заголовок запиту.

Якщо використовується стратегія з двома токенами (access і refresh), то необхідно додатково створити перехоплювач для відповідей сервера, який виконує такі кроки:

- надсилає запит на сервер;
- якщо отримує відповідь без помилок, то нічого не робить;
- якщо є помилка, статус код якої 401 (Unauthorized), то надсилається запит на сервер з refresh токеном для оновлення пари токенів;
- повторює оригінальний запит, використовуючи новий токен доступу.

Варто звернути увагу, якщо буде отримана помилка при спробі оновити токени, то перехоплювач відповіді буде нескінченно намагатися їх оновити. Щоб це виправити, потрібно до конфігу запиту додати властивість, яка відповідатиме за те, чи повторюється запит і зробити додаткову перевірку. Також може

виникнути помилка, код якої не дорівнює 401. Цей випадок також бажано обробити [22, 24, 40].

2.2.2 Менеджер стану. MobX. Для початку роботи з MobX потрібно встановити бібліотеки `mobx` та `mobx-react-lite`. Далі створити клас, який відповідає за стан програми та почати за ним спостерігати за допомогою однієї з двох функцій: `makeObservable` для ручного налаштування параметрів або `makeAutoObservable` для автоматичного налаштування [35].

Для управління станом та отримання його значень створимо відповідні методи (actions) та гетери. Особливістю MobX є можливість змінювати стан шляхом зміни властивостей об'єкта чи елементів масиву, тобто не потрібно робити копію об'єкта для оновлення стану. Взаємодія зі станом здійснюється таким чином:

- у головному файлі програми (`index.jsx`) за допомогою функції createContext з бібліотеки `react` створюється глобальний контекст (`Context`) і за допомогою тега `Context.Provider` передається екземпляр класу стану;
- для доступу до стану з будь-якого компонента використовується хук `useContext`;
- для автоматичного оновлення необхідного компонента при зміні стану використовується функція `observer` з бібліотеки `mobx-react-lite`.

2.2.3 Маршрутизація. Створимо файл `routes.js` у якому визначимо три масиви (publicRoutes, authRoutes та adminRoutes) для зберігання доступних маршрутів за якими можуть переходити всі користувачі, тільки авторизовані та тільки адміністратори відповідно. Кожен елемент масиву являє собою об'єкт із двома властивостями:

- path - відносна адреса;
- Component - відповідний до цієї адреси компонент.

Далі додаємо компонент `AppRouter`, у якому створюємо маршрути для publicRoutes. Залежно від того, чи користувач авторизований і чи має права адміністратора, створюємо маршрути для authRoutes і adminRoutes відповідно. Головний компонент (`App`) виглядає так:
`<BrowserRouter><AppRouter /></BrowserRouter>`. Також, всередині `BrowserRouter` можна додати інші елементи веб-сторінки (наприклад, navbar або footer). Наприкінці додаємо маршрут, який відповідає за будь-яку іншу адресу (яка не існує) та робить переадресацію на головну сторінку [20, 44].

2.2.4 Компоненти. Візуальна частина. Умовно, розроблені компоненти (елементи веб-сторінки) можна поділити на 3 групи:

1. самостійна функціональна частина (наприклад, NavBar - шапка сайту чи DocumentList - блок, відповідальний за відображення та взаємодію з документами);
2. модальне вікно. Відображається посередині екрана, затемнюючи та блокуючи доступ до інших елементів веб-сторінки. Використовується для додавання функціоналу на поточну сторінку, не займаючи зайвого місця (тобто показується за необхідності, інакше ховається);
3. цілі веб-сторінки. Відповідають за відображення основної частини веб-сторінки (без шапки та футера). Складаються з перших двох груп із необхідними доповненнями.

Для оновлення контенту веб-сторінки під час оновлення відповідних даних, необхідно:

- дані, з яких генеруються елементи в браузері (наприклад, список користувачів або текст, що відображається в полі введення), зберігати та змінювати за допомогою стану (хук `useState`). У такому випадку, якщо змінитися список користувачів, то React автоматично зробить оновлення потрібної частини сторінки [45];
- якщо над даними потрібно зробити додаткові дії (наприклад, впорядкувати), то додається хук `useEffect`, у списку залежностей якого

міститься стан із даними. Дані обробляються і передаються в інший стан (наприклад, sortedUsers), з якого генерується контент для відображення. У цьому випадку важливо використовувати саме різні стани, інакше `useEffect` буде спрацьовувати під час зміни стану, який він сам же і змінює (тобто нескінченні оновлення) [43].

Для відображення інформації, яку необхідно отримати з сервера, використовується такий алгоритм:

- у потрібному компоненті створюються локальні стани за допомогою хука `useState`, які зберігатимуть необхідні дані та поточний стан завантаження (булеве значення, true - відбувається завантаження, false - завантаження завершено);
- за допомогою хука `useEffect` (з порожнім масивом залежностей) відправляється запит на сервер. Після отримання даних оновлюються необхідні стани (зберігаються отримані значення та припиняється завантаження);
- залежно від стану завантаження, компонент або показує Spinner (анімація завантаження), або відображає необхідний контент;
- якщо необхідно зробити запит до сервера під час певних дій користувача на самій сторінці (тобто не в момент її показу, а, наприклад, у разі натискання на кнопку), то потрібно додати відповідний `useEffect`, у списку залежностей якого будуть стани (локальні або глобальні), що змінюються під час дій користувача.

Варто звернути увагу, що при роботі з `useState` і використанні об'єктів (або масивів), для оновлення значення стану, необхідно встановити як нове значення інший об'єкт (інше посилання). Цього можна досягти шляхом копіювання значень попереднього об'єкта. У випадку з MobX цього робити не треба.

Логіка роботи з модальними вікнами виглядає так:

- модальне вікно завжди знаходиться всередині необхідного компонента, але має властивість (props), що відповідає за відображення [8];
- для керування значенням властивості відображення модального вікна у

компоненті створюється додатковий стан (наприклад, modalVisible) аналогічний до завантаження;

- додається обробник події (наприклад, `onClick` для натискання кнопки), в результаті якої відображається модальне вікно (змінюється значення modalVisible на true);
- визначаються і передаються в якості властивостей необхідні дані та функції для роботи модального вікна, серед яких функція для його закриття (зміна значення modalVisible на false).

Також була використана бібліотека `react-contexify` [21] для створення спливаючих меню, логіка роботи з якими дуже схожа на модальні вікна.

Використовуючи описані вище методики та розроблений сервер, було створено графічний інтерфейс з можливостями представленими у табл. 2.2.

Таблиця 2.2 – Можливості графічного інтерфейсу

Роль користувача	**Можливості**
неавторизований користувач	● перегляд головної сторінки з новинами або іншою інформацією про веб-сайт; ● авторизація або реєстрація.
USER (авторизований користувач)	● можливість самостійно знайти та записатися у відкриту групу; ● перегляд загальної інформації та доступ до відповідних ресурсів груп - дисциплін - модулів.
TEACHER (викладач)	● попередні можливості; ● управління навчальним матеріалом модулів; ● можливість додавати та видаляти учасників з групи.
ADMIN (адміністратор)	● попередні можливості; ● адмін панель з такими розділами: 1. користувачі: ● перегляд списку всіх користувачів сайту; ● можливість видавати/забирати роль викладача; ● редагування списку груп користувача; 2. групи: ● перегляд списку всіх груп; ● створення/видалення групи; ● зміна налаштувань групи; ● редагування списку дисциплін для вивчення;

Продовження табл. 2.2

	3. дисципліни: ● перегляд списку всіх дисциплін; ● створення/видалення дисципліни; ● зміна налаштувань дисципліни; 4. модулі: ● перегляд списку модулів за обраною дисципліною; ● додавання/видалення модуля; ● зміна налаштувань модуля.
MAIN_ADMIN (головний адміністратор)	● усі попередні можливості; ● можливість видавати/забирати роль адміністратора.

Приклади реалізації графічного інтерфейсу наведено у додатку Б.

2.3 Запуск проєкту та базовий контент

Відповідно до теми «Задача побудови найкоротших шляхів у орієнтованому графі, та її узагальнення обмеженням ресурсів виконавця», було створено набір лекцій та реалізацій алгоритмів, а також скрипт для автоматичного завантаження матеріалів на сайт.

За допомогою Docker [15] та Docker-compose [14] налаштовано роботу додатка всередині контейнерів, що дозволяє запускати проєкт у різних операційних системах без необхідності встановлювати використовувані технології та гарантує коректну роботу.

ВИСНОВКИ

У ході роботи над проєктом було проаналізовано існуючі технології та методики розробки, що дозволяють створити навчальну веб-платформу.

Використовуючи PostgreSQL – надійну реляційну базу даних з великою кількістю додаткових можливостей та NestJS – прогресивний фреймворк для створення ефективних, масштабованих серверних додатків на основі платформи NodeJS, було розроблено серверну частину проєкту зі значним поцентіалом для розширення функціональності. За допомогою Swagger створено зручну документацію для взаємодії із сервером.

Використовуючи React - бібліотеку для створення складних інтерфейсів з високою продуктивністю і Bootstrap - фреймворк для розробки адаптивних веб-додатків, було розроблено клієнтську частину проєкту для комфортного навчання на різних пристроях.

Систематизувавши інформацію за темою «Задача побудови найкоротших шляхів у орієнтованому графі, та її узагальнення обмеженням ресурсів виконавця», було створено набір лекцій та реалізації відповідних алгоритмів для додавання матеріалу на навчальну платформу.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Про токены, JSON Web Tokens (JWT), аутентификацию и авторизацию. Token-Based Authentication. URL: https://gist.github.com/staslysak/07cff1fe0d2c3ffa0da7e7277a7b1d5a
2. Управление паролями. Разбираемся, как правильно шифровать… | by Bohdan Balov | Medium. URL: https://medium.com/@balovbohdan/управление-паролями-82d99005207
3. A quick introduction to Functional Reactive Programming (FRP). URL: https://www.freecodecamp.org/news/functional-reactive-programming-frp-imperative-vs-declarative-vs-reactive-style-84878272c77f/
4. About | Node.js. URL: https://nodejs.org/en/about
5. bcryptjs - npm. URL: https://www.npmjs.com/package/bcryptjs
6. Bootstrap · The most popular HTML, CSS, and JS library in the world. URL: https://getbootstrap.com/
7. class-validator - npm. URL: https://www.npmjs.com/package/class-validator
8. Components and Props - React. URL: https://legacy.reactjs.org/docs/components-and-props.html
9. Controllers | NestJS - A progressive Node.js framework. URL: https://docs.nestjs.com/controllers
10. CORS | NestJS - A progressive Node.js framework. URL: https://docs.nestjs.com/security/cors
11. cross-env - npm. URL: https://www.npmjs.com/package/cross-env
12. Data Transfer Object DTO Definition and Usage | Okta.

URL: https://www.okta.com/identity-101/dto/

13. Dependency injection - Wikipedia.
    URL: https://en.wikipedia.org/wiki/Dependency_injection
14. Docker Compose overview.
    URL: https://docs.docker.com/compose/
15. Docker: Accelerated, Containerized Application Development.
    URL: https://www.docker.com/
16. Documentation | NestJS - A progressive Node.js framework.
    URL: https://docs.nestjs.com/
17. Exception filters | NestJS - A progressive Node.js framework.
    URL: https://docs.nestjs.com/exception-filters
18. Express - Node.js web application framework.
    URL: https://expressjs.com/
19. Fastify, Fast and low overhead web framework, for Node.js.
    URL: https://www.fastify.io/
20. Feature Overview v6.10.0 | React Router.
    URL: https://reactrouter.com/en/main/start/overview
21. fkhadra/react-contexify: Add a context menu to your react app with ease. URL: https://github.com/fkhadra/react-contexify
22. Getting Started | Axios Docs.
    URL: https://axios-http.com/docs/intro
23. Guards | NestJS - A progressive Node.js framework.
    URL: https://docs.nestjs.com/guards
24. HTTP response status codes - HTTP | MDN. URL: https://developer.mozilla.org/en-US/docs/Web/HTTP/Status
25. Interceptors | NestJS - A progressive Node.js framework.
    URL: https://docs.nestjs.com/interceptors
26. Introducing JSX - React.
    URL: https://legacy.reactjs.org/docs/introducing-jsx.html
27. JSON Web Token Introduction - jwt.io.

URL: https://jwt.io/introduction

28. Middleware | NestJS - A progressive Node.js framework.
    URL: https://docs.nestjs.com/middleware
29. Modules | NestJS - A progressive Node.js framework.
    URL: https://docs.nestjs.com/modules
30. OpenAPI (Swagger) | NestJS - A progressive Node.js framework.
    URL: https://docs.nestjs.com/openapi/introduction
31. Overview - CLI | NestJS - A progressive Node.js framework.
    URL: https://docs.nestjs.com/cli/overview
32. Pipes | NestJS - A progressive Node.js framework.
    URL: https://docs.nestjs.com/pipes
33. PostgreSQL: About. URL: https://www.postgresql.org/about/
34. Providers | NestJS - A progressive Node.js framework.
    URL: https://docs.nestjs.com/providers
35. README · MobX. URL: https://mobx.js.org/README.html
36. Rendering Elements - React.
    URL: https://legacy.reactjs.org/docs/rendering-elements.html
37. Sequelize | Feature-rich ORM for modern TypeScript & JavaScript.
    URL: https://sequelize.org/
38. sequelize-typescript - npm.
    URL: https://www.npmjs.com/package/sequelize-typescript
39. SQL (Sequelize) | NestJS - A progressive Node.js framework.
    URL: https://docs.nestjs.com/recipes/sql-sequelize
40. The Axios Instance | Axios Docs.
    URL: https://axios-http.com/docs/instance
41. TypeScript: JavaScript With Syntax For Types.
    URL: https://www.typescriptlang.org/
42. useContext - React.
    URL: https://react.dev/reference/react/useContext
43. useEffect - React.

URL: https://react.dev/reference/react/useEffect

44. useNavigate v6.10.0 | React Router.
    URL: https://reactrouter.com/en/main/hooks/use-navigate
45. useState - React.
    URL: https://react.dev/reference/react/useState
46. What is an ORM - The Meaning of Object Relational Mapping Database Tools. URL: https://www.freecodecamp.org/news/what-is-an-orm-the-meaning-of-object-relational-mapping-database-tools/
47. Why use PostgreSQL as a Database for my Next Project in 2022 - Fulcrum.
    URL: https://fulcrum.rocks/blog/why-use-postgresql-database

ДОДАТКИ

Додаток А

Приклади документації API розробленої за допомогою Swagger

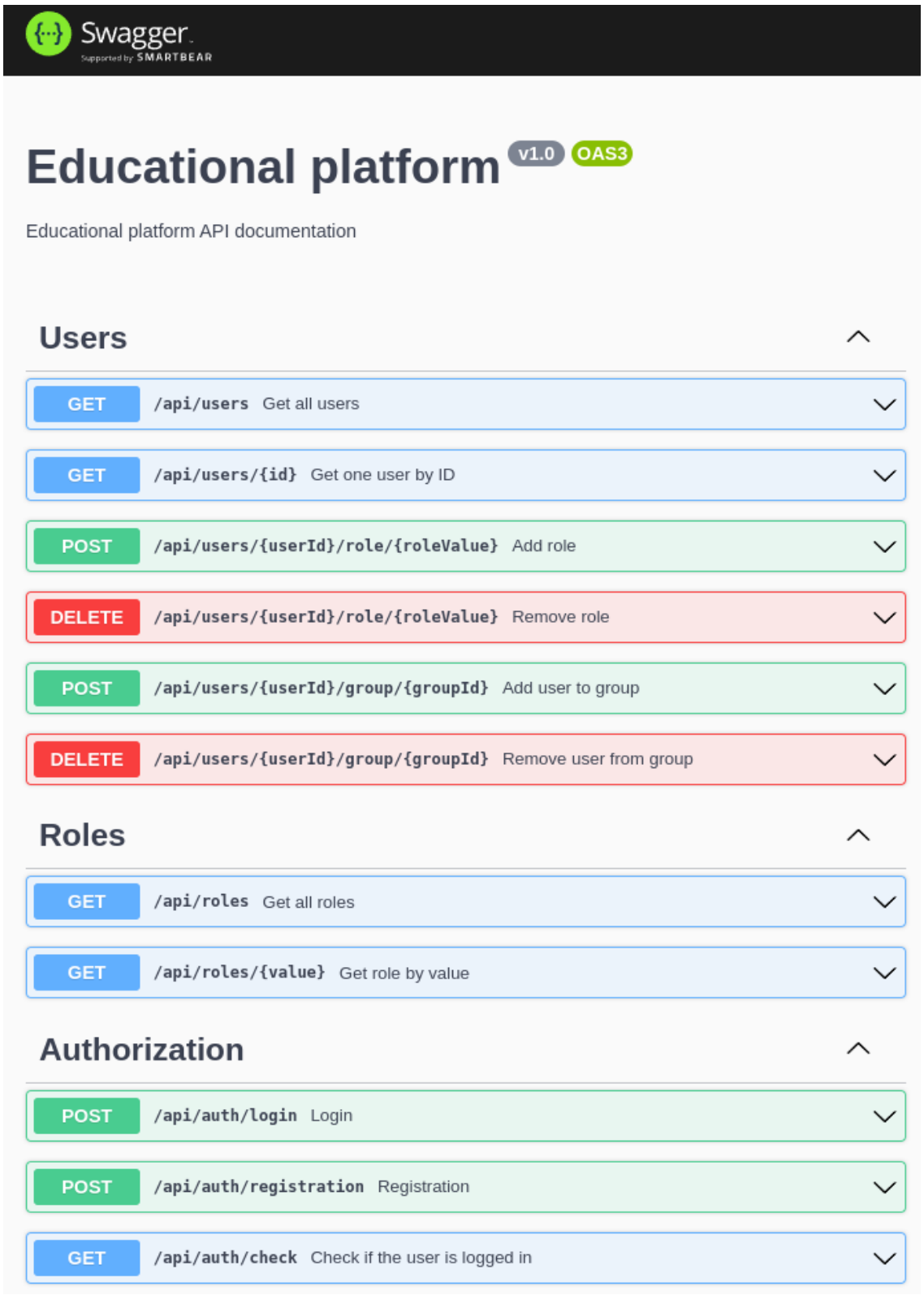

Рисунок А.1 – Список кінцевих точок API

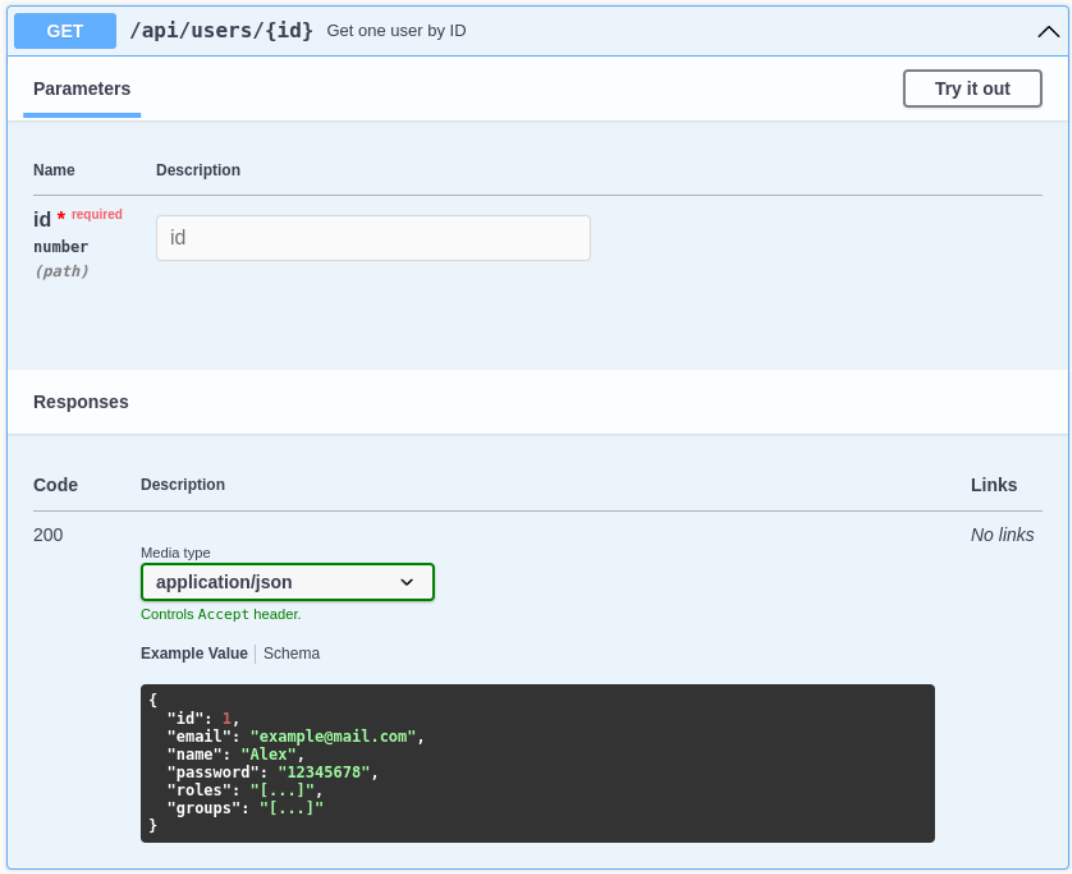

Рисунок А.2 – Інформація про кінцеву точку API

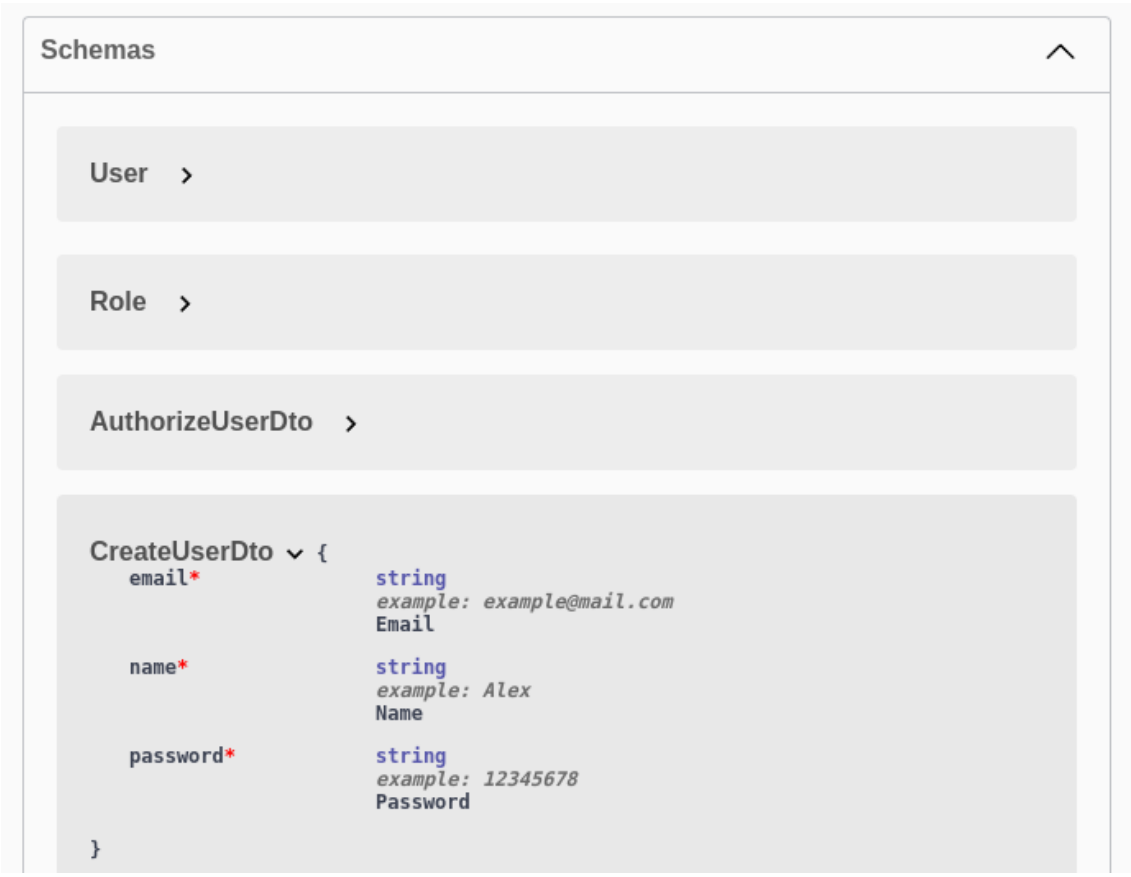

Рисунок А.3 – Моделі та DTO

Додаток Б

Приклади розробленого графічного інтерфейсу

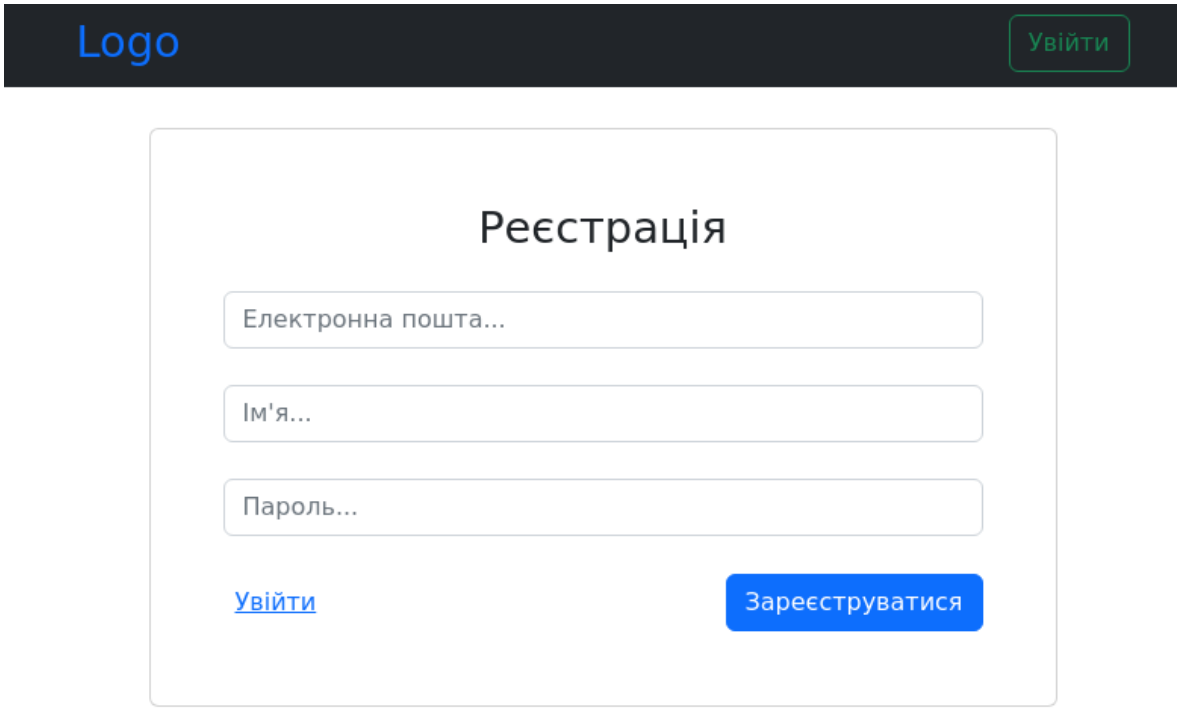

Рисунок Б.1 – Форма реєстрації

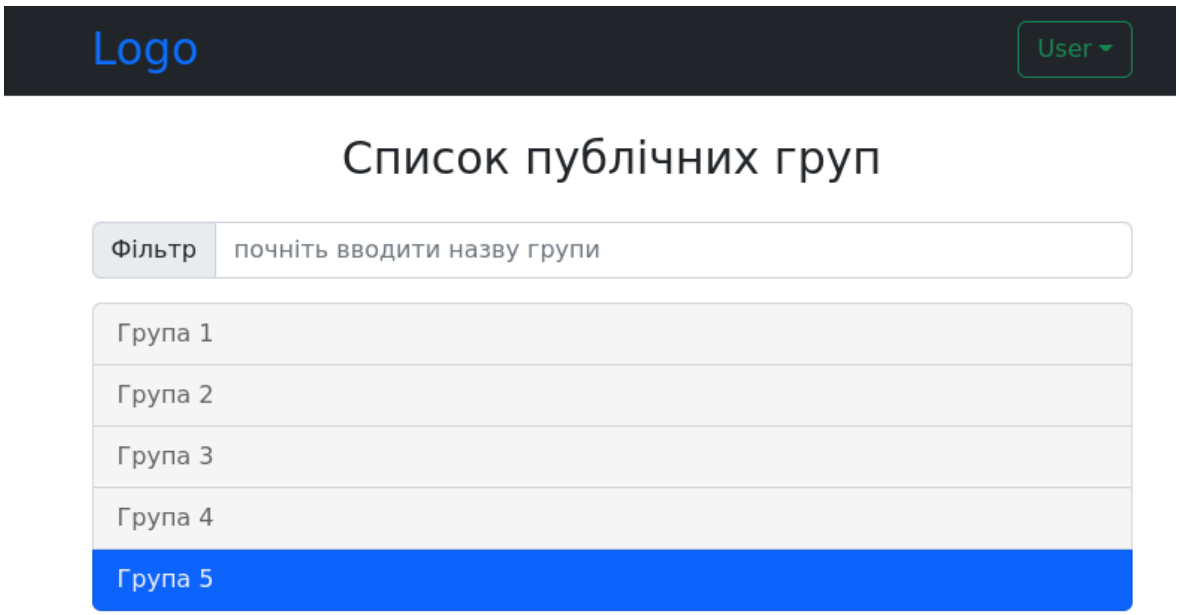

Рисунок Б.2 – Пошук відкритої групи

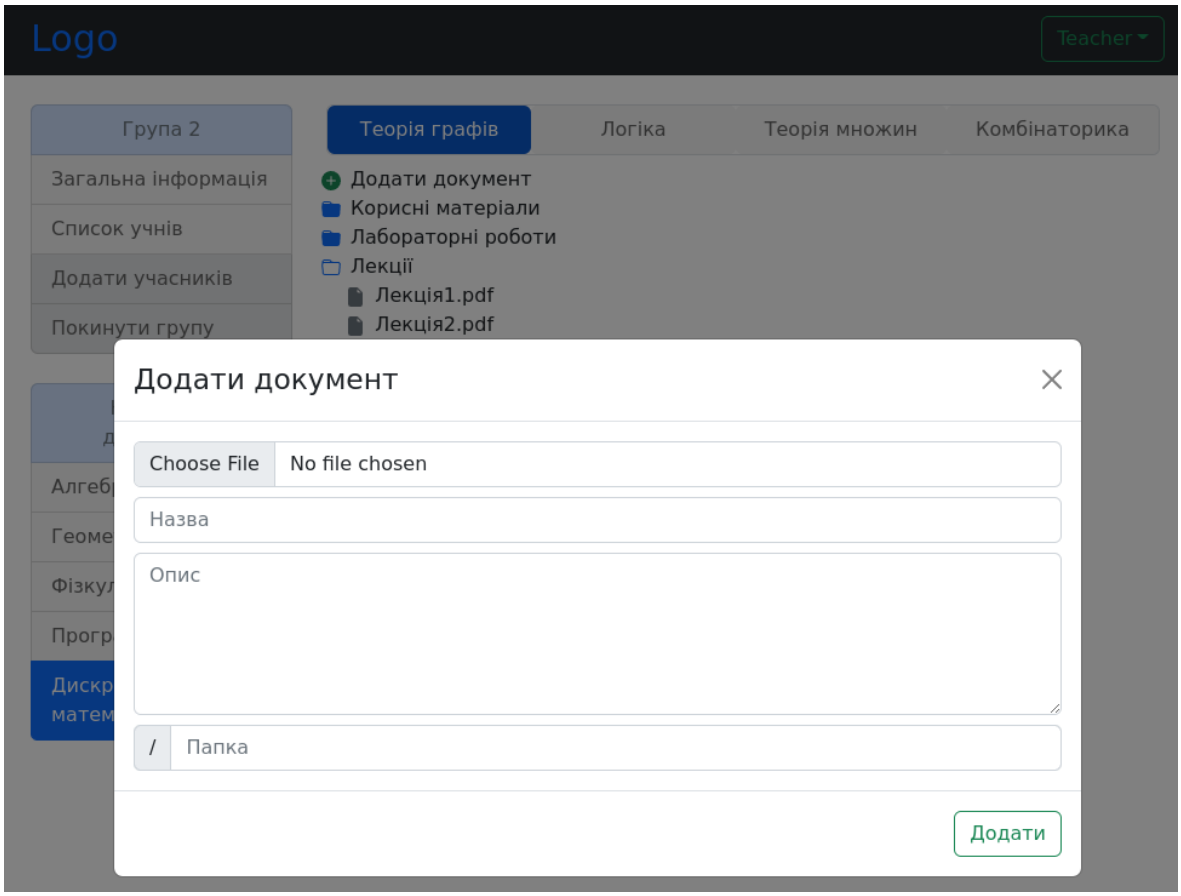

Рисунок Б.3 – Можливості викладача. Додавання матеріалу

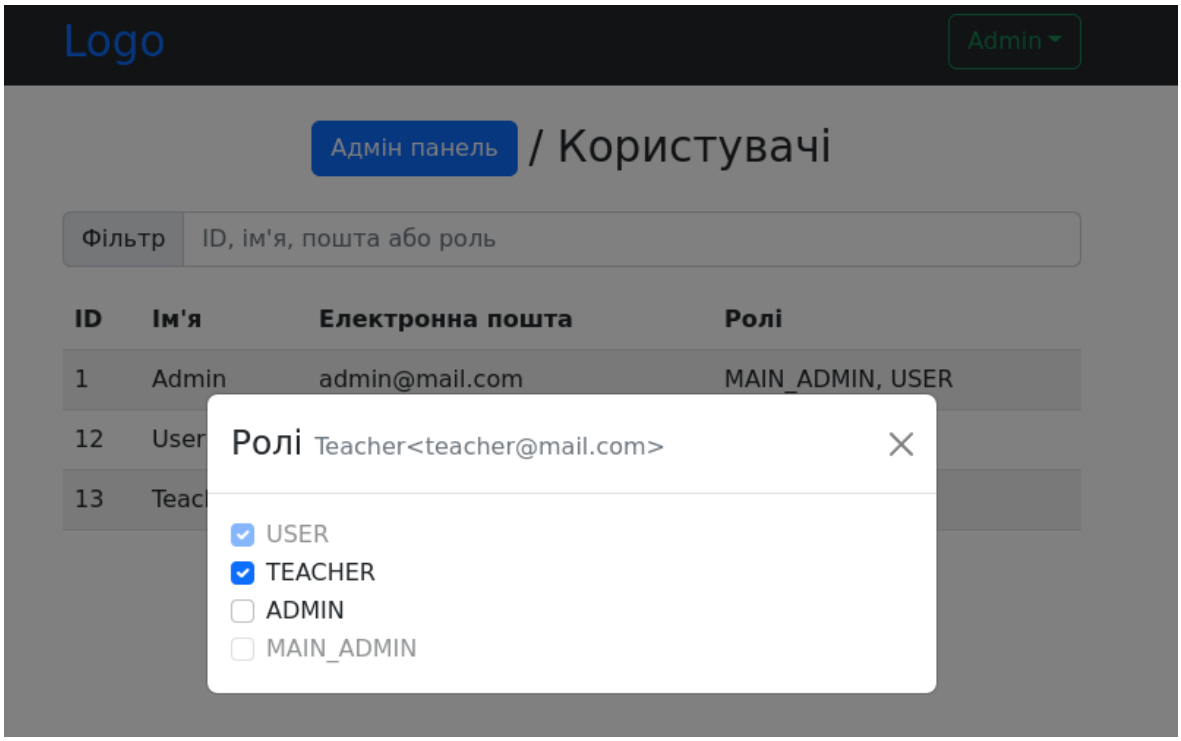

Рисунок Б.4 – Адмін панель. Видача ролей

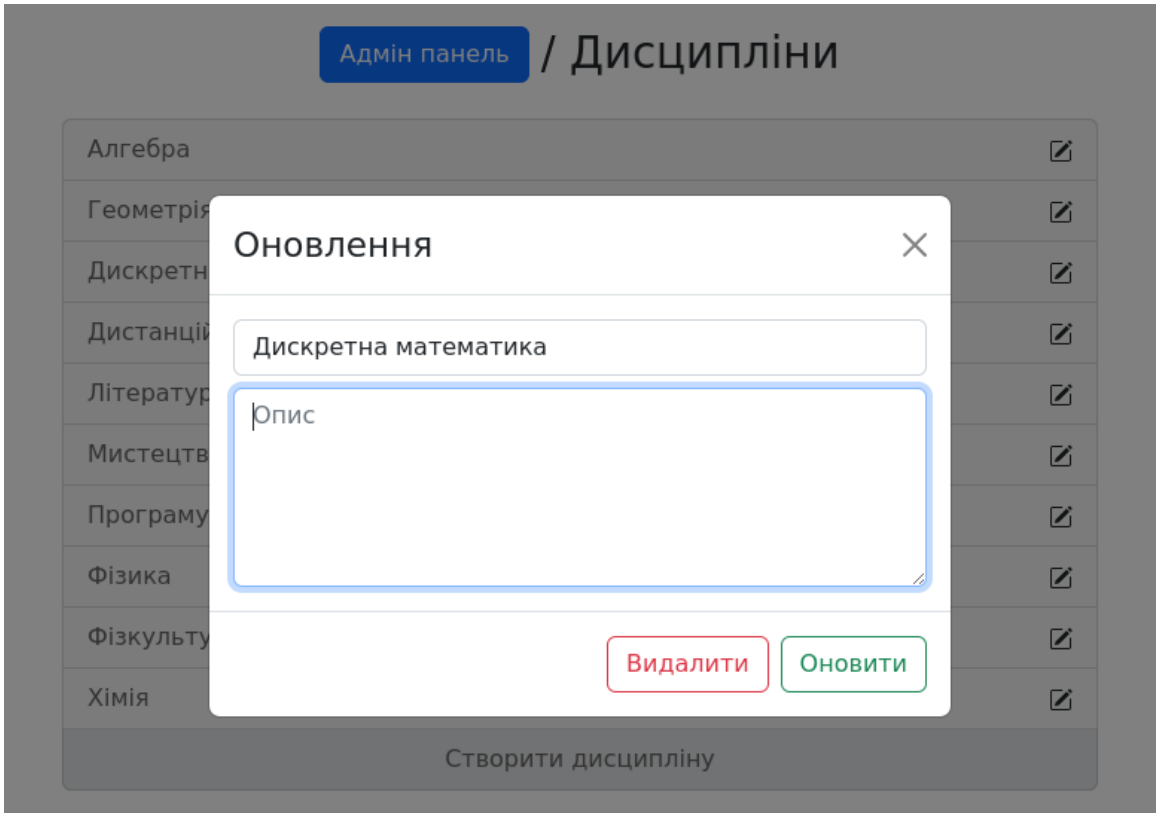

Рисунок Б.5 – Адмін панель. Оновлення дисципліни

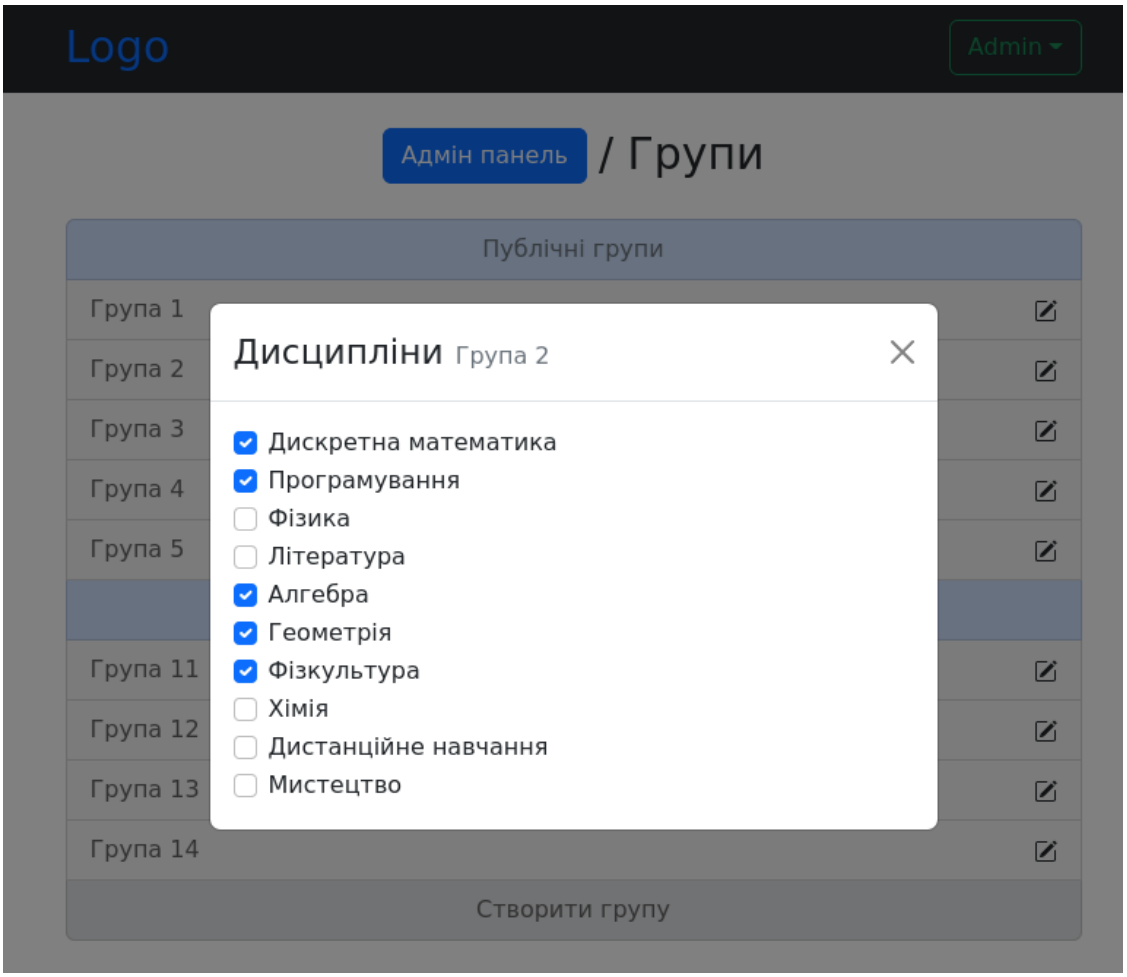

Рисунок Б.6 – Адмін панель. Оновлення списку дисциплін для вивчення у групі

КОДЕКС АКАДЕМІЧНОЇ ДОБРОЧЕСНОСТІ
ЗДОБУВАЧА ВИЩОЇ ОСВІТИ
ХЕРСОНСЬКОГО ДЕРЖАВНОГО УНІВЕРСИТЕТУ

Я, Нетьосов Микита Володимирович, учасник(ця) освітнього процесу Херсонського державного університету, **УСВІДОМЛЮЮ**, що академічна доброчесність – це фундаментальна етична цінність усієї академічної спільноти світу.

ЗАЯВЛЯЮ, що у своїй освітній і науковій діяльності **ЗОБОВ'ЯЗУЮСЯ**:

– дотримуватися:
- вимог законодавства України та внутрішніх нормативних документів університету, зокрема Статуту Університету;
- принципів та правил академічної доброчесності;
- нульової толерантності до академічного плагіату;
- моральних норм та правил етичної поведінки;
- толерантного ставлення до інших;
- дотримуватися високого рівня культури спілкування;

– надавати згоду на:
- безпосередню перевірку курсових, кваліфікаційних робіт тощо на ознаки наявності академічного плагіату за допомогою спеціалізованих програмних продуктів;
- оброблення, збереження й розміщення кваліфікаційних робіт у відкритому доступі в інституційному репозитарії;
- використання робіт для перевірки на ознаки наявності академічного плагіату в інших роботах виключно з метою виявлення можливих ознак академічного плагіату;

– самостійно виконувати навчальні завдання, завдання поточного й підсумкового контролю результатів навчання;
– надавати достовірну інформацію щодо результатів власної навчальної (наукової, творчої) діяльності, використаних методик досліджень та джерел інформації;
– не використовувати результати досліджень інших авторів без використання покликань на їхню роботу;
– своєю діяльністю сприяти збереженню та примноженню традицій університету, формуванню його позитивного іміджу;
– не чинити правопорушень і не сприяти їхньому скоєнню іншими особами;
– підтримувати атмосферу довіри, взаємної відповідальності та співпраці в освітньому середовищі;
– поважати честь, гідність та особисту недоторканність особи, незважаючи на її стать, вік, матеріальний стан, соціальне становище, расову належність, релігійні й політичні переконання;
– не дискримінувати людей на підставі академічного статусу, а також за національною, расовою, статевою чи іншою належністю;
– відповідально ставитися до своїх обов'язків, вчасно та сумлінно виконувати необхідні навчальні та науководослідницькі завдання;
– запобігати виникненню у своїй діяльності конфлікту інтересів, зокрема не використовувати службових і родинних зв'язків з метою отримання нечесної переваги в навчальній, науковій і трудовій діяльності;
– не брати участі в будь-якій діяльності, пов'язаній із обманом, нечесністю, списуванням, фабрикацією;
– не підроблювати документи;
– не поширювати неправдиву та компрометуючу інформацію про інших здобувачів вищої освіти, викладачів і співробітників;
– не отримувати і не пропонувати винагород за несправедливе отримання будь-яких переваг або здійснення впливу на зміну отриманої академічної оцінки;
– не залякувати й не проявляти агресії та насильства проти інших, сексуальні домагання;
– не завдавати шкоди матеріальним цінностям, матеріально-технічній базі університету та особистій власності інших студентів та/або працівників;
– не використовувати без дозволу ректорату (деканату) символіки університету в заходах, не пов'язаних з діяльністю університету;
– не здійснювати і не заохочувати будь-яких спроб, спрямованих на те, щоб за допомогою нечесних і негідних методів досягати власних корисних цілей;
– не завдавати загрози власному здоров'ю або безпеці іншим студентам та/або працівникам.

УСВІДОМЛЮЮ, що відповідно до чинного законодавства у разі недотримання Кодексу академічної доброчесності буду нести академічну та/або інші види відповідальності й до мене можуть бути застосовані заходи дисциплінарного характеру за порушення принципів академічної доброчесності.

12.09.2019		Микита Нетьосов
(дата)	(підпис)	(ім'я, прізвище)