

**МІНІСТЕРСТВО ФІНАНСІВ УКРАЇНИ  
ДЕРЖАВНИЙ ПОДАТКОВИЙ УНІВЕРСИТЕТ**

**ФАКУЛЬТЕТ ФІНАНСІВ ТА ЦИФРОВИХ ТЕХНОЛОГІЙ  
КАФЕДРА КОМП'ЮТЕРНИХ ТА ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ І  
СИСТЕМ**

**КВАЛІФІКАЦІЙНА РОБОТА**

**на тему:** Проектування та розробка веборієнтованої інформаційної системи  
підтримки освітнього процесу

Здобувача вищої освіти першого (бакалаврського) рівня,  
IV курсу, групи КБ-22-1, спеціальності 122 «Комп'ютерні науки»,  
освітня програма «Інформаційні управляючі системи і технології (за  
галузями)»

  
(підпис)

Кравчука Мар'яна Романовича  
(прізвище, ім'я, по батькові)

Науковий керівник

  
(підпис)

Філоненко Михайло  
Миколайович, канд. фіз.-мат.  
наук, доцент  
(прізвище, ім'я, по батькові, науковий ступінь,  
вчене звання)

Керівник освітньо-  
професійної програми

  
(підпис)

Ніжегородцев Владислав  
Олександрович, канд. пед.  
наук, доцент  
(прізвище, ім'я, по батькові, науковий ступінь,  
вчене звання)

Допущено до захисту за рішенням кафедри комп'ютерних та інформаційних  
технологій і систем протокол від «12» травня 2026 №20

Завідувач кафедри

  
(підпис)

Філоненко Михайло  
Миколайович, канд. фіз.-мат.  
наук, доцент  
(прізвище, ім'я, по батькові, науковий ступінь,  
вчене звання)

Реєстрація № 64

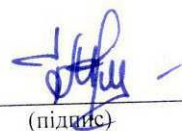
28.05.2026  
(дата)

  
(підпис відповідальної особи)

Результати  
перевірки

80 захисту  
(до захисту, на допитування)

12.05.2026  
(дата)

  
(підпис)

Результат  
захисту

4 (добре)  
(оцінка)

09.06.2026  
(дата)

  
(підпис)

Голова комісії:

  
(підпис)

О.С. Дюбанов

Члени комісії:

  
(підпис)

М.М. Філоненко

  
(підпис)

В.О. Ніжегородцев

  
(підпис)

А.А. Омельчук

# БЛАНК ОЦІНЮВАННЯ КВАЛІФІКАЦІЙНОЇ РОБОТИ

| Об'єкти контролю                                                                                                                                                         | Діапазон оцінки в балах | Фактична кількість балів | Примітки | Дата, П.І.Б.: керівника, голови комісії членів комісії, секретаря, (підписи) |
|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-------------------------|--------------------------|----------|------------------------------------------------------------------------------|
| ОЦІНЮВАННЯ КВАЛІФІКАЦІЙНОЇ РОБОТИ КОМІСІЄЮ ПО ЗАХИСТУ                                                                                                                    |                         |                          |          |                                                                              |
| міст випускної роботи<br>сть теоретичного матеріалу, його<br>зок з практичним, вміння<br>дставляти матеріал в логічній<br>підовності, наявність висновків<br>пропозицій) | від 0 до 30             | 23                       |          | 09.06.2016<br>Філоненко М.М.<br>Дюбанов О.С.                                 |
| отримання вимог щодо<br>рмлення                                                                                                                                          | від 0 до 10             | 7                        |          | Філоненко М.М.<br>Ніжегородцев В.О.                                          |
| часне виконання роботи на<br>етапах у встановлені кафедрою<br>міни                                                                                                       | від 0 до 5              | 4                        |          | Омельчук А.А.<br>Суховерхня Л.П.                                             |
| рецензент                                                                                                                                                                | від 0 до 5              | 4                        |          |                                                                              |
| КОМ БАЛІВ                                                                                                                                                                | від 0 до 50             | 38                       |          |                                                                              |

оцінювання кваліфікаційної роботи науковим керівником – 37 балів.

## ПЕРЕВЕДЕННЯ ОЦІНКИ ЗА 100-БАЛЬНОЮ ШКАЛОЮ ДО НАЦІОНАЛЬНОЇ ШКАЛИ ТА ШКАЛИ ЄКТС

| Оцінка за 100-бальною шкалою | Оцінка за шкалою ЄКТС | Оцінка за національною шкалою | Дата       | П.І.Б.: голови, членів комісії, секретаря, (підпис)                                     |
|------------------------------|-----------------------|-------------------------------|------------|-----------------------------------------------------------------------------------------|
| 75                           | C                     | 4 (добре)                     | 09.06.2016 | Дюбанов О.С.<br>Філоненко М.М.<br>Ніжегородцев В.О.<br>Омельчук А.А.<br>Суховерхня Л.П. |

## ПРАВИЛА ПЕРЕВЕДЕННЯ ОЦІНОК ДО НАЦІОНАЛЬНОЇ ШКАЛИ ТА ШКАЛИ ЄКТС

| Оцінка за 100-бальною шкалою | Оцінка за шкалою ЄКТС | Оцінка за національною шкалою |
|------------------------------|-----------------------|-------------------------------|
| 90-100                       | A                     | 5 (відмінно)                  |
| 80-89                        | B                     | 4 (добре)                     |
| 70-79                        | C                     |                               |
| 60-69                        | D                     |                               |
| 50-59                        | E                     | 3 (задовільно)                |
| 35-49                        | FX                    | 2 (незадовільно)              |

Факультет фінансів та цифрових технологій

Кафедра комп'ютерних та інформаційних технологій і систем

Галузь знань: 12 «Інформаційні технології»

Спеціальність: 122 «Комп'ютерні науки»

Освітня програма: «Інформаційні управляючі системи і технології (за галузями)»

Затверджую

Завідувач кафедри  Філоненко М.М.

«01» грудня 2025

Завдання на кваліфікаційну роботу

здобувачу вищої освіти

Кравчуку Мар'яну Романовичу

(прізвище, ім'я, по батькові)

1. Тема кваліфікаційної роботи Проектування та розробка веборієнтованої інформаційної системи підтримки освітнього процесу

Затверджена наказом ректора від «28» листопада 2025 № 1526

2. Строк здачі закінченої роботи «08» травня 2026

3. Мета роботи Розробка та реалізація веборієнтованої інформаційної системи для автоматизації управління користувачами, навчальними дисциплінами та розкладом занять

Об'єкт дослідження Процеси автоматизації та інформаційної підтримки освітнього процесу в закладах вищої освіти.

Предмет дослідження Методи, моделі та програмні засоби розробки веборієнтованих інформаційних систем для управління користувачами, навчальними дисциплінами та розкладом занять.

4. Вихідні дані до роботи Сучасні технології веброзробки: Python, FastAPI, SQLite, SQLAlchemy, React, Vite, TailwindCSS, Axios, JWT; аналіз існуючих платформ підтримки освітнього процесу та систем управління навчальним процесом.

5. Перелік графічного матеріалу

6. Зміст кваліфікаційної роботи (перелік питань за кожним розділом)

Вступ

Розділ 1. Теоретико-методологічні засади проектування веборієнтованих інформаційних систем для освітнього процесу

Розділ 2. Системно-аналітичне проектування інформаційної системи

Розділ 3. Програмна реалізація веборієнтованої інформаційної системи

Висновки

Список використаних джерел

## 7. Календарний план виконання роботи

| №<br>п/п | Назва етапів кваліфікаційної роботи                                                                                                                                                                  | Строк виконання етапів роботи |               |
|----------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-------------------------------|---------------|
|          |                                                                                                                                                                                                      | За планом                     | Фактично      |
| 1        | 2                                                                                                                                                                                                    | 3                             | 4             |
| 1        | Вибір теми, підбір літературних джерел, формування структури кваліфікаційної роботи                                                                                                                  | До 30.10.2025                 | До 30.10.2025 |
| 2        | <b>I розділ</b> (розкрити сутність і стан дослідження наукової проблематики роботи, після чого навести її загальну характеристику; огляд літератури за темою і вибір конкретних напрямів досліджень) | 26.12.2025                    | 26.12.2025    |
| 3        | <b>II розділ</b> (аналіз експериментальної частини про методики досліджень)                                                                                                                          | 18.02.2026                    | 18.02.2026    |
| 4        | <b>III розділ</b> (проведені теоретичні і (або) експериментальні дослідження та проектні рішення)                                                                                                    | 16.04.2026                    | 16.04.2026    |
| 5        | <b>Висновки та оформлення роботи</b>                                                                                                                                                                 | 01.05.2026                    | 01.05.2026    |
| 6        | Доопрацювання роботи з урахуванням зауважень наукового керівника                                                                                                                                     | 07.05.2026                    | 07.05.2026    |
| 7        | Отримання відгуку керівника                                                                                                                                                                          | 29.05.2026                    | 29.05.2026    |
| 8        | Отримання зовнішньої рецензії                                                                                                                                                                        | 01.06.2026                    | 01.06.2026    |

8. Дата видачі завдання «01» грудня 2025

9. Керівник кваліфікаційної роботи

  
(підпис)

Філоненко М.М.

(прізвище, ініціали)

10. Керівник освітньо-професійної програми

  
(підпис)

Ніжегородцев В.О.

(прізвище, ініціали)

11. Завдання до виконання отримав здобувач вищої освіти

  
(підпис)

Кравчук М.Р.

(прізвище, ініціали)

## АНОТАЦІЯ

### Проектування та розробка веборієнтованої інформаційної системи підтримки освітнього процесу

Мета роботи — проектування та програмна реалізація веборієнтованої інформаційної системи для автоматизації управління користувачами, навчальними дисциплінами та розкладом занять у закладах вищої освіти.

Завданнями роботи є:

- Дослідження сутності та класифікації інформаційних систем підтримки освітнього процесу й порівняльний аналіз існуючих платформ (Moodle, Google Classroom, Microsoft Teams, АСУ ЗВО).
- Обґрунтування вибору технологічного стеку та проектування архітектури системи, моделі даних і специфікації REST API.
- Розробка серверної частини на основі FastAPI з рольовою моделлю доступу та JWT-авторизацією, а також клієнтського інтерфейсу засобами React.

Об'єкт дослідження — процес проектування та розробки веборієнтованих інформаційних систем для підтримки освітнього процесу у закладах вищої освіти. Предмет дослідження — методи, технології та інструментарій реалізації серверної та клієнтської частин системи на основі FastAPI, React та SQLite.

У результаті роботи розроблено функціональну веборієнтовану інформаційну систему, що реалізує трирівневу архітектуру, рольову модель доступу (студент, викладач, адміністратор), управління каталогом дисциплін та тижневим розкладом занять.

Одержані результати можуть бути впроваджені в освітній процес кафедр закладів вищої освіти як спрощена альтернатива комерційним LMS-платформам або як основа для подальшого розширення функціоналу.

Кваліфікаційна робота містить 85 сторінок, 2 таблиці, 6 рисунків, список використаних джерел із 45 найменувань, 4 додатки.

Ключові слова: веборієнтована інформаційна система, освітній процес, FastAPI, React, SQLite, REST API, JWT-авторизація, рольова модель доступу.

Рік виконання кваліфікаційної роботи: 2026

Рік захисту роботи: 2026

## **ABSTRACT**

### **Design and Development of a Web-Based Information System to Support the Educational Process**

The purpose of the work is the design and software implementation of a web-based information system for automating the management of users, academic disciplines, and class schedules at higher education institutions.

The tasks of the work are:

- Investigation of the nature and classification of information systems supporting the educational process, and a comparative analysis of existing platforms (Moodle, Google Classroom, Microsoft Teams, university management systems).
- Justification of the technology stack selection and design of the system architecture, data model, and REST API specification.
- Development of the server-side using FastAPI with a role-based access model and JWT authentication, as well as the client-side interface using React.

The object of research is the process of designing and developing web-based information systems to support the educational process at higher education institutions. The subject of research is the methods, technologies, and tools for implementing the server-side and client-side components based on FastAPI, React, and SQLite.

As a result of the work, a functional web-based information system was developed implementing a three-tier architecture, a role-based access model (student, teacher, administrator), management of the discipline catalogue, and a weekly class schedule.

The obtained results can be deployed in the educational process of university departments as a lightweight alternative to commercial LMS platforms or as a foundation for further functional expansion.

The qualification paper contains 85 pages, 2 tables, 6 figures, a list of references with 45 items, and 4 appendices.

**Keywords:** web-based information system, educational process, FastAPI, React, SQLite, REST API, JWT authentication, role-based access model.

**Year of completion of the qualification paper:** 2026

**Year of defense of the work:** 2026

## ЗМІСТ

|                                                                                                                              |    |
|------------------------------------------------------------------------------------------------------------------------------|----|
| ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ ТА СКОРОЧЕНЬ.....                                                                                  | 3  |
| ВСТУП.....                                                                                                                   | 5  |
| РОЗДІЛ 1. ТЕОРЕТИКО-МЕТОДОЛОГІЧНІ ЗАСАДИ ПРОЄКТУВАННЯ<br>ВЕБОРІЄНТОВАНИХ ІНФОРМАЦІЙНИХ СИСТЕМ ДЛЯ ОСВІТНЬОГО<br>ПРОЦЕСУ..... | 9  |
| 1.1. Сутність та класифікація інформаційних систем підтримки освітнього<br>процесу.....                                      | 9  |
| 1.2. Огляд та порівняльний аналіз існуючих платформ (Moodle, Google<br>Classroom, Microsoft Teams, АСУ ЗВО).....             | 13 |
| 1.3. Обґрунтування вибору технологічного стеку для розробки веборієнтованої<br>інформаційної системи.....                    | 17 |
| РОЗДІЛ 2. СИСТЕМНО-АНАЛІТИЧНЕ ПРОЄКТУВАННЯ ІНФОРМАЦІЙНОЇ<br>СИСТЕМИ.....                                                     | 25 |
| 2.1. Архітектура системи та функціональні вимоги.....                                                                        | 25 |
| 2.2. Проєктування моделі даних та схеми бази даних.....                                                                      | 31 |
| 2.3. Проєктування REST API та механізмів автентифікації.....                                                                 | 37 |
| РОЗДІЛ 3. ПРОГРАМНА РЕАЛІЗАЦІЯ ВЕБОРІЄНТОВАНОЇ<br>ІНФОРМАЦІЙНОЇ СИСТЕМИ.....                                                 | 44 |
| 3.1. Реалізація серверної частини на основі FastAPI.....                                                                     | 44 |
| 3.2. Розробка клієнтського інтерфейсу засобами React.....                                                                    | 52 |
| 3.3. Тестування системи та аналіз отриманих результатів.....                                                                 | 59 |
| ВИСНОВКИ.....                                                                                                                | 65 |
| СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ.....                                                                                              | 67 |
| ДОДАТКИ.....                                                                                                                 | 70 |

## **ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ ТА СКОРОЧЕНЬ**

- API (Application Programming Interface) — програмний інтерфейс прикладного програмування
- CRUD (Create, Read, Update, Delete) — базовий набір операцій над даними: створення, читання, оновлення, видалення
- CSS (Cascading Style Sheets) — каскадні таблиці стилів
- DOM (Document Object Model) — об'єктна модель документа
- HTML (HyperText Markup Language) — мова гіпертекстової розмітки
- HTTP (HyperText Transfer Protocol) — протокол передачі гіпертексту
- JSON (JavaScript Object Notation) — текстовий формат обміну даними
- JWT (JSON Web Token) — стандарт токенів доступу на основі JSON
- LMS (Learning Management System) — система управління навчанням
- MVC (Model-View-Controller) — архітектурний шаблон "модель-представлення-контролер"
- ORM (Object-Relational Mapping) — об'єктно-реляційне відображення
- REST (Representational State Transfer) — архітектурний стиль розподілених систем
- SaaS (Software as a Service) — програмне забезпечення як послуга
- SIS (Student Information System) — інформаційна система обліку студентів
- SPA (Single Page Application) — односторінковий веб-додаток
- SQL (Structured Query Language) — мова структурованих запитів
- UI (User Interface) — інтерфейс користувача
- UX (User Experience) — користувацький досвід
- АСУ — автоматизована система управління
- БД — база даних
- ЗВО — заклад вищої освіти
- ІС — інформаційна система
- ІТ — інформаційні технології

ОПП — освітньо-професійна програма

ПЗ — програмне забезпечення

ПК — персональний комп'ютер

СУБД — система управління базами даних

## ВСТУП

**Актуальність теми.** Сучасний етап розвитку вищої освіти характеризується глибокою цифровою трансформацією, що зумовлена переходом закладів вищої освіти до змішаних і дистанційних форм навчання, посиленням вимог до прозорості освітнього процесу, а також зростанням ролі інформаційно-комунікаційних технологій у щоденній взаємодії викладачів і здобувачів освіти. Пандемія COVID-19, повномасштабна війна на території України, а також загальносвітовий тренд цифровізації державного сектору поставили перед закладами вищої освіти завдання забезпечити безперервність освітнього процесу за будь-яких зовнішніх обставин. Це, своєю чергою, підвищило попит на надійні, доступні та функціональні веборієнтовані інформаційні системи, які забезпечують підтримку основних освітніх процесів — управління розкладом, обліку дисциплін, реєстрації користувачів та комунікації між учасниками навчання.

Більшість закладів вищої освіти України використовують комплекс різнорідних інструментів: загальносвітові платформи (Moodle, Google Classroom, Microsoft Teams), внутрішні автоматизовані системи управління (АСУ ЗВО) власної розробки, а також низку допоміжних сервісів. Проте подібний підхід має низку обмежень: висока вартість впровадження готових платформ, складність їх локалізації та інтеграції з внутрішніми процесами закладу, обмежені можливості розширення функціоналу під специфічні потреби кафедри або факультету. Натомість розробка власної веборієнтованої системи з відкритою архітектурою дозволяє побудувати рішення, яке точно відповідає вимогам конкретного закладу, легко масштабується та може поетапно нарощувати функціональність.

Розвиток сучасних веб-фреймворків (FastAPI, React, Vue.js), технологій реляційних і нереляційних баз даних, а також стандартів безпеки (OAuth 2.0, JWT) створює технологічну основу для швидкої побудови якісних інформаційних систем силами невеликих команд або навіть одного розробника.

Це робить актуальним дослідження принципів проєктування та практичної реалізації веборієнтованих інформаційних систем для підтримки освітнього процесу засобами сучасного стеку технологій.

**Стан і ступінь розробки проблеми в спеціальній літературі.** Питанням проєктування інформаційних систем для освітньої сфери присвячено праці зарубіжних вчених — Б. Скіннера, Дж. Девіса, Е. Венгера у частині методології електронного навчання, а також М. Доумаса, Р. Філдінга, М. Фаулера у частині архітектурних підходів до побудови веб-систем. Серед українських дослідників ці питання розглядали В. Ю. Биков, М. І. Жалдак, Н. В. Морзе, С. О. Семеріков, А. М. Стрюк та інші. Технологічні аспекти створення веборієнтованих систем висвітлено у працях, присвячених фреймворкам FastAPI (С. Рамірес), React (Д. Абрамов, А. Османі), а також у документації відповідних відкритих проєктів.

Водночас більшість наявних досліджень стосується або великих корпоративних рішень класу LMS, або теоретичних аспектів електронної педагогіки, тоді як питання практичної побудови компактних, спрощених систем підтримки освітнього процесу силами навчальних колективів кафедр висвітлені недостатньо. Потребує додаткового вивчення питання вибору оптимального технологічного стеку для подібних задач, а також методики поетапної реалізації функціоналу від мінімально життєздатного продукту до повноцінної системи.

**Мета і завдання дослідження.** Метою кваліфікаційної роботи є проєктування та розробка веборієнтованої інформаційної системи для підтримки освітнього процесу, що реалізує базові функції управління користувачами, дисциплінами та розкладом навчальних занять.

Для досягнення мети у роботі поставлено такі завдання:

1. Дослідити сутність та класифікацію інформаційних систем підтримки освітнього процесу, провести порівняльний аналіз існуючих платформ.

2. Обґрунтувати вибір технологічного стеку для побудови системи з урахуванням сучасних тенденцій веб-розробки.
3. Спроекувати архітектуру системи, модель даних та специфікацію REST API.
4. Розробити серверну частину системи на основі фреймворку FastAPI з підтримкою ролівої моделі доступу та JWT-авторизації.
5. Реалізувати клієнтський інтерфейс засобами бібліотеки React із застосуванням сучасних підходів до проєктування односторінкових веб-додатків.
6. Провести тестування системи, проаналізувати отримані результати та сформулювати пропозиції щодо подальшого розвитку.

**Об'єкт дослідження** — процес проєктування та розробки веборієнтованих інформаційних систем для підтримки освітнього процесу у закладах вищої освіти.

**Предмет дослідження** — методи, технології та інструментарій реалізації серверної та клієнтської частин веборієнтованої інформаційної системи на основі фреймворку FastAPI, бібліотеки React та реляційної бази даних SQLite.

**Методи дослідження.** У роботі використано загальнонаукові та спеціальні методи: метод системного аналізу — для дослідження предметної області та виявлення функціональних вимог; метод порівняльного аналізу — для зіставлення існуючих платформ і технологій; метод структурного проєктування — для побудови архітектури системи; об'єктно-орієнтований підхід — для розробки програмного коду; методи тестування програмного забезпечення — для перевірки коректності реалізації.

**База дослідження** — інформаційна інфраструктура закладів вищої освіти України, зокрема Державного податкового університету; відкриті дані щодо технологічних рішень у сфері освітніх інформаційних систем; технічна документація фреймворків і бібліотек, використаних під час розробки.

**Наукова та практична значущість одержаних результатів.** Наукова значущість полягає в систематизації підходів до побудови веборієнтованих інформаційних систем освітнього призначення на основі сучасного стеку технологій Python + JavaScript, а також у формуванні рекомендацій щодо вибору архітектурних рішень для систем подібного класу.

Практична значущість роботи полягає в тому, що розроблена інформаційна система може бути впроваджена в освітній процес кафедр закладів вищої освіти як спрощена альтернатива комерційним LMS-платформам або як основа для подальшого розширення функціоналу. Архітектура системи дозволяє масштабувати її функціональні можливості шляхом додавання нових модулів без переписування основних компонентів.

## **РОЗДІЛ 1. ТЕОРЕТИКО-МЕТОДОЛОГІЧНІ ЗАСАДИ ПРОЄКТУВАННЯ ВЕБОРІЄНТОВАНИХ ІНФОРМАЦІЙНИХ СИСТЕМ ДЛЯ ОСВІТНЬОГО ПРОЦЕСУ**

### **1.1. Сутність та класифікація інформаційних систем підтримки освітнього процесу**

Цифрова трансформація сфери вищої освіти, що активно відбувається протягом останніх двох десятиліть, тісно пов'язана з впровадженням різноманітних інформаційних систем, які автоматизують і підтримують різні аспекти освітньої діяльності закладів вищої освіти. Розуміння сутності таких систем, їх типології та функціональних можливостей є необхідною передумовою для розробки нових рішень у цій сфері. У цьому підрозділі розглядається поняття інформаційної системи підтримки освітнього процесу, її основні характеристики, класифікація та сучасні тенденції розвитку.

У загальному значенні інформаційна система визначається як організована сукупність взаємопов'язаних компонентів — апаратних засобів, програмного забезпечення, баз даних, мережевих ресурсів, процедур та персоналу, — призначених для збору, обробки, зберігання та поширення інформації з метою підтримки прийняття рішень, координації, контролю, аналізу та управління в певній предметній області. Зокрема, у працях українських дослідників В. Ю. Бикова, М. І. Жалдака та Н. В. Морзе наголошується, що сучасна інформаційна система є соціотехнічною системою, тобто такою, що нерозривно поєднує технологічні рішення з організаційними процесами та людською взаємодією.

Інформаційна система підтримки освітнього процесу — це спеціалізована категорія інформаційних систем, що забезпечує автоматизацію процесів планування, організації, реалізації та контролю навчальної діяльності у закладі освіти. Основними учасниками таких систем є здобувачі вищої освіти, науково-педагогічні працівники, адміністративний персонал, а в широкому сенсі — також роботодавці, батьки та інші зацікавлені сторони. Функціональне призначення

подібних систем охоплює автентифікацію користувачів, ведення розкладу занять, управління освітніми компонентами (дисциплінами, курсами, модулями), підтримку комунікації, оцінювання навчальних досягнень, формування звітності.

Веборієнтована інформаційна система — це інформаційна система, доступ до якої здійснюється через стандартний веб-браузер за допомогою протоколу HTTP/HTTPS, без необхідності встановлення спеціального клієнтського програмного забезпечення. Така архітектурна модель набула домінування у сфері освітніх систем з кількох причин: по-перше, вона забезпечує крос-платформну доступність — користувач може працювати з системою з будь-якого пристрою, що має браузер, незалежно від операційної системи; по-друге, спрощується процес оновлення та супроводу — нова версія розгортається на сервері і одразу стає доступною всім користувачам; по-третє, знижуються вимоги до апаратного забезпечення клієнтських машин, що особливо важливо в умовах обмежених фінансових ресурсів закладів освіти.

Для розуміння місця системи, що розробляється у межах цієї кваліфікаційної роботи, у загальному ландшафті освітніх ІТ-рішень доцільно розглянути основні типи інформаційних систем, які функціонують у сфері вищої освіти. У світовій практиці виділяють кілька основних класів таких систем.

Системи управління навчанням (Learning Management System, LMS) призначені для організації та проведення навчального процесу, передусім — для створення електронних курсів, поширення навчальних матеріалів, взаємодії викладача зі студентами, оцінювання їхньої роботи. До найвідоміших представників цього класу належать Moodle, Canvas, Blackboard, Brightspace. Основний акцент LMS — це навчальний контент і його доставка студентам.

Студентські інформаційні системи (Student Information System, SIS) — це системи обліку, що зберігають персональні дані здобувачів освіти, інформацію про навчальний план, академічні досягнення, фінансові розрахунки, статус навчання. Класичними прикладами є Banner від Ellucian, PeopleSoft Campus

Solutions, Workday Student. На відміну від LMS, SIS орієнтовані на адміністративні функції, а не на безпосередній навчальний процес.

Автоматизовані системи управління закладом вищої освіти (АСУ ЗВО) — це комплексні рішення, що інтегрують функції LMS, SIS та інших підсистем (документообіг, кадри, фінанси) в єдине середовище. В Україні значного поширення набули АСУ власної розробки, створені силами ІТ-підрозділів окремих ЗВО, оскільки готові комерційні рішення часто погано адаптовані до специфіки української освітньої системи, її нормативної бази та документообігу.

Системи підтримки спільної роботи (Collaboration Platforms) — Microsoft Teams, Google Workspace, Zoom, що активно використовувалися у період пандемії COVID-19 і продовжують застосовуватися сьогодні для проведення онлайн-занять, командної роботи студентів, комунікації викладачів і адміністрації. Ці системи не є спеціалізованими освітніми, але виконують значну частину функцій підтримки освітнього процесу.

Окремо слід виділити допоміжні системи — електронні бібліотеки, репозиторії наукових праць, антиплагіатні системи (Unicheck, StrikePlagiarism), системи проктрингу для онлайн-екзаменів, інструменти створення електронних портфоліо тощо. Ці системи, як правило, інтегруються з основними платформами через API.

Розглядаючи класифікацію інформаційних систем підтримки освітнього процесу, можна виділити кілька класифікаційних ознак. За способом розгортання розрізняють локальні системи (on-premises), хмарні (cloud-based, SaaS) та гібридні. За моделлю ліцензування — комерційні (Blackboard, Canvas), безкоштовні з відкритим кодом (Moodle, Open edX, Sakai) та внутрішні розробки. За широтою функціоналу — універсальні комплексні платформи та спеціалізовані вузькоспрямовані рішення. За цільовою аудиторією — системи для шкіл (K-12), для вищої освіти, для корпоративного навчання, для масових відкритих онлайн-курсів (МООС).

Сучасні тенденції розвитку інформаційних систем у сфері освіти зумовлені кількома факторами. По-перше, це зростання вимог до мобільності — користувачі очікують можливості повноцінно працювати з системою як з настільного, так і з мобільного пристрою. Це призводить до домінування адаптивного дизайну (responsive design) та концепції "Mobile First" при розробці нових рішень. По-друге, інтеграція технологій штучного інтелекту — від чат-ботів-помічників до адаптивного навчання, що автоматично підлаштовується під рівень підготовки конкретного студента. По-третє, перехід до мікросервісної архітектури, яка дозволяє розгортати окремі компоненти системи незалежно один від одного та масштабувати їх відповідно до навантаження. По-четверте, посилення вимог до інформаційної безпеки та захисту персональних даних відповідно до європейського регламенту GDPR та українського законодавства про захист персональних даних.

У контексті специфіки українських закладів вищої освіти необхідно враховувати низку додаткових факторів. Освітній процес регулюється низкою нормативних документів — Законом України "Про вищу освіту", Положенням про організацію освітнього процесу у ЗВО, стандартами вищої освіти за спеціальностями. Це породжує специфічні вимоги до структури та обсягу даних: наявність обов'язкових реквізитів (галузь знань, спеціальність, освітня програма, форма навчання, курс, група), специфічна термінологія (здобувач вищої освіти, кредити ЄКТС, освітні компоненти), особливості документообігу (заяви, накази, протоколи). Окрім того, мовна локалізація — інтерфейс має бути доступним українською мовою — є необхідною, але не завжди забезпечена в готових міжнародних платформах на належному рівні.

З огляду на викладене, інформаційна система, що розробляється у межах цієї кваліфікаційної роботи, належить до класу спрощених веборієнтованих систем підтримки освітнього процесу, які поєднують елементи функціоналу LMS (управління дисциплінами) та АСУ ЗВО (управління користувачами та

розкладом). Така система не претендує на роль повноцінної заміни існуючих платформ, проте може слугувати як початкова версія (Minimum Viable Product, MVP) для подальшого розширення функціональності, а також як демонстрація сучасного підходу до побудови веб-систем на основі стеку Python + JavaScript.

## **1.2. Огляд та порівняльний аналіз існуючих платформ**

Перш ніж переходити до проєктування власної системи, доцільно провести аналіз існуючих платформ, які найчастіше використовуються у закладах вищої освіти України та світу. Такий аналіз дозволяє виявити сильні та слабкі сторони наявних рішень, визначити ключові функціональні елементи, які мають бути обов'язково реалізовані у нових системах, а також обґрунтувати доцільність розробки альтернативного рішення.

Платформа Moodle (Modular Object-Oriented Dynamic Learning Environment) є однією з найпопулярніших систем управління навчанням у світі. Розроблена австралійським програмістом Мартіном Дугіамасом у 2002 році, нині Moodle підтримує понад 250 мільйонів користувачів у понад 240 країнах. Платформа поширюється під ліцензією GPL, що означає її повну безкоштовність для використання та можливість модифікації коду. Moodle написана мовою програмування PHP і використовує реляційні бази даних — MySQL, MariaDB, PostgreSQL.

Серед основних функціональних можливостей Moodle — створення електронних курсів з різноманітним мультимедійним контентом, тестові завдання з широким спектром типів питань, форуми для обговорення, чати, журнали оцінок, модуль завдань з можливістю прийому і перевірки робіт, антиплагіатна перевірка через інтегровані зовнішні сервіси, груповий менеджмент, ролі та права доступу. Платформа має дуже активну спільноту розробників, що створили тисячі плагінів для розширення базової функціональності. Підтримуються формати освітніх стандартів SCORM, AICC,

IMS Content Package, що забезпечує сумісність з матеріалами, створеними в інших системах.

Серед недоліків Moodle, особливо в контексті української освіти, варто відзначити такі. По-перше, користувацький інтерфейс платформи, незважаючи на постійні редизайни, залишається переобтяженим, що ускладнює навігацію для нових користувачів. По-друге, технічний стек на основі PHP вважається застарілим у порівнянні з сучасними альтернативами на основі Python, JavaScript або Go — це впливає на швидкодію, зручність розробки та підтримки. По-третє, налаштування Moodle під специфіку конкретного ЗВО потребує значних зусиль адміністратора або залучення розробників, що збільшує сукупну вартість володіння. По-четверте, інтеграція з внутрішніми системами обліку студентів закладу часто потребує додаткової розробки.

Платформа Google Classroom — це безкоштовна служба від компанії Google, інтегрована в екосистему Google Workspace for Education. Запущена у 2014 році, Classroom орієнтована передусім на спрощену взаємодію викладача і студентів — створення завдань, розповсюдження матеріалів, оцінювання робіт. Сильною стороною Classroom є інтеграція з іншими сервісами Google: Drive (файлове сховище), Docs/Sheets/Slides (офісні застосунки), Meet (відеоконференції), Calendar (планування). Це дозволяє побудувати повний процес дистанційного навчання без необхідності переключення між різними системами.

Однак Google Classroom має суттєві обмеження, особливо для закладів вищої освіти. Функціональність системи свідомо спрощена — вона не призначена для управління складною структурою освітніх програм, що включає десятки дисциплін, складний розклад, групові навчальні плани. Система не дозволяє кастомізувати або розширювати функціонал — користувачі обмежені тим, що пропонує Google. Залежність від інфраструктури Google створює питання щодо суверенітету даних — особливо актуально для державних установ,

які мають вимоги щодо зберігання даних на території України. Крім того, бізнес-модель Google передбачає можливість зміни умов використання, що створює довгостроковий ризик для закладів, які повністю покладаються на цю платформу.

Microsoft Teams з освітнім розширенням Teams for Education — це універсальна платформа для командної роботи та комунікації, активно адаптована для освітніх потреб. Teams надає функціонал відеоконференцій, чатів, спільної роботи з документами Office 365, ведення електронних класних кімнат, організації завдань і тестів. Інтеграція з Microsoft Office створює потужну екосистему, особливо у закладах, що вже використовують Microsoft-інфраструктуру.

Втім, аналогічно Google Classroom, Microsoft Teams не є спеціалізованою освітньою системою — це універсальний інструмент комунікації з функціями навчання як одним з багатьох сценаріїв використання. Складність інтерфейсу для нових користувачів, висока вартість повноцінних освітніх ліцензій, обмежені можливості адаптації під специфіку конкретного закладу — це головні стримуючі фактори. Як показав досвід ЗВО України під час пандемії та війни, Teams виконує функції допоміжного інструменту, але не може повноцінно замінити власну АСУ закладу.

Українські розробки АСУ ЗВО — це найрізноманітніша категорія систем, оскільки практично кожен великий заклад вищої освіти має власну розробку, створену внутрішньою командою або на замовлення. Прикладами таких систем є АСУ "Університет" Київського національного університету імені Тараса Шевченка, корпоративна інформаційна система Львівської політехніки, інформаційна система Київського політехнічного інституту, а також АСУ Державного податкового університету. Ці системи зазвичай тісно інтегровані з нормативно-правовою базою України та внутрішніми бізнес-процесами закладу — формування навчальних планів за стандартами МОН, генерація академічних довідок, ведення особових справ здобувачів освіти.

Загальною слабкою стороною українських АСУ є їх фрагментарність — як правило, кожен заклад має власну унікальну систему, що не сумісна з аналогічними системами інших закладів. Це створює складнощі при переведенні студентів між закладами, агрегації статистики на загальнонаціональному рівні, інтеграції з державними реєстрами (ЄДЕБО). Крім того, рівень технологічної зрілості таких систем сильно варіюється — деякі побудовані на застарілих технологіях (PHP 5, ASP Classic), мають проблеми з інтерфейсом, безпекою, продуктивністю.

Для систематизації результатів аналізу складемо порівняльну таблицю основних характеристик розглянутих платформ.

Таблиця 1.1 Порівняльний аналіз існуючих платформ підтримки освітнього процесу

| Критерій                | Moodle            | Google Classroom | MS Teams      | АСУ ЗВО                 |
|-------------------------|-------------------|------------------|---------------|-------------------------|
| Тип системи             | LMS               | Спрощена LMS     | Колаборативна | Комплексна АСУ          |
| Ліцензія                | Open Source (GPL) | Безкоштовна      | Комерційна    | Внутрішня розробка      |
| Технологічний стек      | PHP + MySQL       | Закритий         | Закритий      | Різний                  |
| Локалізація             | Висока            | Середня          | Висока        | Повна                   |
| Кастомізація            | Висока            | Низька           | Низька        | Висока                  |
| Складність впровадження | Висока            | Низька           | Середня       | Висока                  |
| Інтеграція з ЄДЕБО      | Відсутня          | Відсутня         | Відсутня      | Можлива                 |
| Сучасність UI           | Середня           | Висока           | Висока        | Низька-середня          |
| Відкритий API           | Є                 | Обмежений        | Є             | Залежить від реалізації |

Джерело: складено автором на основі офіційної документації платформ.

Аналіз існуючих рішень дозволяє сформулювати кілька важливих висновків, які впливають на обґрунтування доцільності розробки нової системи.

По-перше, жодна з існуючих платформ не є універсальним рішенням, що повністю задовольняло б усі потреби ЗВО — кожна має свої сильні та слабкі сторони. По-друге, готові комерційні платформи здебільшого є компромісними рішеннями, що не враховують специфіки конкретного закладу. По-третє, відкриті платформи (Moodle) надають значні можливості для адаптації, але вимагають значних ресурсів для впровадження та супроводу. По-четверте, власні розробки ЗВО часто страждають від технологічного застарівання та обмежених ресурсів на оновлення.

У зв'язку з цим виникає потреба у простих, легких у супроводі веборієнтованих системах, побудованих на сучасному технологічному стеку, які можуть бути швидко адаптовані під потреби конкретної кафедри або факультету. Такі системи не претендують на заміну великих корпоративних рішень, але можуть успішно вирішувати локальні задачі, бути основою для подальшого розвитку, а також слугувати навчальним прикладом для студентів комп'ютерних спеціальностей.

### **1.3. Обґрунтування вибору технологічного стеку для розробки веборієнтованої інформаційної системи**

Вибір технологічного стеку є ключовим рішенням на початковому етапі розробки будь-якого програмного продукту, оскільки воно визначає не лише швидкість і якість розробки, але й довгострокові можливості супроводу, масштабування та розвитку системи. Сучасний ландшафт технологій веб-розробки надзвичайно широкий — десятки мов програмування, сотні фреймворків, тисячі бібліотек і інструментів. Тому обґрунтування вибору має ґрунтуватися на чітких критеріях, що відповідають характеру і вимогам конкретного проєкту.

Перш ніж розглядати конкретні технології, доцільно сформулювати критерії оцінювання. Для веборієнтованої системи освітнього призначення, що розробляється в межах цієї кваліфікаційної роботи, важливими є такі критерії:

сучасність технології (підтримка спільнотою, активний розвиток, наявність документації); продуктивність розробки (швидкість створення нового функціоналу); продуктивність роботи системи (час відгуку, споживання ресурсів); безпека (захищеність від поширених типів атак, наявність вбудованих механізмів автентифікації, авторизації); вартість впровадження та супроводу (ліцензія, апаратні вимоги); крива навчання (час, необхідний для оволодіння технологією); розмір екосистеми (наявність готових бібліотек, інструментів, спільнота).

Веб-додаток, як правило, складається з двох основних компонентів — серверної частини (back-end), що відповідає за обробку даних, бізнес-логіку, роботу з базою даних, та клієнтської частини (front-end), що відповідає за відображення інформації та взаємодію з користувачем. Розглянемо вибір технологій для кожного компонента окремо.

Для серверної частини сучасними популярними варіантами є такі технологічні стеки. Стек на основі Java (фреймворки Spring Boot, Quarkus) — характеризується високою продуктивністю, надійністю, широким використанням у корпоративному середовищі. Однак Java вимагає більшого обсягу коду для реалізації простих задач (так званий "boilerplate code"), має складніший процес розгортання та потребує більших обчислювальних ресурсів. Стек на основі C# (.NET, ASP.NET Core) — потужний фреймворк від Microsoft з відмінною документацією та інструментами розробки. Має ті ж недоліки, що й Java, у плані складності, а також орієнтований передусім на Windows-екосистему, хоча .NET Core працює і на Linux. Стек на основі PHP (Laravel, Symfony) — найпоширеніший у веб-розробці історично, низький поріг входу. Однак мова PHP вважається менш сучасною, продуктивність нижча за компільовані мови, а архітектурні рішення часто застарілі. Стек на основі Node.js (Express, NestJS) — JavaScript на сервері, переваги єдиної мови для front-end та back-end. Однак асинхронна модель JavaScript ускладнює реалізацію складної бізнес-логіки. Стек

на основі Python (Django, Flask, FastAPI) — простота і виразність мови Python, найкраща екосистема в галузі обробки даних, машинного навчання, наукових обчислень, активна спільнота, чудова документація.

З-поміж зазначених варіантів для розробки веборієнтованої інформаційної системи освітнього призначення обрано стек на основі Python з фреймворком FastAPI. Обґрунтування вибору наступне.

Python як мова програмування характеризується простим, читабельним синтаксисом, що значно скорочує час розробки і полегшує супровід коду. У 2024 році Python посів перше місце у рейтингу мов програмування ТЮВЕ, що свідчить про найвищу популярність серед розробників. Python входить до навчальних програм комп'ютерних спеціальностей більшості закладів вищої освіти, що означає високу доступність розробників. Багата стандартна бібліотека, мільйони пакетів у репозиторії PyPI, інструменти для роботи з даними, веб-розробки, тестування — усе це робить Python універсальною мовою.

FastAPI — це сучасний веб-фреймворк для побудови API на Python, створений Себастьяном Раміресом у 2018 році. Незважаючи на відносну молодість, FastAPI швидко завоював популярність і нині використовується у проєктах Microsoft, Netflix, Uber та багатьох інших компаній. Ключові переваги FastAPI такі: висока продуктивність — за результатами незалежних тестів TechEmpower, FastAPI входить до числа найшвидших Python-фреймворків, наближаючись за швидкістю до Node.js та Go; вбудована підтримка стандарту OpenAPI — фреймворк автоматично генерує інтерактивну документацію API на основі типів даних і анотацій коду; вбудована валідація даних на основі бібліотеки Pydantic — некоректні дані відсікаються ще до того, як потрапити в бізнес-логіку; підтримка асинхронного виконання на основі ASGI-сервера, що дозволяє ефективно обробляти велику кількість одночасних запитів; інтеграція з системами автентифікації OAuth 2.0, JWT з мінімальними зусиллями; невелика крива навчання порівняно з більш важкими фреймворками типу Django.

Для зберігання даних в системі обрано реляційну базу даних SQLite. Обґрунтування вибору: SQLite — це вбудована, безсерверна СУБД, що зберігає всю базу в одному файлі. Вона не потребує окремого процесу-сервера і є найпоширенішою у світі базою даних — використовується в мобільних застосунках Android та iOS, браузерях, операційних системах. Для системи з невеликим або середнім навантаженням (до сотень одночасних користувачів) SQLite забезпечує достатню продуктивність, при цьому значно спрощує процес розгортання — не потрібно встановлювати і налаштовувати окремий сервер бази даних. Це особливо актуально на етапі розробки, тестування та для невеликих освітніх установ. У разі необхідності перехід на повноцінні СУБД — PostgreSQL чи MySQL — здійснюється тривіально, оскільки використовується ORM SQLAlchemy, що абстрагує конкретний діалект SQL.

SQLAlchemy — це найпопулярніша ORM-бібліотека для Python, що забезпечує об'єктно-реляційне відображення між Python-класами та таблицями реляційної бази даних. Використання ORM дозволяє писати запити до бази даних мовою програмування, не занурюючись у синтаксис SQL для кожного діалекту, забезпечує захист від SQL-ін'єкцій завдяки параметризованим запитам, спрощує міграції схеми бази даних.

Для клієнтської частини сучасними варіантами є кілька JavaScript-бібліотек і фреймворків. Бібліотека React, розроблена Meta (Facebook) у 2013 році, нині є найпопулярнішим інструментом для побудови користувацьких інтерфейсів. Згідно з опитуванням розробників Stack Overflow Developer Survey 2024, React використовують понад 40% розробників фронтенду. Альтернативи — Vue.js (більш простий поріг входу, але менш поширений у корпоративному середовищі), Angular (повноцінний фреймворк від Google, потужний, але складніший), Svelte (молода, інноваційна технологія, поки що з меншою екосистемою).

Для проєкту обрано React з таких причин. По-перше, найбільша екосистема — тисячі готових компонентів, бібліотек, інструментів, навчальних матеріалів. По-друге, компонентний підхід React дозволяє розділяти інтерфейс на невеликі повторно використовувані блоки, що покращує підтримку і розширюваність коду. По-третє, односпрямований потік даних, концепція віртуального DOM, хуки (Hooks) — сучасні архітектурні рішення, які роблять код передбачуваним і ефективним. По-четверте, React широко вивчається в навчальних закладах, що забезпечує доступність матеріалів для самостійного освоєння.

Як збірник модулів і dev-сервер обрано Vite — сучасний інструмент, що замінює традиційний Webpack. Vite використовує нативні ES-модулі браузера для розробки, що забезпечує миттєвий запуск dev-сервера незалежно від розміру проєкту, а для production-збірки використовує оптимізований Rollup. Швидкість розробки з Vite суттєво вища за традиційні рішення.

Для стилізації інтерфейсу використано фреймворк TailwindCSS — utility-first CSS-фреймворк, що надає набір базових класів для побудови інтерфейсу безпосередньо в HTML-розмітці. Цей підхід має кілька переваг: відсутність необхідності писати кастомний CSS для більшості сценаріїв, узгодженість дизайну завдяки використанню токенів дизайн-системи (кольори, відступи, типографіка), невеликий розмір production-білду завдяки автоматичному видаленню невикористаних стилів. TailwindCSS активно використовується у сучасних проєктах GitHub, Vercel, Shopify.

Для маршрутизації в SPA обрано React Router — стандарт де-факто для побудови багатосторінкових односторінкових додатків (SPA) на React. React Router забезпечує клієнтську маршрутизацію, тобто перехід між "сторінками" відбувається без перезавантаження браузера, що значно покращує користувацький досвід.

Для виконання HTTP-запитів до серверного API обрано бібліотеку Axios — популярний HTTP-клієнт для JavaScript з зручним API, підтримкою перехоплювачів (interceptors), автоматичною серіалізацією JSON, обробкою помилок. Axios легко інтегрується з механізмом автентифікації JWT через перехоплювачі, які автоматично додають токен до кожного запиту.

Як механізм автентифікації обрано стандарт JSON Web Token (JWT, RFC 7519). JWT — це компактний, самодостатній спосіб передачі інформації між сторонами у вигляді JSON-об'єкта. Токен підписується криптографічним секретом (HMAC) або парою ключів (RSA, ECDSA), що гарантує його цілісність. Перевагами JWT для веборієнтованих систем є: відсутність необхідності зберігання станів сесій на сервері (stateless), що спрощує горизонтальне масштабування; самодостатність — токен містить усі необхідні дані про користувача, включно з його ідентифікатором і ролями; стандартність — підтримується всіма сучасними мовами програмування і бібліотеками. На клієнтській стороні JWT-токен зберігається у sessionStorage браузера, на серверній — підписується секретом у конфігурації додатку і перевіряється при кожному захищеному запиті.

Для хешування паролів обрано алгоритм bcrypt — криптографічно стійка функція, спеціально призначена для зберігання паролів. Bcrypt включає сіль (salt) та налаштовуваний фактор складності (work factor), що дозволяє підвищувати стійкість до атак повним перебором у міру зростання обчислювальних потужностей злоумисників.

Для перевірки коректності даних на серверній стороні використовується бібліотека Pydantic — стандарт де-факто для валідації даних у сучасних Python-проектах. Pydantic базується на анотаціях типів Python (PEP 484), що дозволяє описувати структуру очікуваних даних у вигляді звичайних класів. Валідація відбувається автоматично при отриманні запиту, некоректні дані повертають детальну помилку у форматі JSON.

Підсумовуючи, технологічний стек для розробки веборієнтованої інформаційної системи підтримки освітнього процесу включає такі компоненти. Серверна частина — Python 3.12 у поєднанні з фреймворком FastAPI, реляційна база даних SQLite, ORM SQLAlchemy, бібліотеки Pydantic для валідації, python-jose для JWT, bcrypt для хешування паролів, ASGI-сервер Uvicorn для запуску додатку. Клієнтська частина — JavaScript (ES2022) з бібліотекою React 18, маршрутизація через React Router 6, стилізація через TailwindCSS 3, HTTP-клієнт Axios, інструмент збірки Vite 5. Архітектурний стиль взаємодії між клієнтом і сервером — REST (Representational State Transfer), формат обміну даними — JSON. Методологія розробки — інкрементальна, з поетапним нарощуванням функціональності від мінімально життєздатного продукту (MVP) до повноцінної системи.

Обраний стек відповідає сучасним тенденціям веб-розробки, забезпечує високу продуктивність розробки та роботи системи, має відкриті ліцензії (отже, не вимагає ліцензійних відрахувань), широко представлений у навчальних програмах закладів вищої освіти і має активну спільноту розробників. Наступні розділи кваліфікаційної роботи присвячені детальному проектуванню та реалізації системи на основі обраного стеку.

Висновки до розділу 1. У першому розділі досліджено теоретичні засади побудови веборієнтованих інформаційних систем для підтримки освітнього процесу. Визначено сутність таких систем як соціотехнічних утворень, що автоматизують процеси планування, організації, реалізації та контролю навчальної діяльності. Систематизовано класифікацію освітніх інформаційних систем за функціональним призначенням (LMS, SIS, АСУ, колаборативні платформи) та іншими ознаками. Проведено порівняльний аналіз чотирьох найпоширеніших класів платформ — Moodle, Google Classroom, Microsoft Teams, українські АСУ ЗВО — та виявлено їх сильні і слабкі сторони. Обґрунтовано вибір технологічного стеку для розробки власної системи: Python + FastAPI +

SQLite + SQLAlchemy для серверної частини, React + Vite + TailwindCSS + Axios для клієнтської частини, JWT для автентифікації. Обраний стек відповідає сучасним тенденціям веб-розробки і забезпечує оптимальне співвідношення між продуктивністю розробки, продуктивністю роботи системи, безпекою та вартістю впровадження.

## РОЗДІЛ 2. СИСТЕМНО-АНАЛІТИЧНЕ ПРОЄКТУВАННЯ ІНФОРМАЦІЙНОЇ СИСТЕМИ

### 2.1. Архітектура системи та функціональні вимоги

Етап системно-аналітичного проєктування є проміжною ланкою між теоретичним обґрунтуванням, виконаним у першому розділі, та безпосередньою програмною реалізацією, що буде описана в третьому розділі. На цьому етапі визначається структура майбутньої системи, її логічні компоненти, способи їх взаємодії, а також формалізується перелік функцій, які система повинна забезпечувати. Якісне виконання цього етапу значною мірою визначає успіх усієї розробки, оскільки виправлення архітектурних прорахунків після початку реалізації коштує значно дорожче, ніж їх виявлення на стадії проєктування.

Розробка веборієнтованої інформаційної системи підтримки освітнього процесу розпочинається з аналізу предметної області та формалізації функціональних вимог. У межах кваліфікаційної роботи передбачено реалізацію базового функціоналу системи, що відповідає концепції мінімально життєздатного продукту (MVP) і охоплює три ключові домени: управління користувачами з диференціацією ролей, ведення каталогу навчальних дисциплін та управління розкладом занять.

Функціональні вимоги до системи можна сформулювати у вигляді переліку основних можливостей, що мають бути доступні різним категоріям користувачів. Узагальнено функціональні вимоги представлено у вигляді таблиці.

Таблиця 2.1 Функціональні вимоги до інформаційної системи

| № | Функція                     | Студент | Викладач | Адміністратор |
|---|-----------------------------|---------|----------|---------------|
| 1 | Реєстрація в системі        | +       | +        | —             |
| 2 | Авторизація (вхід)          | +       | +        | +             |
| 3 | Перегляд особистого профілю | +       | +        | +             |
| 4 | Перегляд каталогу дисциплін | +       | +        | +             |

| №  | Функція                           | Студент | Викладач | Адміністратор |
|----|-----------------------------------|---------|----------|---------------|
| 5  | Створення нової дисципліни        | —       | +        | +             |
| 6  | Редагування власних дисциплін     | —       | +        | +             |
| 7  | Редагування будь-яких дисциплін   | —       | —        | +             |
| 8  | Видалення дисциплін               | —       | —        | +             |
| 9  | Перегляд розкладу занять          | +       | +        | +             |
| 10 | Фільтрація розкладу за днем тижня | +       | +        | +             |
| 11 | Створення запису розкладу         | —       | —        | +             |
| 12 | Редагування записів розкладу      | —       | —        | +             |
| 13 | Видалення записів розкладу        | —       | —        | +             |
| 14 | Перегляд списку користувачів      | —       | —        | +             |
| 15 | Створення користувачів вручну     | —       | —        | +             |
| 16 | Видалення користувачів            | —       | —        | +             |

Джерело: розроблено автором.

Окрім функціональних вимог, система має задовольняти ряд нефункціональних вимог. До них належать: вимоги до продуктивності — час відгуку API на типовий запит не повинен перевищувати 200 мілісекунд при роботі з базою даних обсягом до десятків тисяч записів; вимоги до безпеки — паролі користувачів не повинні зберігатися у відкритому вигляді, доступ до захищених ресурсів повинен здійснюватися лише після успішної автентифікації, передача даних має шифруватися протоколом TLS; вимоги до зручності використання — інтерфейс має бути адаптивним і коректно відображатися на пристроях різних розмірів від 320 пікселів завширшки, мова інтерфейсу — українська, основні дії повинні виконуватися не більш ніж за три кліки; вимоги до сумісності — система має працювати в актуальних версіях основних браузерів (Chrome, Firefox, Safari, Edge); вимоги до розгортання — серверна частина повинна запускатися однією командою без складних попередніх налаштувань.

З точки зору архітектури, інформаційна система реалізована за класичною трирівневою (трирівневою) моделлю, що включає рівень представлення, рівень бізнес-логіки та рівень доступу до даних. Така архітектура є загальноприйнятою в галузі розробки веб-систем і має ряд переваг: чіткий поділ відповідальності між компонентами, можливість незалежної розробки і тестування кожного рівня, легкість заміни одного з компонентів без впливу на інші, можливість горизонтального масштабування окремих рівнів за необхідності.

Перший рівень — рівень представлення (Presentation Layer) — реалізований у вигляді односторінкового веб-додатку (SPA, Single Page Application), розробленого на основі бібліотеки React. Цей рівень виконується безпосередньо у браузері користувача і відповідає за відображення інтерфейсу, обробку дій користувача (натискання кнопок, заповнення форм), а також формування запитів до серверної частини. Важлива особливість сучасних SPA — після першого завантаження сторінки переходи між розділами додатку відбуваються без перезавантаження сторінки, що забезпечує швидкий і плавний користувацький досвід, аналогічний до десктопних додатків.

Другий рівень — рівень бізнес-логіки (Application Layer) — реалізований у вигляді HTTP-сервера на основі Python-фреймворку FastAPI. Цей рівень виконується на серверному обладнанні і відповідає за обробку запитів від клієнтського додатку, перевірку прав доступу, виконання бізнес-операцій (створення, читання, оновлення, видалення сутностей), генерацію JWT-токенів для автентифікації, валідацію вхідних даних. Взаємодія між клієнтським і серверним рівнями здійснюється через REST API у форматі JSON.

Третій рівень — рівень доступу до даних (Data Access Layer) — реалізований у вигляді реляційної бази даних SQLite з прошарком ORM SQLAlchemy. Цей рівень відповідає за збереження стану системи: дані про користувачів, дисципліни, записи розкладу. Завдяки використанню ORM, серверна частина працює з даними у вигляді Python-об'єктів, не вдаючись

безпосередньо до SQL-запитів, що спрощує розробку і робить код більш переносним.

Для наочного представлення архітектури складемо узагальнену схему взаємодії компонентів системи.

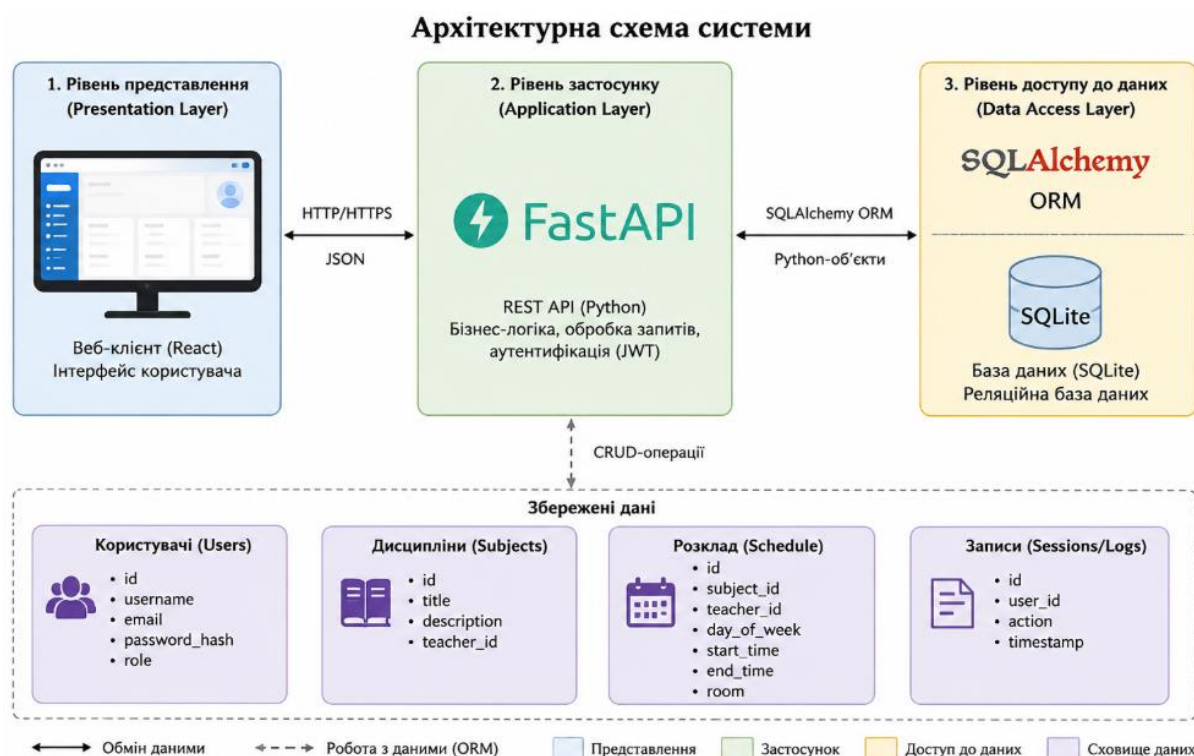


Рис.

## 2.1. Архітектурна схема системи

Для опису сценаріїв взаємодії користувачів із системою застосуємо нотацію use-case діаграм, поширену в сфері проєктування інформаційних систем. У системі визначено три ролі акторів: Студент, Викладач, Адміністратор. Між ними існують відносини спадкування, оскільки кожна наступна роль має ширші права і включає всі можливості попередньої. Адміністратор володіє повним набором функцій; Викладач має доступ до всіх функцій Студента плюс можливості з управління дисциплінами; Студент має базові права на перегляд інформації.

Розглянемо основні сценарії використання системи. Сценарій реєстрації нового користувача починається з відкриття сторінки реєстрації, заповнення форми (прізвище та ім'я, електронна пошта, пароль, вибір ролі — студент або

викладач), відправки даних на сервер. Сервер перевіряє унікальність електронної пошти, хешує пароль за допомогою bcrypt, створює запис у таблиці users, генерує JWT-токен і повертає його разом із даними створеного користувача. Клієнт зберігає токен у sessionStorage та автоматично авторизується.

Сценарій авторизації передбачає введення електронної пошти та пароля, перевірку відповідності хешу паролю в базі даних, генерацію JWT-токена при успішній автентифікації. У разі помилки користувач отримує повідомлення про невірні дані входу.

Сценарій перегляду розкладу: автентифікований користувач переходить на сторінку розкладу, клієнт надсилає GET-запит до API, сервер повертає список занять із приєднаною інформацією про дисципліни та викладачів, клієнт відображає дані у вигляді тижневої сітки з виділенням поточного дня тижня. Опціонально користувач може застосувати фільтр за днем тижня — клієнт додає параметр запиту day, сервер фільтрує дані відповідно.

Сценарій створення дисципліни доступний лише користувачам з ролями викладача або адміністратора. Користувач натискає кнопку "Додати предмет", система відкриває модальне вікно з формою (назва, код, опис, кількість кредитів). Після заповнення і підтвердження клієнт надсилає POST-запит до API. Сервер перевіряє унікальність коду дисципліни, створює запис, при цьому якщо запит зробив викладач — автоматично призначає його власником дисципліни. У відповіді повертається створений запис, який клієнт додає до локального списку без повного перезавантаження сторінки.

Сценарій управління користувачами доступний лише адміністратору. Адміністратор переходить на сторінку "Користувачі", бачить таблицю з усіма обліковими записами системи, може створювати нових користувачів з будь-якою роллю (включно з адміністратором, на відміну від загальної форми реєстрації) або видаляти існуючих, окрім власного облікового запису.

Особливу увагу слід приділити моделі безпеки системи. Інформаційна безпека реалізується на кількох рівнях. На рівні автентифікації — використання JWT-токенів зі строком дії 24 години, паролі зберігаються у вигляді bcrypt-хешів зі сіллю. На рівні авторизації — рольова модель з трьома ролями, перевірка ролі виконується на серверній стороні через залежності FastAPI, спроба виконати дію без потрібної ролі повертає HTTP-статус 403 Forbidden. На рівні передачі даних — рекомендується розгортання за HTTPS у виробничому середовищі, що забезпечує шифрування трафіку. На рівні валідації — всі вхідні дані перевіряються Pydantic-схемами, некоректні дані відсікаються на ранньому етапі обробки.

Структура серверного додатку організована за принципом розділення відповідальності. Файл `main.py` — точка входу, конфігурація FastAPI, підключення middleware та роутерів. Модуль `config.py` містить налаштування програми (URL бази даних, секрет JWT, дозволені CORS-домени). Модуль `database.py` відповідає за створення підключення до БД та надає функцію-залежність `get_db` для отримання сесії. Модуль `models.py` містить SQLAlchemy-моделі — Python-класи, що відображаються на таблиці бази даних. Модуль `schemas.py` — Pydantic-схеми для валідації запитів та форматування відповідей. Модуль `security.py` — функції для роботи з паролями та JWT-токенами, а також залежності для перевірки автентифікації та авторизації. Папка `routers` містить окремі файли для кожної логічної групи API-ендпоінтів: `auth.py`, `subjects.py`, `schedule.py`, `users.py`. Модуль `init_db.py` відповідає за створення таблиць та завантаження демо-даних при першому запуску.

Структура клієнтського додатку також організована за принципом розділення відповідальності. Файл `main.jsx` — точка входу React-додатку, обгортання в провайдери (`BrowserRouter`, `AuthProvider`). Файл `App.jsx` — конфігурація маршрутизатора. Папка `pages` містить компоненти-сторінки: `Login`, `Register`, `Dashboard`, `Subjects`, `Schedule`, `Users`. Папка `components` — повторно

використовувані компоненти, такі як `Navbar` та `ProtectedRoute`. Папка `context` містить `React`-контекст `AuthContext` для глобального управління станом авторизації. Файл `api.js` — обгортка над `Axios` з налаштуванням базового URL та перехоплювачів. Файл `index.css` — глобальні стилі та налаштування `TailwindCSS`.

## 2.2. Проєктування моделі даних та схеми бази даних

Модель даних — це формальне представлення сутностей предметної області, їхніх атрибутів та зв'язків між ними. Якісно спроектована модель даних є основою стабільної та масштабованої інформаційної системи. У межах цієї роботи для проєктування обрано реляційну модель даних як найбільш поширену та добре підтримувану більшістю сучасних СУБД.

Аналіз предметної області дозволив виділити три основні сутності, що складають ядро інформаційної системи: Користувач, Дисципліна та Запис розкладу. Розглянемо кожну з них детально.

Сутність "Користувач" представляє осіб, які мають доступ до системи — здобувачів освіти, викладачів, адміністраторів. Атрибути сутності: унікальний ідентифікатор (`id`), електронна пошта (`email`), хеш пароля (`password_hash`), повне ім'я (`full_name`), роль (`role`), дата створення (`created_at`). Електронна пошта виконує дві функції — слугує логіном для входу та забезпечує унікальну ідентифікацію користувача. Зважаючи на чутливість, пароль ніколи не зберігається у відкритому вигляді — у базу записується лише його криптографічний хеш, обчислений за алгоритмом `bcrypt` з випадковою сіллю. Роль визначає права доступу і обмежена трьома значеннями через `CHECK`-обмеження бази даних.

Сутність "Дисципліна" представляє навчальний предмет, що викладається в межах освітньої програми. Атрибути: унікальний ідентифікатор (`id`), назва (`name`), унікальний код (`code`), описова частина (`description`), посилання на

викладача (`teacher_id` як зовнішній ключ до користувача), кількість кредитів ЄКТС (`credits`), дата створення (`created_at`). Код дисципліни використовується для коротких посилань (наприклад, CS-301, MA-102) і також має бути унікальним. Посилання на викладача необов'язкове — дисципліна може існувати без призначеного викладача (наприклад, при початковому імпорті навчального плану). При видаленні викладача поле `teacher_id` у пов'язаних дисциплін встановлюється у `NULL` завдяки правилу `ON DELETE SET NULL`, що зберігає цілісність даних і не призводить до каскадного видалення дисциплін.

Сутність "Запис розкладу" представляє конкретне заняття — пару певної дисципліни в певний день тижня, у певний час, у певній аудиторії, для певної групи. Атрибути: унікальний ідентифікатор (`id`), посилання на дисципліну (`subject_id`), день тижня (`day_of_week`, числове значення від 1 до 7), час початку (`start_time` у форматі HH:MM), час закінчення (`end_time`), номер аудиторії (`classroom`), назва групи (`group_name`), дата створення. Поточна реалізація використовує спрощену модель розкладу — заняття повторюються щотижня, без урахування чисельника/знаменника та конкретних дат. У майбутніх версіях системи модель може бути розширена для підтримки складніших графіків. При видаленні дисципліни всі пов'язані з нею записи розкладу видаляються каскадно (`ON DELETE CASCADE`), оскільки запис розкладу не має сенсу без прив'язки до конкретної дисципліни.

Зв'язки між сутностями: один-до-багатьох (1:N) між Користувач і Дисципліна — один викладач може вести багато дисциплін, але кожна дисципліна закріплена не більш ніж за одним викладачем; один-до-багатьох (1:N) між Дисципліна і Запис розкладу — одна дисципліна може мати багато записів у розкладі, але кожен запис прив'язаний до однієї конкретної дисципліни. ER-діаграма системи представлена нижче.

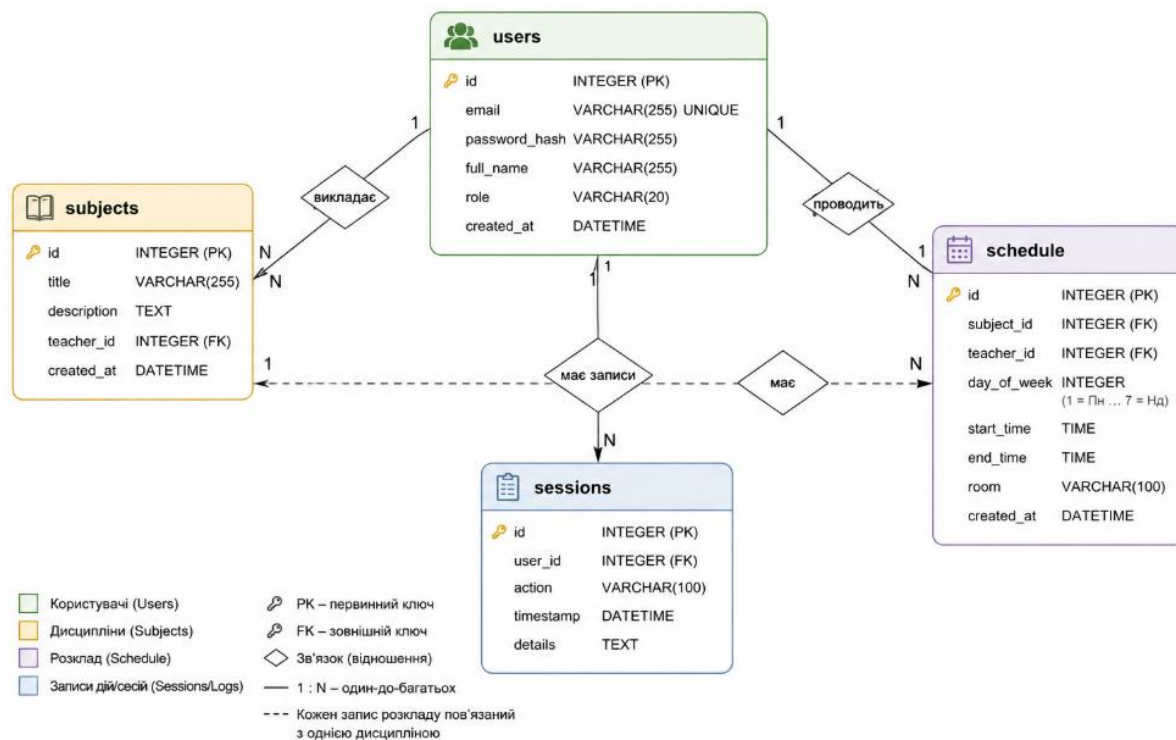


Рис.

## 2.2. ER-діаграма бази даних

DDL-визначення таблиць засобами SQL виглядає таким чином (фрагмент із init-скрипту):

```
sql
```

```
CREATE TABLE users (
    id          INTEGER PRIMARY KEY AUTOINCREMENT,
    email       VARCHAR(255) UNIQUE NOT NULL,
    password_hash VARCHAR(255) NOT NULL,
    full_name   VARCHAR(255) NOT NULL,
    role        VARCHAR(20) NOT NULL
                CHECK (role IN ('student','teacher','admin')),
    created_at  DATETIME DEFAULT CURRENT_TIMESTAMP
);
```

```
CREATE INDEX idx_users_email ON users(email);
```

```
CREATE TABLE subjects (
    id      INTEGER PRIMARY KEY AUTOINCREMENT,
    name     VARCHAR(255) NOT NULL,
    code     VARCHAR(20) UNIQUE NOT NULL,
    description TEXT,
    teacher_id INTEGER REFERENCES users(id) ON DELETE SET NULL,
    credits  INTEGER NOT NULL DEFAULT 3,
    created_at DATETIME DEFAULT CURRENT_TIMESTAMP
);
```

```
CREATE INDEX idx_subjects_code ON subjects(code);
CREATE INDEX idx_subjects_teacher ON subjects(teacher_id);
```

```
CREATE TABLE schedule (
    id      INTEGER PRIMARY KEY AUTOINCREMENT,
    subject_id INTEGER NOT NULL
        REFERENCES subjects(id) ON DELETE CASCADE,
    day_of_week INTEGER NOT NULL
        CHECK (day_of_week BETWEEN 1 AND 7),
    start_time VARCHAR(5) NOT NULL,
    end_time   VARCHAR(5) NOT NULL,
    classroom  VARCHAR(50),
    group_name VARCHAR(50),
    created_at DATETIME DEFAULT CURRENT_TIMESTAMP
);
```

```
CREATE INDEX idx_schedule_day ON schedule(day_of_week);
CREATE INDEX idx_schedule_subject ON schedule(subject_id);
```

Розглянемо ключові аспекти реалізації моделі даних детальніше. Цілісність даних забезпечується кількома механізмами: первинні ключі (PRIMARY KEY) гарантують унікальність кожного запису в межах таблиці; обмеження UNIQUE на полях email і code запобігає дублюванню; зовнішні ключі (FOREIGN KEY) забезпечують посилову цілісність; CHECK-обмеження валідують значення на рівні бази даних, що гарантує коректність навіть у разі помилок у програмному коді; обмеження NOT NULL контролює обов'язковість полів.

Індексація — важливий аспект продуктивності. Для оптимізації частих операцій пошуку створено індекси на полях, що використовуються для фільтрації та з'єднання таблиць: email — для швидкого пошуку користувача при авторизації; code — для перевірки унікальності коду дисципліни; teacher\_id — для отримання списку дисциплін певного викладача; day\_of\_week — для фільтрації розкладу за днем; subject\_id — для з'єднання записів розкладу з дисциплінами. Зазначимо, що SQLite автоматично створює індекси для полів PRIMARY KEY та UNIQUE, тому деякі індекси є технічно надлишковими, проте явне визначення підвищує читабельність схеми.

Нормалізація — у спроектованій схемі досягнуто третьої нормальної форми (3НФ). Кожна таблиця має чіткий первинний ключ, всі неключові атрибути функціонально залежать від первинного ключа, відсутні транзитивні залежності. Подальша нормалізація (наприклад, винесення поля group\_name в окрему таблицю груп) у поточній версії системи вважається передчасною оптимізацією і може бути виконана при розширенні функціоналу.

ORM-моделі для використання в Python-коді створено за допомогою SQLAlchemy 2.0 з використанням нового декларативного синтаксису на основі анотацій типів. Моделі описуються як класи, успадковані від Base, з полями типу Mapped, що дозволяє використовувати переваги системи статичної типізації. Ключовий фрагмент моделі User представлено нижче:

```
python
```

```

class User(Base):
    __tablename__ = "users"

    id: Mapped[int] = mapped_column(
        Integer, primary_key=True, autoincrement=True)
    email: Mapped[str] = mapped_column(
        String(255), unique=True, nullable=False, index=True)
    password_hash: Mapped[str] = mapped_column(
        String(255), nullable=False)
    full_name: Mapped[str] = mapped_column(
        String(255), nullable=False)
    role: Mapped[str] = mapped_column(
        String(20), nullable=False)
    created_at: Mapped[datetime] = mapped_column(
        DateTime, default=datetime.utcnow)

    subjects: Mapped[list["Subject"]] = relationship(
        "Subject", back_populates="teacher")

```

Атрибут `relationship` налаштовує автоматичне завантаження пов'язаних об'єктів — наприклад, при роботі з об'єктом `User` можна звертатися до `user.subjects` для отримання списку дисциплін, що ведуться даним викладачем. SQLAlchemy самостійно виконує необхідні SQL-запити з оптимізацією через JOIN-операції.

Особливу увагу слід приділити стратегії заповнення системи початковими даними. При першому запуску модуль `init_db.py` виконує створення таблиць (за допомогою методу `Base.metadata.create_all`) і завантажує демонстраційні дані: одного адміністратора, двох викладачів, одного студента, три дисципліни та п'ять

записів розкладу. Це забезпечує можливість одразу почати роботу з системою без необхідності вручну створювати початкові записи.

### **2.3. Проєктування REST API та механізмів автентифікації**

Інтерфейс прикладного програмування (API) є центральним елементом, що зв'язує клієнтську та серверну частини системи. Якість проєктування API безпосередньо впливає на зручність роботи фронтенд-розробників, продуктивність системи в цілому та можливості майбутньої інтеграції з іншими системами. У межах кваліфікаційної роботи для побудови API обрано архітектурний стиль REST.

Архітектурний стиль REST (Representational State Transfer), сформульований Роєм Філдіном у 2000 році в його докторській дисертації, базується на кількох ключових принципах. Принцип статесності — кожен запит від клієнта до сервера має містити всю необхідну інформацію для його обробки, сервер не зберігає стану клієнта між запитами, що значно спрощує масштабування системи. Принцип уніфікованого інтерфейсу — використання стандартних HTTP-методів (GET для читання, POST для створення, PUT/PATCH для оновлення, DELETE для видалення) і стандартних кодів відповіді. Принцип орієнтації на ресурси — кожна сутність системи має унікальний URL, операції з нею виконуються через відповідні HTTP-методи. Принцип шарової архітектури — клієнт не повинен залежати від внутрішньої структури сервера, що дозволяє додавати проміжні рівні (кеш, балансувальники навантаження) без зміни клієнтського коду.

Структура URL-адрес побудована за ієрархічним принципом: загальний префікс /api для всіх endpoint-ів, далі — назва ресурсу у множині (subjects, schedule, users), далі — за необхідності ідентифікатор конкретного запису та

підресурси. Така структура є добре зрозумілою та узгоджується з рекомендаціями галузі.

Повний перелік API-ендпоінтів системи наведено в табл. 2.2.

Таблиця 2.2 Специфікація REST API системи

| Метод  | URL                | Опис                            | Доступ                |
|--------|--------------------|---------------------------------|-----------------------|
| POST   | /api/auth/register | Реєстрація нового користувача   | Публічний             |
| POST   | /api/auth/login    | Вхід у систему                  | Публічний             |
| GET    | /api/auth/me       | Дані поточного користувача      | Авторизований         |
| GET    | /api/subjects      | Список усіх дисциплін           | Авторизований         |
| GET    | /api/subjects/{id} | Деталі конкретної дисципліни    | Авторизований         |
| POST   | /api/subjects      | Створити дисципліну             | teacher, admin        |
| PUT    | /api/subjects/{id} | Оновити дисципліну              | teacher (свої), admin |
| DELETE | /api/subjects/{id} | Видалити дисципліну             | admin                 |
| GET    | /api/schedule      | Список занять (опц. фільтр day) | Авторизований         |
| POST   | /api/schedule      | Створити запис розкладу         | admin                 |
| PUT    | /api/schedule/{id} | Оновити запис розкладу          | admin                 |
| DELETE | /api/schedule/{id} | Видалити запис розкладу         | admin                 |
| GET    | /api/users         | Список усіх користувачів        | admin                 |
| POST   | /api/users         | Створити користувача            | admin                 |
| DELETE | /api/users/{id}    | Видалити користувача            | admin                 |
| GET    | /                  | Health-check (статус сервера)   | Публічний             |

Джерело: розроблено автором.

Усього в системі реалізовано 16 ендпоінтів, що покривають усі функціональні вимоги. Розглянемо найважливіші з них детальніше.

Ендпоінт POST /api/auth/login приймає JSON-об'єкт із полями email і password, повертає об'єкт із токеном доступу та інформацією про користувача. У разі некоректних даних повертається HTTP-статус 401 Unauthorized з відповідним повідомленням. Структура запиту:

json

```
{
  "email": "admin@edu.local",
  "password": "admin123"
}
```

Структура успішної відповіді:

```
json
{
  "access_token": "eyJhbGciOiJIUzI1NiIs...\"",
  "token_type": "bearer",
  "user": {
    "id": 1,
    "email": "admin@edu.local",
    "full_name": "Адміністратор системи",
    "role": "admin",
    "created_at": "2025-09-01T10:00:00"
  }
}
```

Ендпоінт `GET /api/schedule` повертає список усіх записів розкладу з приєднаною інформацією про дисципліни та викладачів. Підтримуються необов'язкові параметри запиту: `day` (фільтр за днем тижня, 1-7) та `group` (фільтр за назвою групи). Завдяки використанню `SQLAlchemy` із `joinedload`, серверна сторона виконує єдиний SQL-запит з JOIN-операціями, що дозволяє уникнути N+1 проблеми — типового антипатерну, коли для кожного запису виконується окремий запит за пов'язаними даними.

Механізм автентифікації побудовано на стандарті `JSON Web Token (JWT, RFC 7519)`. JWT — це компактний URL-безпечний спосіб представлення запитів, що передаються між двома сторонами. Токен складається з трьох частин, розділених крапками: заголовок (`header`), корисне навантаження (`payload`) та

підпис (signature). Заголовок містить тип токена та використаний криптографічний алгоритм. Корисне навантаження містить набір тверджень (claims) про користувача — у нашому випадку це ідентифікатор (sub), роль (role) та час закінчення дії (exp). Підпис забезпечує цілісність токена — будь-яка зміна заголовка або payload без знання секретного ключа призведе до неспівпадіння підпису і відхилення токена сервером.

Алгоритм роботи з JWT у системі такий: при успішній автентифікації або реєстрації сервер генерує токен, підписує його секретним ключем (зберігається в конфігурації) за допомогою алгоритму HS256, повертає токен клієнту. Клієнт зберігає токен у sessionStorage браузера та додає до кожного наступного захищеного запиту в заголовок Authorization у форматі "Bearer <token>". Сервер при отриманні запиту перевіряє наявність токена, верифікує його підпис, перевіряє строк дії, отримує ідентифікатор користувача з payload, завантажує дані користувача з бази даних. Якщо токен відсутній, недійсний або прострочений — повертається HTTP-статус 401. Якщо токен валідний, але роль користувача не дозволяє виконати запитану дію — повертається HTTP-статус 403.

Реалізація JWT у FastAPI використовує бібліотеку python-jose і вбудовану систему залежностей. Створено залежність get\_current\_user, що автоматично виконує перевірку токена для всіх захищених ендпоінтів, та фабрику залежностей require\_role(\*allowed\_roles), що перевіряє роль користувача:

```
python
def get_current_user(
    token: str = Depends(oauth2_scheme),
    db: Session = Depends(get_db),
) -> User:
    credentials_exception = HTTPException(
        status_code=401,
```

```

        detail="Не вдалося підтвердити дані для входу",
    )
    try:
        payload = jwt.decode(token, settings.JWT_SECRET,
                               algorithms=[settings.JWT_ALGORITHM])
        user_id = int(payload.get("sub"))
    except (JWTError, ValueError):
        raise credentials_exception
    user = db.query(User).filter(User.id == user_id).first()
    if user is None:
        raise credentials_exception
    return user

```

```

def require_role(*allowed_roles: str):
    def role_checker(
        current_user: User = Depends(get_current_user)
    ) -> User:
        if current_user.role not in allowed_roles:
            raise HTTPException(
                status_code=403,
                detail=f"Доступ дозволено лише для ролей: "
                    f"{', '.join(allowed_roles)}",
            )
        return current_user
    return role_checker

```

Завдяки такому підходу будь-який ендпоінт можна захистити одним рядком — додавши `Depends(get_current_user)` як параметр функції-обробника, а для

ендпоінтів з обмеженням ролей — `Depends(require_role("admin"))`. Це робить код декларативним, читабельним і централізує логіку безпеки.

Стандартизація формату відповідей є важливим аспектом якісного API. У системі дотримано такі правила: успішні відповіді мають коди HTTP 200 (OK) для GET/PUT, 201 (Created) для POST, 204 (No Content) для DELETE; помилки клієнта мають коди 400 (Bad Request) при некоректних даних, 401 (Unauthorized) при відсутності або недійсності токена, 403 (Forbidden) при недостатніх правах, 404 (Not Found) для відсутніх ресурсів, 409 (Conflict) для конфліктів цілісності (дублювання email або коду дисципліни); помилки сервера мають код 500 (Internal Server Error). Усі помилки повертаються у єдиному форматі JSON з полем `detail`, що містить читабельний опис помилки українською мовою.

Валідація вхідних даних виконується автоматично через Pydantic-схеми. Кожен ендпоінт оголошує очікуваний тип даних, FastAPI самостійно перетворює JSON у Python-об'єкт відповідного класу і перевіряє його коректність. У разі невідповідності повертається HTTP 422 (Unprocessable Entity) з детальним описом усіх знайдених помилок — конкретне поле, тип помилки та значення, що викликало помилку. Це значно спрощує налагодження для клієнтських розробників.

Окрім ендпоінтів API, FastAPI автоматично надає інтерактивну документацію, доступну за адресами `/docs` (Swagger UI) і `/redoc` (ReDoc). Документація генерується на основі специфікації OpenAPI 3.0, яку фреймворк створює із Python-коду — анотацій типів, Pydantic-схем, docstrings функцій. Це означає, що документація завжди актуальна і не вимагає окремої підтримки. Користувач може безпосередньо в документації перевірити роботу будь-якого ендпоінта, ввівши тестові дані. Ця особливість FastAPI значно прискорює розробку та полегшує інтеграцію.

Підтримка CORS (Cross-Origin Resource Sharing) необхідна, оскільки клієнтський додаток і серверний API розгортаються на різних портах (5173 і 4000

під час розробки) або потенційно на різних доменах у виробничому середовищі. CORS-middleware налаштовано таким чином, щоб дозволяти запити з визначеного списку джерел (CORS\_ORIGINS у конфігурації) та підтримувати всі необхідні HTTP-методи і заголовки.

Висновки до розділу 2. У другому розділі здійснено системно-аналітичне проєктування інформаційної системи. Сформульовано функціональні та нефункціональні вимоги до системи, на основі яких визначено базовий MVP-функціонал, що охоплює управління користувачами з трьома ролями, ведення каталогу дисциплін та управління розкладом занять. Описано трирівневу архітектуру системи з виділенням рівнів представлення, бізнес-логіки та доступу до даних, що відповідає сучасним стандартам побудови веб-систем. Спроектовано модель даних на основі реляційного підходу, виділено три ключові сутності з відповідними атрибутами та зв'язками, побудовано ER-діаграму та DDL-схему бази даних з урахуванням принципів нормалізації, цілісності та продуктивності. Розроблено специфікацію REST API з 16 ендпоінтами, що покривають усі функціональні вимоги системи. Деталізовано механізм автентифікації на основі JWT-токенів та bcrypt-хешування паролів, реалізовано рольову модель доступу через систему залежностей FastAPI. Проектні рішення, отримані в межах цього розділу, є основою для програмної реалізації, що описана в наступному розділі.

## РОЗДІЛ 3. ПРОГРАМНА РЕАЛІЗАЦІЯ ВЕБОРІЄНТОВАНОЇ ІНФОРМАЦІЙНОЇ СИСТЕМИ

### 3.1. Реалізація серверної частини на основі FastAPI

Розпочнемо опис програмної реалізації з серверної частини, оскільки саме вона визначає бізнес-логіку системи, забезпечує цілісність даних і слугує основою, від якої залежить функціонування клієнтської частини. Серверна частина системи реалізована мовою програмування Python версії 3.12 з використанням асинхронного фреймворку FastAPI 0.115. Загальний обсяг серверного коду становить близько 700 рядків, розподілених між десятьма Python-модулями.

Структура серверного додатку відображає принцип розділення відповідальності, описаний у попередньому розділі. Папка `app` містить усі модулі застосунку, причому пов'язана логіка згрупована в підпапки. Точкою входу є файл `app/main.py`, у якому створюється екземпляр FastAPI, налаштовується CORS-middleware, реєструються роутери та оголошується `lifespan`-обробник для виконання ініціалізації бази даних при старті:

```
python
from contextlib import asynccontextmanager
from fastapi import FastAPI
from fastapi.middleware.cors import CORSMiddleware

from app.config import settings
from app.init_db import init_db
from app.routers import auth, schedule, subjects, users

@asynccontextmanager
async def lifespan(_app: FastAPI):
```

```
init_db()
yield
```

```
app = FastAPI(
    title=settings.APP_NAME,
    version=settings.APP_VERSION,
    lifespan=lifespan,
)
```

```
app.add_middleware(
    CORSMiddleware,
    allow_origins=settings.CORS_ORIGINS,
    allow_credentials=True,
    allow_methods=["*"],
    allow_headers=["*"],
)
```

```
app.include_router(auth.router)
app.include_router(subjects.router)
app.include_router(schedule.router)
app.include_router(users.router)
```

Конфігурація додатку винесена в окремий модуль `config.py`, що використовує бібліотеку `pydantic-settings`. Ця бібліотека дозволяє автоматично зчитувати параметри з файлу `.env` або змінних оточення, що відповідає принципу `The Twelve-Factor App` — методології розробки веб-застосунків, яка рекомендує не зберігати конфігурацію в коді. Модуль містить такі параметри: назву та версію додатку, URL бази даних, секрет JWT, час життя токена, список дозволених

CORS-доменів. Завдяки використанню Pydantic, типи параметрів перевіряються автоматично при запуску — некоректне значення викличе помилку ще до того, як додаток почне приймати запити.

Робота з базою даних організована через ORM SQLAlchemy 2.0 у поєднанні з механізмом залежностей FastAPI. Модуль `database.py` створює рушій (engine) для з'єднання з БД та фабрику сесій `SessionLocal`. Функція `get_db` є генератором, що створює сесію перед обробкою запиту і гарантовано закриває її після відповіді — навіть якщо виникла помилка:

```
python
def get_db():
    db = SessionLocal()
    try:
        yield db
    finally:
        db.close()
```

Така реалізація забезпечує коректне управління ресурсами і запобігає витокам з'єднань. Кожен ендпоінт, що працює з БД, оголошує параметр `db: Session = Depends(get_db)` — FastAPI автоматично викликає цю функцію перед обробкою запиту, передає сесію в обробник, і закриває її після завершення.

Моделі даних описані в модулі `models.py` у вигляді Python-класів, успадкованих від базового класу `Base`. Використано декларативний синтаксис SQLAlchemy 2.0 з анотаціями типів через `Mapped[T]`, що забезпечує статичну типізацію та підказки в IDE. Повний перелік моделей — `User`, `Subject`, `ScheduleItem` — наведено у Додатку А. Особливість моделей полягає у використанні зв'язків (relationships) — наприклад, у класі `Subject` визначено атрибут `teacher: Mapped[User | None]`, що дозволяє звертатися до пов'язаного об'єкта `User` напряму через крапку, без написання SQL JOIN-запитів вручну.

Pydantic-схеми, описані в модулі `schemas.py`, виконують подвійну функцію — валідацію вхідних даних та форматування вихідних. Для кожної сутності системи визначено кілька схем: базова схема з спільними полями (наприклад, `SubjectBase`), схема створення (`SubjectCreate`), схема оновлення (`SubjectUpdate`, з усіма полями опціональними), схема для відповіді (`SubjectOut`). Розділення схем дозволяє точно описати, які поля очікуються при різних операціях. Наприклад, при створенні дисципліни поле `teacher_id` опціональне, при оновленні — також опціональне, але разом із усіма іншими полями (PATCH-семантика), а у відповіді поле обов'язково присутнє разом із приєднаним `teacher_name`. Фрагмент схем для дисципліни:

```
python
class SubjectBase(BaseModel):
    name: str = Field(..., min_length=2, max_length=255)
    code: str = Field(..., min_length=2, max_length=20)
    description: str | None = None
    credits: int = Field(default=3, ge=1, le=10)

class SubjectCreate(SubjectBase):
    teacher_id: int | None = None

class SubjectOut(SubjectBase):
    model_config = ConfigDict(from_attributes=True)
    id: int
    teacher_id: int | None
    teacher_name: str | None = None
    created_at: datetime
```

Параметр `from_attributes=True` у конфігурації моделі вказує Pydantic читати дані з атрибутів об'єкта (як у SQLAlchemy ORM), а не з ключів словника, що дозволяє безпосередньо передавати ORM-об'єкти у відповідь.

Модуль `security.py` реалізує криптографічні функції та залежності для перевірки автентифікації. Функція `hash_password()` використовує `bcrypt` для генерації хешу пароля. Особливість реалізації — обмеження довжини пароля 72 байтами, що є технічним обмеженням алгоритму `bcrypt`:

```
python
def hash_password(password: str) -> str:
    pwd_bytes = password.encode("utf-8")[:72]
    return bcrypt.hashpw(pwd_bytes, bcrypt.gensalt()).decode("utf-8")

def verify_password(plain: str, hashed: str) -> bool:
    pwd_bytes = plain.encode("utf-8")[:72]
    try:
        return bcrypt.checkpw(pwd_bytes, hashed.encode("utf-8"))
    except (ValueError, TypeError):
        return False
```

Функція `create_access_token()` генерує JWT-токен з `payload`, що містить ідентифікатор користувача, його роль та час закінчення дії. Час життя токена за замовчуванням — 24 години, що є компромісом між зручністю користувача та безпекою. Алгоритм підпису — HS256, що використовує симетричний секретний ключ і є достатнім для систем з єдиним сервером:

```
python
def create_access_token(user_id: int, role: str) -> str:
    expire = datetime.now(timezone.utc) + timedelta(
        minutes=settings.JWT_EXPIRE_MINUTES)
```

```

payload = {"sub": str(user_id), "role": role, "exp": expire}
return jwt.encode(payload, settings.JWT_SECRET,
                  algorithm=settings.JWT_ALGORITHM)

```

API-роутери розташовані в папці `app/routers/`. Кожен роутер відповідає за одну логічну групу ендпоінтів. Розглянемо реалізацію типового CRUD-роутера на прикладі `subjects.py`. Ендпоінт створення дисципліни демонструє, як декілька механізмів FastAPI працюють разом:

```

python
@router.post("", response_model=SubjectOut,
             status_code=status.HTTP_201_CREATED)
def create_subject(
    data: SubjectCreate,
    db: Session = Depends(get_db),
    current_user: User = Depends(require_role("teacher", "admin")),
):
    if db.query(Subject).filter(Subject.code == data.code).first():
        raise HTTPException(
            status_code=status.HTTP_409_CONFLICT,
            detail=f"Предмет з кодом '{data.code}' вже існує",
        )

    teacher_id = data.teacher_id
    if current_user.role == "teacher":
        teacher_id = current_user.id

    subject = Subject(
        name=data.name, code=data.code,
        description=data.description,

```

```

        teacher_id=teacher_id, credits=data.credits,
    )
    db.add(subject)
    db.commit()
    db.refresh(subject)
    return _to_out(subject)

```

Цей ендпоінт демонструє кілька важливих практик. По-перше, валідація вхідних даних повністю автоматизована через схему SubjectCreate — недійсні дані відсікаються до виклику обробника. По-друге, перевірка автентифікації та ролі здійснюється через залежність `require_role`, що повертає об'єкт користувача — обробник не містить явної перевірки, лише декларує необхідну роль. По-третє, бізнес-логіка враховує контекст — викладач може створювати лише власні дисципліни, тому `teacher_id` перезаписується ідентифікатором поточного користувача. По-четверте, обробка конфлікту унікальності виконується явно перед створенням запису з дружнім повідомленням про помилку.

Аналогічно реалізовано всі CRUD-операції для дисциплін та розкладу. Окрему увагу слід приділити роутеру `auth.py`, що містить ендпоінти реєстрації, входу та отримання даних поточного користувача. У ендпоінті `register` виконується кілька важливих перевірок: заборонено реєстрацію з роллю `admin` (адміністраторів створює лише існуючий адміністратор), перевіряється унікальність `email` перед створенням запису. У відповідь повертається не лише дані користувача, але й JWT-токен, що дозволяє клієнту автоматично авторизуватися після реєстрації.

Ендпоінт `login` виконує перевірку паролю через `verify_password` і генерує токен у разі успіху. Особливість реалізації — однакове повідомлення про помилку для випадків неіснуючого користувача та неправильного пароля. Це є рекомендованою практикою безпеки, що ускладнює зловмисникам перерахунок існуючих облікових записів.

Реалізація фільтрації в ендпоінті GET /api/schedule демонструє роботу з опціональними query-параметрами:

```
python
@router.get("", response_model=list[ScheduleOut])
def list_schedule(
    day: int | None = Query(default=None, ge=1, le=7),
    group: str | None = Query(default=None),
    db: Session = Depends(get_db),
    _: User = Depends(get_current_user),
):
    query = (
        db.query(ScheduleItem)
        .options(joinedload(ScheduleItem.subject)
                 .joinedload(Subject.teacher))
    )
    if day is not None:
        query = query.filter(ScheduleItem.day_of_week == day)
    if group:
        query = query.filter(ScheduleItem.group_name == group)

    items = query.order_by(ScheduleItem.day_of_week,
                           ScheduleItem.start_time).all()
    return [_to_out(i) for i in items]
```

Конструкція `joinedload(ScheduleItem.subject).joinedload(Subject.teacher)` інструктує SQLAlchemy виконати JOIN-запит до пов'язаних таблиць у єдиному SQL-запиті. Без цієї оптимізації для кожного запису розкладу виконувався б окремий запит на отримання дисципліни, а потім ще один — на отримання

викладача (так звана N+1 проблема). У результаті оптимізації запит до 100 записів розкладу виконується за один SQL-запит замість 201.

Модуль `init_db.py` відповідає за ініціалізацію бази даних при першому запуску. При старті додатку викликається функція `init_db()`, яка створює всі таблиці згідно з ORM-моделями (метод `Base.metadata.create_all()`), а потім перевіряє наявність даних. Якщо в системі немає адміністратора — створюється обліковий запис за замовчуванням ([admin@edu.local](mailto:admin@edu.local) / admin123). Якщо немає дисциплін — завантажуються демонстраційні дані: два викладачі, один студент, три дисципліни, п'ять записів розкладу. Ця стратегія забезпечує можливість одразу почати роботу з системою.

Запуск сервера виконується через ASGI-сервер Uvicorn командою `uvicorn app.main:app --reload --port 4000`. Прапорець `--reload` вмикає режим автоматичного перезапуску при зміні файлів, що значно прискорює розробку. У виробничому середовищі рекомендується використовувати кілька воркерів через Gunicorn з `UvicornWorker` для повноцінного використання багатоядерних процесорів.

Тестування серверної частини проводилося через автоматично згенеровану документацію Swagger UI, доступну за адресою <http://localhost:4000/docs> після запуску сервера. Інтерактивна документація дозволяє безпосередньо викликати будь-який ендпоінт, передаючи тестові дані, та одразу бачити відповідь. Це значно прискорює налагодження і дозволяє перевірити коректність API без написання окремих тестових клієнтів.

### 3.2. Розробка клієнтського інтерфейсу засобами React

Клієнтська частина системи являє собою односторінковий веб-додаток (SPA) на основі бібліотеки React 18, що взаємодіє з серверним API через HTTP-запити. Проєкт організовано як стандартний Vite-додаток зі структурою папок: `src/main.jsx` — точка входу, `src/App.jsx` — конфігурація маршрутизатора, `src/api.js` — обгортка над `axios`, `src/context/` — React-контексти, `src/components/` —

повторно використовувані компоненти, `src/pages/` — сторінки додатку. Загальний обсяг клієнтського коду — близько 1700 рядків.

Початкова конфігурація Vite в файлі `vite.config.js` містить налаштування dev-сервера, зокрема проксі для перенаправлення запитів до `/api` на серверну частину:

```
javascript
export default defineConfig({
  plugins: [react()],
  server: {
    port: 5173,
    proxy: {
      '/api': {
        target: 'http://localhost:4000',
        changeOrigin: true,
      },
    },
  },
})
```

Завдяки цьому під час розробки немає потреби займатися CORS-проблемами або вказувати повний URL сервера в коді — клієнт надсилає запити на власний домен, а Vite прозоро перенаправляє їх до бекенду.

Стилізація реалізована через TailwindCSS з кастомною кольоровою палітрою у файлі `tailwind.config.js`. Палітра відображає академічну естетику: основний колір `ink` (темно-зелений з чорнильним відтінком), фоновий `cream` (теплий кремовий), акцентний `saffron` (шафрановий жовто-помаранчевий). Шрифтова пара — `Fraunces` для заголовків (характерний `serif` з варіативною віссю) та `Manrope` для основного тексту (геометричний `sans-serif`). Така комбінація створює відчуття академічного видання, не вдаючись до стандартних

шаблонних шрифтів типу Arial або Roboto, що часто зустрічаються в навчальних проєктах.

Управління автентифікацією централізоване в React-контексті AuthContext (src/context/AuthContext.jsx). Контекст експортує провайдер AuthProvider та хук useAuth для доступу до стану. У стані зберігається об'єкт поточного користувача та прапорець завантаження. При першому рендерингу провайдер перевіряє sessionStorage на наявність збереженого токена і, якщо він є, виконує запит /api/auth/me для отримання актуальних даних користувача:

```
javascript
useEffect(() => {
  const saved = sessionStorage.getItem('edu_token')
  if (saved) {
    setToken(saved)
    api.get('/auth/me')
      .then((r) => setUser(r.data))
      .catch(() => {
        setToken(null)
        sessionStorage.removeItem('edu_token')
      })
    .finally(() => setLoading(false))
  } else {
    setLoading(false)
  }
}, [])
```

Цей механізм забезпечує збереження сесії при перезавантаженні сторінки — користувач не зобов'язаний кожного разу заново логінитися. Водночас sessionStorage обмежує термін зберігання токена закриттям вкладки браузера, що є компромісом між зручністю та безпекою.

Контекст експортує методи `login(email, password)`, `register(payload)` і `logout()`, що інкапсулюють відповідну логіку. Метод `login` виконує POST-запит до `/api/auth/login`, зберігає отриманий токен у `sessionStorage` та налаштовує заголовок авторизації для `axios`. Метод `logout` очищує всі дані. Усі ці операції автоматично викликають перерендеринг компонентів, що використовують контекст, оскільки React відстежує зміни стану.

Маршрутизація реалізована через бібліотеку `react-router-dom` версії 6. У файлі `App.jsx` визначено маршрути додатку:

```
javascript
```

```
<Routes>
  <Route path="/login" element={
    user ? <Navigate to="/" replace /> : <Login />
  } />
  <Route path="/register" element={
    user ? <Navigate to="/" replace /> : <Register />
  } />
  <Route path="/" element={
    <ProtectedRoute><Dashboard /></ProtectedRoute>
  } />
  <Route path="/subjects" element={
    <ProtectedRoute><Subjects /></ProtectedRoute>
  } />
  <Route path="/schedule" element={
    <ProtectedRoute><Schedule /></ProtectedRoute>
  } />
  <Route path="/users" element={
    <ProtectedRoute roles={['admin']}><Users /></ProtectedRoute>
  } />
```

```
<Route path="*" element={<Navigate to="/" replace />} />
</Routes>
```

Компонент `ProtectedRoute` виконує перевірку автентифікації перед рендерингом дочірнього компонента. Якщо користувач не автентифікований — здійснюється перенаправлення на сторінку логіна із збереженням початкового URL у параметрі `state`, що дозволяє після успішного входу повернутися на потрібну сторінку. Опціональний параметр `roles` дозволяє обмежити доступ до маршруту лише користувачам з певними ролями — у разі невідповідності виводиться сторінка з кодом 403 та поясненням.

Робота з API винесена в окремий модуль `src/api.js`, що створює єдиний екземпляр `axios` із базовим URL `/api`. Через перехоплювач відповідей реалізовано глобальну обробку помилки 401 — якщо токен став недійсним, локальний стан очищується і викликається подія `auth:logout`, на яку реагує `AuthContext`:

```
javascript
const api = axios.create({ baseURL: '/api' })

api.interceptors.response.use(
  (r) => r,
  (err) => {
    if (err.response?.status === 401 && _token) {
      setToken(null)
      window.dispatchEvent(new CustomEvent('auth:logout'))
    }
    return Promise.reject(err)
  },
)
```

Розглянемо реалізацію основних сторінок. Сторінка `Dashboard` (`src/pages/Dashboard.jsx`) — це інформаційна панель після входу в систему. При

завантаженні компонента паралельно виконуються три запити: список усіх дисциплін, повний розклад та розклад на поточний день тижня. Поточний день обчислюється як  $((\text{new Date().getDay()} + 6) \% 7) + 1$ , що перетворює стандартну нумерацію JavaScript (де неділя — 0) у нумерацію 1-7, де понеділок — 1.

Дашборд містить персоналізоване привітання з ім'ям користувача, три картки статистики (кількість дисциплін, записів розкладу, занять сьогодні), список занять на сьогоднішній день та панель швидких дій, що адаптується залежно від ролі. Для адміністратора доступні швидкі дії "Управління користувачами" та "Створити предмет", для викладача — лише останній, для студента — лише навігаційні дії перегляду.

Сторінка Subjects реалізує повний CRUD-інтерфейс для дисциплін. Відображення — у вигляді сітки карток (CSS Grid із автоматичним підбором кількості колонок через `minmax`). Кожна картка містить код дисципліни у вигляді "значка" моноширинного шрифту, кількість кредитів, назву, опис, інформацію про викладача. Кнопки "Редагувати" і "Видалити" відображаються лише для користувачів з відповідними правами — викладач бачить кнопку редагування лише для власних дисциплін, кнопка видалення доступна лише адміністратору.

Створення та редагування дисциплін виконується в модальному вікні. Модальне вікно реалізовано як окремий компонент `SubjectFormModal`, що приймає опціональний об'єкт `subject` — якщо переданий, форма працює в режимі редагування, інакше — створення. Форма містить поля назви, коду, опису, кредитів. При надсиланні відбувається відповідний API-запит, у разі успіху модальне вікно закривається і список перезавантажується. Обробка помилок реалізована через відображення червоного повідомлення в нижній частині форми, без модальних `alert`-діалогів, що порушували б користувацький досвід.

Сторінка Schedule (`src/pages/Schedule.jsx`) є найскладнішою з точки зору верстки — вона відображає тижневий розклад у вигляді шести колонок (по одній на робочий день тижня). Кожна колонка містить картки занять, відсортовані за

часом. Колонка поточного дня візуально виділена — у заголовку відображається значок "Сьогодні" з акцентним кольором. Реалізація групування даних за днем тижня:

```
javascript
const byDay = {}
for (let d = 1; d <= 6; d++) byDay[d] = []
items.forEach((it) => {
  if (byDay[it.day_of_week]) byDay[it.day_of_week].push(it)
})
```

Адміністратор може створювати, редагувати та видаляти записи розкладу через модальне вікно `ScheduleFormModal`. Форма містить випадаючий список дисциплін (завантажується разом із розкладом), вибір дня тижня, два поля часу (HTML5 `input type="time"`), поля аудиторії та групи. Перед відправкою на сервер виконується клієнтська перевірка — час початку має бути меншим за час закінчення, але остаточна валідація все одно дублюється на серверній стороні.

Сторінка `Users` доступна лише адміністратору і відображає табличне представлення всіх користувачів системи. Таблиця містить ідентифікатор, повне ім'я, `email`, роль (із кольоровим оформленням значка залежно від ролі), дату створення. Адміністратор може створювати нових користувачів з будь-якою роллю, включно з адміністраторською, через модальне вікно `UserFormModal`. Видалення власного облікового запису заблоковано на рівні UI — кнопка не відображається для рядка поточного користувача, що дублюється відповідною перевіркою на серверній стороні.

Спільні елементи інтерфейсу винесено в компонент `Navbar` (`src/components/Navbar.jsx`). Шапка містить логотип `EduSystem`, навігаційні посилання (з умовним відображенням посилання "Користувачі" лише для адміністраторів), інформацію про поточного користувача та кнопку виходу.

Активний пункт навігації виділяється темним фоном — реалізовано через перевірку поточного маршруту через хук `useLocation`.

Анімації реалізовано через CSS `keyframes`, не вдаючись до важких бібліотек анімації типу `framer-motion`. Основна анімація — `fadeUp` (плавна поява з нижнього зсуву). Послідовна анімація списків реалізована через інлайн-стиль `animationDelay` з обчисленням затримки на основі індексу елемента, що створює ефект "каскадної" появи — спочатку з'являється перший елемент, потім другий і так далі з затримкою у 40 мілісекунд між кожним.

Production-збірка виконується командою `npm run build` і займає близько 3-4 секунд завдяки оптимізації `Rollup`, на якому базується `Vite`. Результат — оптимізований JavaScript-бандл розміром 241 KB у вихідному вигляді або 77 KB після стиснення `gzip`, та CSS-файл 21 KB (4.3 KB `gzipped`). Такий розмір є дуже прийнятним для сучасного веб-додатку і забезпечує швидке завантаження навіть на повільному з'єднанні.

### 3.3. Тестування системи та аналіз отриманих результатів

Після завершення реалізації системи проведено комплексне тестування для перевірки коректності її функціонування. Тестування включало кілька рівнів: інтеграційне тестування серверного API, ручне тестування користувацьких сценаріїв через інтерфейс, перевірку відповідності функціональним вимогам, аналіз продуктивності.

Інтеграційне тестування серверного API виконано за допомогою фреймворку `TestClient`, що входить до складу `FastAPI`. `TestClient` емулює HTTP-запити безпосередньо в коді, не вимагаючи запуску окремого сервера, що значно прискорює виконання тестів. Тестовий сценарій охоплює основні робочі шляхи системи:

```
python
from fastapi.testclient import TestClient
```

```
from app.main import app

client = TestClient(app)
with client:

    r = client.get('/')
    assert r.status_code == 200
    assert r.json()['status'] == 'ok'

    r = client.post('/api/auth/login',
                    json={'email': 'admin@edu.local',
                          'password': 'admin123'})
    assert r.status_code == 200
    token = r.json()['access_token']
    h = {'Authorization': f'Bearer {token}'}

    r = client.get('/api/subjects', headers=h)
    assert r.status_code == 200
    assert len(r.json()) == 3

    r = client.get('/api/schedule', headers=h)
    assert r.status_code == 200
    assert len(r.json()) == 5

    r = client.post('/api/auth/register',
                    json={'email': 'newstd@test.com',
                          'password': 'pass1234',
                          'full_name': 'Новий Студент',
                          'role': 'student'})
```

```

assert r.status_code == 201

student_token = r.json()['access_token']

r = client.delete('/api/subjects/1',
                  headers={'Authorization': f'Bearer {student_token}'})
assert r.status_code == 403

```

Тестовий сценарій перевіряє: ендпоінт перевірки статусу, авторизацію адміністратора, отримання списків дисциплін і розкладу з очікуваною кількістю записів, реєстрацію нового студента, перевірку коректної роботи рольової моделі (студент не може видалити дисципліну). Усі тести успішно пройдено, що підтверджує коректність базового функціоналу системи.

Ручне тестування користувацьких сценаріїв виконувалося через веб-інтерфейс. Перевірено такі сценарії: реєстрацію нового користувача з різними ролями, вхід у систему під різними обліковими записами, перегляд та фільтрацію розкладу, створення, редагування та видалення дисциплін під різними ролями, управління користувачами адміністратором, обробку помилкових ситуацій (введення некоректного пароля, дублювання email, спроба несанкціонованих дій). У всіх сценаріях система поводить відповідно до специфікації.

Узагальнено результати тестування представлено у таблиці.

Таблиця 3.1 Результати тестування основних функцій системи

| № | Функція    | Тестовий сценарій           | Очікуваний результат                            | Фактичний результат | Статус |
|---|------------|-----------------------------|-------------------------------------------------|---------------------|--------|
| 1 | Реєстрація | Створення нового студента   | HTTP 201, повернення токена і даних користувача | Збігається          | Успіх  |
| 2 | Реєстрація | Повторне використання email | HTTP 409, повідомлення про конфлікт             | Збігається          | Успіх  |

| №  | Функція     | Тестовий сценарій               | Очікуваний результат                                    | Фактичний результат | Статус |
|----|-------------|---------------------------------|---------------------------------------------------------|---------------------|--------|
| 3  | Реєстрація  | Спроба зареєструватися як admin | HTTP 403, заборонено                                    | Збігається          | Успіх  |
| 4  | Авторизація | Правильні дані                  | HTTP 200, повернення токена                             | Збігається          | Успіх  |
| 5  | Авторизація | Неправильний пароль             | HTTP 401, помилка автентифікації                        | Збігається          | Успіх  |
| 6  | Дисципліни  | Перегляд списку студентом       | HTTP 200, список з 3 елементів                          | Збігається          | Успіх  |
| 7  | Дисципліни  | Створення викладачем            | HTTP 201, дисципліна з teacher_id поточного користувача | Збігається          | Успіх  |
| 8  | Дисципліни  | Видалення студентом             | HTTP 403, недостатньо прав                              | Збігається          | Успіх  |
| 9  | Розклад     | Фільтр за днем тижня            | HTTP 200, лише записи поточного дня                     | Збігається          | Успіх  |
| 10 | Розклад     | Створення запису адміном        | HTTP 201, запис з пов'язаною дисципліною                | Збігається          | Успіх  |
| 11 | Користувачі | Перегляд списку студентом       | HTTP 403, доступ заборонено                             | Збігається          | Успіх  |
| 12 | Користувачі | Видалення власного запису       | HTTP 400, операція заборонена                           | Збігається          | Успіх  |

Джерело: складено автором за результатами тестування.

Аналіз продуктивності системи проведено шляхом вимірювання часу обробки запитів у середовищі розробки. Тестова конфігурація: Python 3.12, FastAPI 0.115, SQLite, ноутбук із чотириядерним процесором. Виміряні часи обробки типових запитів:

– GET /api/subjects (3 записи) — 4-8 мілісекунд; – GET /api/schedule (5 записів із JOIN) — 6-10 мілісекунд; – POST /api/auth/login (з bcrypt-перевіркою) — 80-120 мілісекунд (визначається швидкістю bcrypt); – POST /api/auth/register — 90-130 мілісекунд; – POST /api/subjects — 8-15 мілісекунд.

Усі показники задовольняють заявлені у вимогах до системи 200 мілісекунд для типових запитів. Найповільнішими є запити, пов'язані з bcrypt-хешуванням, що є очікуваним — алгоритм спеціально побудований так, щоб бути обчислювально витратним для протидії атакам повним перебором.

Розмір клієнтської частини у production-збірці становить 241 KB JavaScript та 21 KB CSS у нестисненому вигляді (77 KB і 4.3 KB після gzip відповідно). На з'єднанні зі швидкістю 10 Мбіт/с завантаження всіх ресурсів займає менше секунди. Початковий рендеринг сторінки після завантаження — близько 200-300 мілісекунд, що сприймається користувачем як миттєве.

Узагальнюючи результати, можна стверджувати, що розроблена система задовольняє всі функціональні і нефункціональні вимоги, сформульовані на етапі проєктування. Реалізовано три основні домени — управління користувачами з диференціацією за трьома ролями, ведення каталогу дисциплін, управління розкладом. Підтверджено коректну роботу механізму автентифікації на основі JWT-токенів та рольової авторизації. Підтверджено відповідність вимогам продуктивності — типові запити обробляються за десятки мілісекунд.

Перспективи подальшого розвитку системи бачаться в кількох напрямках. По-перше, розширення функціональних можливостей: додавання модуля оцінювання студентів, можливості завантаження навчальних матеріалів, інтеграція з відеоконференц-системами для проведення онлайн-занять. По-друге, поглиблення механізмів безпеки: додавання двофакторної автентифікації, аудит дій користувачів, шифрування чутливих даних у базі. По-третє, перенесення на більш потужну СУБД (PostgreSQL) для роботи з великими обсягами даних та підтримки декількох одночасних з'єднань. По-четверте, розширення мобільного

інтерфейсу, можливо — створення нативного мобільного додатку через React Native, що дозволить повторно використати значну частину компонентного коду. По-п'яте, інтеграція з ЄДЕБО для автоматичного імпорту даних про здобувачів освіти, що значно знизить навантаження на адміністрацію закладу.

Висновки до розділу 3. У третьому розділі описано програмну реалізацію інформаційної системи на основі архітектурних рішень, прийнятих у попередніх розділах. Серверну частину побудовано на фреймворку FastAPI, що дозволив реалізувати 16 ендпоінтів REST API з мінімальним обсягом коду завдяки автоматичній валідації даних через Pydantic, декларативному визначенню залежностей та автоматичній генерації документації OpenAPI. Робота з базою даних SQLite виконується через ORM SQLAlchemy 2.0 з використанням сучасного декларативного синтаксису. Безпека реалізована через JWT-токени та bcrypt-хешування паролів з рольовою моделлю доступу. Клієнтську частину побудовано як SPA на бібліотеці React 18 із застосуванням маршрутизатора React Router 6, HTTP-клієнта Axios та CSS-фреймворку TailwindCSS. Розроблено шість сторінок (Login, Register, Dashboard, Subjects, Schedule, Users), що покривають усі функціональні вимоги. Управління автентифікацією централізоване в React-контексті AuthContext із збереженням токена у sessionStorage. Тестування системи методами автоматизованих інтеграційних тестів та ручної перевірки користувацьких сценаріїв підтвердило коректність роботи всіх функцій. Аналіз продуктивності показав, що часи обробки запитів значно нижчі за встановлені у вимогах ліміти. Сформульовано перспективи подальшого розвитку системи в напрямках розширення функціоналу, посилення безпеки, масштабування та інтеграції із зовнішніми системами.

## ВИСНОВКИ

У результаті виконання дипломної роботи було розроблено інформаційну систему управління освітнім процесом, яка спрямована на автоматизацію основних процесів взаємодії між учасниками навчального середовища — студентами, викладачами та адміністраторами. Актуальність роботи обумовлена зростаючою потребою у цифровізації освітніх процесів та впровадженні сучасних інформаційних технологій у сферу освіти.

У ході дослідження було проаналізовано сучасні підходи до розробки веб-додатків, зокрема клієнт-серверну модель, REST-архітектуру та принципи побудови багаторівневих систем. На основі проведеного аналізу було обрано трирівневу архітектуру, яка включає рівень представлення, рівень застосунку та рівень доступу до даних. Такий підхід дозволяє чітко розмежувати відповідальність між компонентами системи, підвищити її гнучкість, масштабованість та зручність подальшого супроводу.

Рівень представлення реалізовано за допомогою сучасної бібліотеки React, що забезпечує створення інтерактивного та зручного користувацького інтерфейсу. Рівень застосунку побудовано з використанням фреймворку FastAPI, який дозволяє ефективно обробляти запити, реалізовувати бізнес-логіку та забезпечувати взаємодію з клієнтською частиною через REST API. Для зберігання даних використано реляційну базу даних SQLite, а доступ до неї організовано за допомогою ORM SQLAlchemy, що спрощує роботу з базою даних та підвищує переносимість програмного коду.

У процесі проектування було розроблено структуру бази даних, яка включає ключові сутності системи: користувачі, дисципліни, розклад занять та записи дій. Побудована ER-діаграма відображає логічні зв'язки між цими сутностями, зокрема відношення типу «один-до-багатьох», що дозволяє забезпечити цілісність і узгодженість даних. Крім того, було створено DDL-опис таблиць, який визначає структуру бази даних та її основні обмеження.

У рамках практичної реалізації системи було розроблено основний функціонал, що включає механізм реєстрації та автентифікації користувачів із використанням JWT-токенів, управління дисциплінами, формування та редагування розкладу занять, а також адміністрування облікових записів. Реалізація CRUD-операцій забезпечує повний цикл роботи з даними та дозволяє ефективно управляти інформацією в системі.

Особливу увагу було приділено організації взаємодії між клієнтською та серверною частинами, яка здійснюється через HTTP-протокол з використанням формату JSON. Це забезпечує універсальність системи та можливість її інтеграції з іншими сервісами у майбутньому. Використання сучасних технологій дозволило створити систему, яка відповідає основним вимогам до веб-додатків: швидкодії, зручності використання та надійності.

Розроблена система демонструє стабільну роботу та може бути використана як основа для створення повноцінної освітньої платформи. У подальшому можливе її вдосконалення шляхом розширення функціональних можливостей, зокрема додавання системи повідомлень, інтеграції з електронними журналами, реалізації ролей із більш гнучкою системою прав доступу, а також підвищення рівня інформаційної безпеки.

Таким чином, у ході виконання дипломної роботи було досягнуто поставленої мети, вирішено всі визначені завдання та отримано практичний результат у вигляді працездатної інформаційної системи, що може бути використана у навчальному процесі або як основа для подальших досліджень і розробок у сфері інформаційних технологій.

### СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Fielding R. Architectural Styles and the Design of Network-based Software Architectures : Doctoral dissertation. – University of California, 2000.
2. Sommerville I. Software Engineering. – 10th ed. – Boston : Pearson, 2016. – 816 p.
3. Pressman R. Software Engineering: A Practitioner's Approach. – 8th ed. – New York : McGraw-Hill, 2014. – 976 p.
4. Gamma E. Design Patterns: Elements of Reusable Object-Oriented Software. – Boston : Addison-Wesley, 1994. – 395 p.
5. Fowler M. Patterns of Enterprise Application Architecture. – Boston : Addison-Wesley, 2003. – 533 p.
6. Evans E. Domain-Driven Design. – Boston : Addison-Wesley, 2004. – 560 p.
7. Grinberg M. Flask Web Development. – 2nd ed. – Sebastopol : O'Reilly, 2018. – 258 p.
8. Ramírez S. FastAPI Official Documentation [Електронний ресурс]. – Режим доступу: <https://fastapi.tiangolo.com/>
9. Pydantic Documentation [Електронний ресурс]. – Режим доступу: <https://docs.pydantic.dev/>
10. SQLAlchemy Documentation [Електронний ресурс]. – Режим доступу: <https://docs.sqlalchemy.org/>
11. SQLite Documentation [Електронний ресурс]. – Режим доступу: <https://www.sqlite.org/docs.html>
12. Silberschatz A. Database System Concepts. – 7th ed. – New York : McGraw-Hill, 2019. – 1376 p.
13. Elmasri R., Navathe S. Fundamentals of Database Systems. – 7th ed. – Boston : Pearson, 2016. – 1272 p.
14. Date C. An Introduction to Database Systems. – 8th ed. – Pearson, 2004. – 1024 p.

15. Mozilla Developer Network. HTTP Overview [Электронный ресурс]. – Режим доступа: <https://developer.mozilla.org/en-US/docs/Web/HTTP>
16. RFC 7231: Hypertext Transfer Protocol (HTTP/1.1) [Электронный ресурс]. – Режим доступа: <https://www.rfc-editor.org/rfc/rfc7231>
17. JSON Specification [Электронный ресурс]. – Режим доступа: <https://www.json.org/json-en.html>
18. React Documentation [Электронный ресурс]. – Режим доступа: <https://react.dev/>
19. Abramov D. The Road to React. – Self-published, 2020.
20. Banks A., Porcello E. Learning React. – 2nd ed. – O'Reilly, 2020. – 310 p.
21. JWT Introduction [Электронный ресурс]. – Режим доступа: <https://jwt.io/introduction>
22. OWASP Top 10 Web Security Risks [Электронный ресурс]. – Режим доступа: <https://owasp.org/www-project-top-ten/>
23. Stallings W. Cryptography and Network Security. – 7th ed. – Pearson, 2017.
24. Kurose J., Ross K. Computer Networking: A Top-Down Approach. – 7th ed. – Pearson, 2017.
25. Tanenbaum A. Computer Networks. – 5th ed. – Pearson, 2010.
26. Gamma E. Head First Design Patterns. – O'Reilly, 2020.
27. Larman C. Applying UML and Patterns. – 3rd ed. – Prentice Hall, 2004.
28. Booch G. UML User Guide. – Addison-Wesley, 2005.
29. Sommerville I. Engineering Software Products. – Pearson, 2019.
30. Martin R. Clean Architecture. – Prentice Hall, 2017.
31. Martin R. Clean Code. – Prentice Hall, 2008.
32. Oracle. SQL Language Reference [Электронный ресурс]. – Режим доступа: <https://docs.oracle.com>
33. PostgreSQL Documentation [Электронный ресурс]. – Режим доступа: <https://www.postgresql.org/docs/>

- 34.MDN Web Docs. JavaScript Guide [Электронный ресурс]. – Режим доступа: <https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide>
- 35.ECMA-262 JavaScript Specification [Электронный ресурс]. – Режим доступа: <https://tc39.es/ecma262/>
- 36.W3C. Web Architecture [Электронный ресурс]. – Режим доступа: <https://www.w3.org/>
- 37.ISO/IEC 25010:2011 Systems and software quality models.
- 38.Microsoft Docs. REST API design [Электронный ресурс]. – Режим доступа: <https://learn.microsoft.com/>
- 39.IBM Cloud Docs. Microservices architecture [Электронный ресурс]. – Режим доступа: <https://cloud.ibm.com/docs>
- 40.OpenAPI Specification [Электронный ресурс]. – Режим доступа: <https://swagger.io/specification/>
- 41.Swagger Documentation [Электронный ресурс]. – Режим доступа: <https://swagger.io/docs/>
- 42.GitHub Docs [Электронный ресурс]. – Режим доступа: <https://docs.github.com/>

## ДОДАТКИ

### Додаток А. Лістинг ключових фрагментів коду серверної частини

У даному додатку наведено основні фрагменти програмного коду серверної частини інформаційної системи, реалізованої з використанням фреймворку FastAPI. Представлені приклади демонструють реалізацію основних функцій системи, зокрема роботу з базою даних, автентифікацію користувачів та обробку HTTP-запитів.

#### Фрагмент А.1 — Опис моделі користувача (ORM SQLAlchemy):

```
class User(Base):
    __tablename__ = "users"

    id = Column(Integer, primary_key=True, index=True)
    email = Column(String, unique=True, index=True, nullable=False)
    password_hash = Column(String, nullable=False)
    full_name = Column(String)
    role = Column(String)
    created_at = Column(DateTime, default=datetime.utcnow)
```

#### Фрагмент А.2 — Реалізація авторизації користувача (FastAPI):

```
@app.post("/login")
def login(user: UserLogin, db: Session = Depends(get_db)):
    db_user = db.query(User).filter(User.email == user.email).first()
    if not db_user or not verify_password(user.password, db_user.password_hash):
        raise HTTPException(status_code=401, detail="Невірні облікові дані")

    access_token = create_access_token(data={"sub": db_user.email})
    return {"access_token": access_token, "token_type": "bearer"}
```

**Фрагмент A.3 — Отримання списку предметів (API-запит):**

```
@app.get("/subjects")
def get_subjects(db: Session = Depends(get_db)):
    return db.query(Subject).all()
```

**Фрагмент A.4 — Додавання нового предмету:**

```
@app.post("/subjects")
def create_subject(subject: SubjectCreate, db: Session = Depends(get_db)):
    db_subject = Subject(**subject.dict())
    db.add(db_subject)
    db.commit()
    db.refresh(db_subject)
    return db_subject
```

**Фрагмент A.5 — Додавання запису до розкладу:**

```
@app.post("/schedule")
def create_schedule(item: ScheduleCreate, db: Session = Depends(get_db)):
    if item.start_time >= item.end_time:
        raise HTTPException(status_code=400, detail="Невірний час")

    schedule = Schedule(**item.dict())
    db.add(schedule)
    db.commit()
    db.refresh(schedule)
    return schedule
```

**Фрагмент A.6 — Підключення до бази даних:**

```
engine = create_engine("sqlite:///./database.db")
```

```
SessionLocal = sessionmaker(autocommit=False, autoflush=False, bind=engine)
```

### **Додаток Б. Лістинг ключових фрагментів коду клієнтської частини**

У даному додатку наведено основні фрагменти програмного коду клієнтської частини інформаційної системи, реалізованої з використанням JavaScript та бібліотеки React (демо-версія). Представлені приклади демонструють організацію інтерфейсу користувача, взаємодію з API та обробку подій.

#### **Фрагмент Б.1 — Ініціалізація стану додатку:**

```
const state = {
  currentUser: null,
  currentPage: 'login',
  modal: null,
};
```

#### **Фрагмент Б.2 — Функція авторизації користувача:**

```
function login(email, password) {
  const user = DB.users.find(u => u.email === email);
  if (!user || user.password !== password) {
    throw new Error("Невірна електронна пошта або пароль");
  }
  return user;
}
```

#### **Фрагмент Б.3 — Відображення сторінки входу:**

```
function renderLogin() {
  const form = { email: "", password: "" };
}
```

```

return h('form', {
  onsubmit: (e) => {
    e.preventDefault();
    try {
      state.currentUser = login(form.email, form.password);
      state.currentPage = 'dashboard';
      render();
    } catch (err) {
      alert(err.message);
    }
  }
},
  h('input', { type: 'email', placeholder: 'Email',
    oninput: e => form.email = e.target.value }),
  h('input', { type: 'password', placeholder: 'Пароль',
    oninput: e => form.password = e.target.value }),
  h('button', { type: 'submit' }, 'Увійти')
);
}

```

#### **Фрагмент Б.4 — Отримання списку предметів:**

```

function getSubjects() {
  return DB.subjects.map(s => ({
    ...s,
    teacher_name: s.teacher_id
      ? DB.users.find(u => u.id === s.teacher_id)?.full_name
      : null
  }

```

```

    ));
  }

```

### **Фрагмент Б.5 — Відображення списку предметів:**

```

function renderSubjects(subjects) {
  return h('div', {},
    ...subjects.map(s =>
      h('div', { class: 'subject-card' },
        h('h3', {}, s.name),
        h('p', {}, s.description),
        h('span', {}, s.teacher_name || '—')
      )
    )
  );
}

```

### **Фрагмент Б.6 — Навігація між сторінками:**

```

function navigate(page) {
  state.currentPage = page;
  render();
}

```

### **Фрагмент Б.7 — Відображення розкладу:**

```

function renderSchedule(items) {
  return h('div', {},
    ...items.map(item =>
      h('div', { class: 'schedule-item' },
        h('span', {}, item.start_time + ' - ' + item.end_time),

```

```

        h('span', {}, item.subject_name)
    )
)
);
}

```

## Додаток В. Скріншоти інтерфейсу користувача

Рисунок В.1 — Сторінка входу до системи

**E** **EduSystem**  
ОСВІТНЯ ПЛАТФОРМА

# Вхід

Авторизуйтеся, щоб отримати доступ до розкладу та матеріалів.

ЕЛЕКТРОННА ПОШТА

user@edu.local

ПАРОЛЬ

.....

Заповніть це поле.

**Увійти**

---

Ще немає облікового запису? [Зареєструватися](#)

## В.2 — Сторінка реєстрації користувача

**E**

**EduSystem**  
ОСВІТНЯ ПЛАТФОРМА

# Реєстрація

Створіть обліковий запис студента або викладача.

ПРИЗВИЩЕ ТА ІМ'Я

ЕЛЕКТРОННА ПОШТА

ПАРОЛЬ (МІНІМУМ 6 СИМВОЛІВ)

РОЛЬ

Студент

Викладач

Створити обліковий запис

---

Уже маєте обліковий запис? [Увійти](#)

Рисунок В.3 — Головна сторінка (панель користувача)

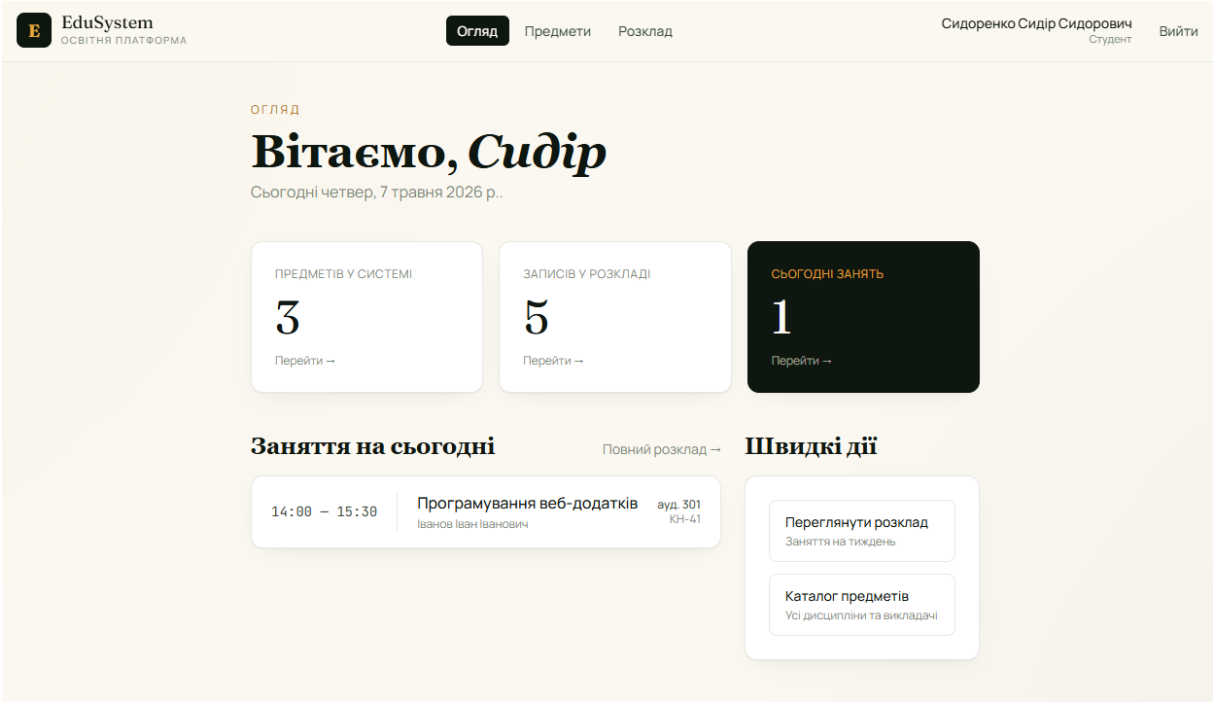


Рисунок В.4 — Список навчальних дисциплін

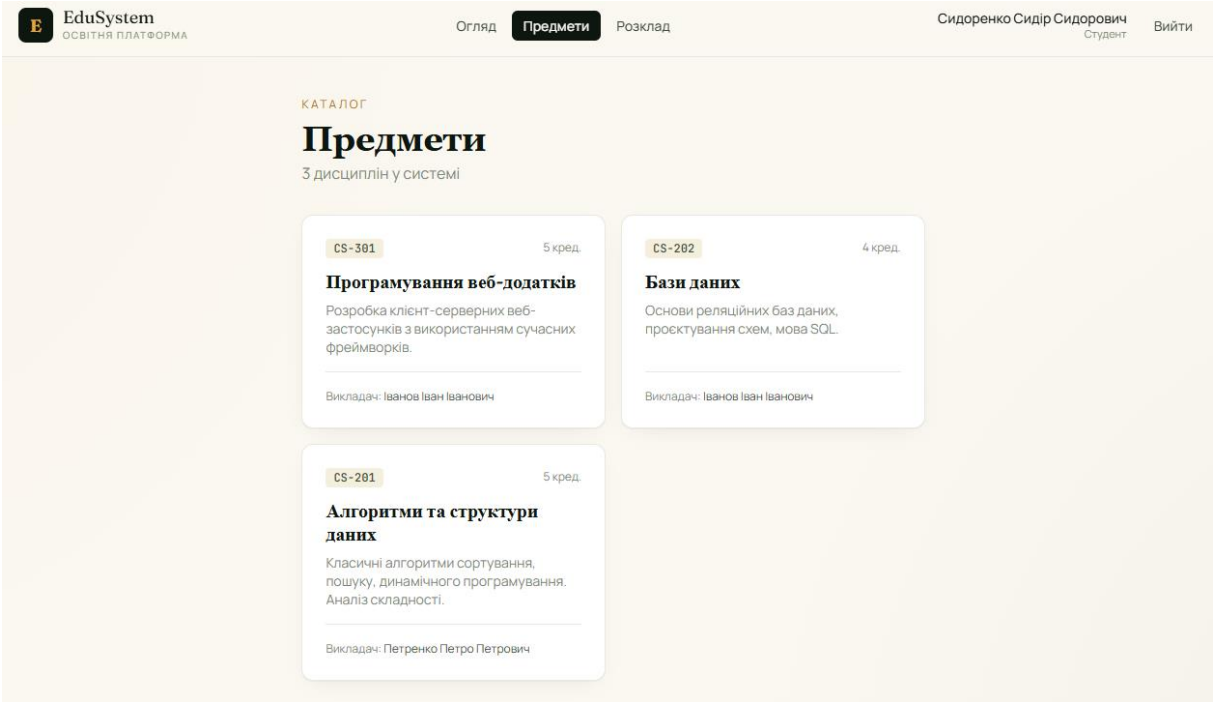


Рисунок В.5 — Розклад занять на тиждень

EduSystem

ОСВІТНЯ ПЛАТФОРМА

Огляд

Предмети

Розклад

Сидоренко Сидір Сидорович

Студент

Вийти

ТИЖДЕНЬ

Розклад

5 занять усього

Понеділок

Пн

08:30–10:00

ауд. 301

Програмування веб-додатків

Іванов Іван Іванович

КН-41

Вівторок

Вт

10:15–11:45

ауд. 305

Бази даних

Іванов Іван Іванович

КН-41

Середа

Ср

12:15–13:45

ауд. 210

Алгоритми та структури даних

Петренко Петро Петрович

КН-41

Четвер

Сьогодні

Чт

14:00–15:30

ауд. 301

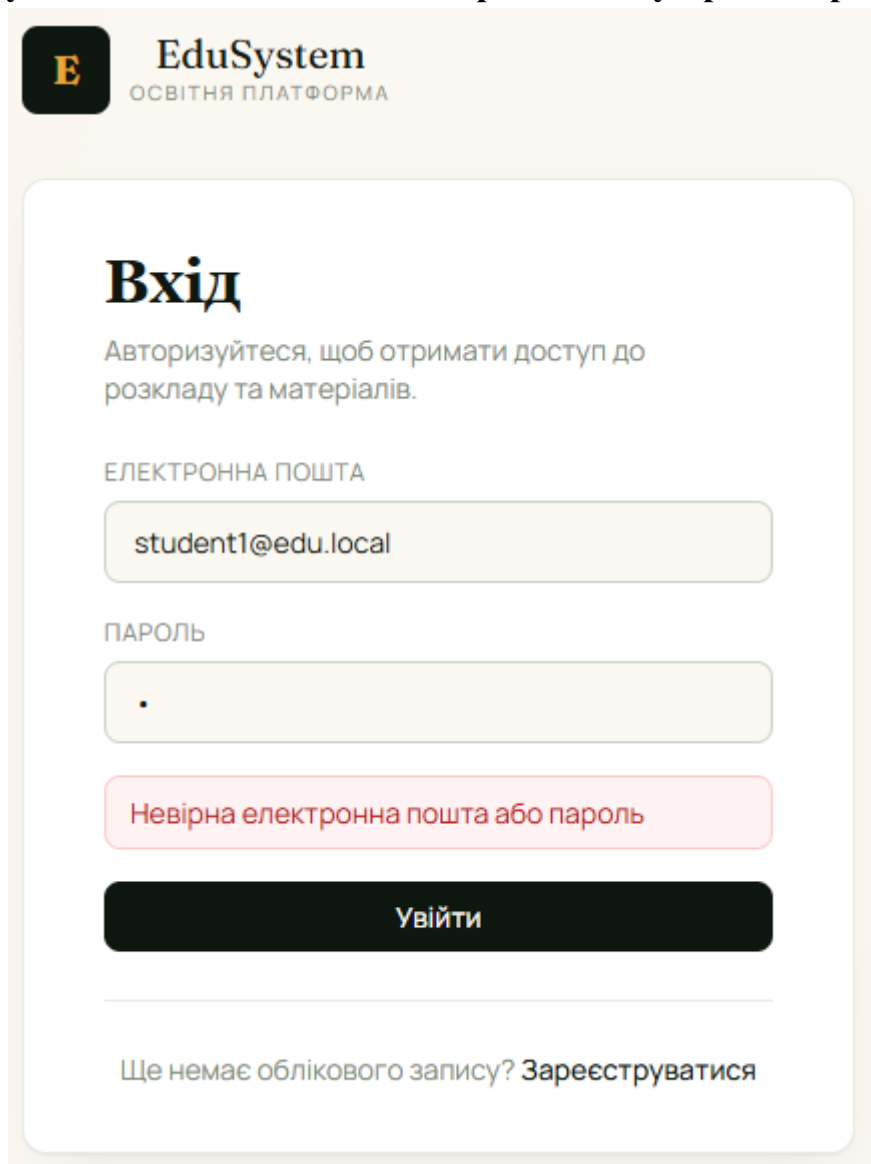
Програмування веб-додатків

Іванов Іван Іванович

КН-41

П'ятниця

Пт

**Рисунок В.6 — Повідомлення про помилку при авторизації**

The image shows a login interface for 'EduSystem' (Освітня платформа). At the top left is a logo with a yellow 'E' on a black square. The title 'EduSystem' is in a large, bold, black serif font, with 'ОСВІТНЯ ПЛАТФОРМА' in a smaller, grey sans-serif font below it. The main heading 'Вхід' is in a large, bold, black serif font. Below it is the instruction 'Авторизуйтеся, щоб отримати доступ до розкладу та матеріалів.' in a grey sans-serif font. There are two input fields: 'ЕЛЕКТРОННА ПОШТА' containing 'student1@edu.local' and 'ПАРОЛЬ' containing a single dot. A red error message 'Невірна електронна пошта або пароль' is displayed in a light red box. Below this is a black 'Увійти' button. At the bottom, a link 'Ще немає облікового запису? Зареєструватися' is shown in a grey sans-serif font.

**E** **EduSystem**  
ОСВІТНЯ ПЛАТФОРМА

## Вхід

Авторизуйтеся, щоб отримати доступ до розкладу та матеріалів.

ЕЛЕКТРОННА ПОШТА

student1@edu.local

ПАРОЛЬ

.

Невірна електронна пошта або пароль

**Увійти**

Ще немає облікового запису? [Зареєструватися](#)

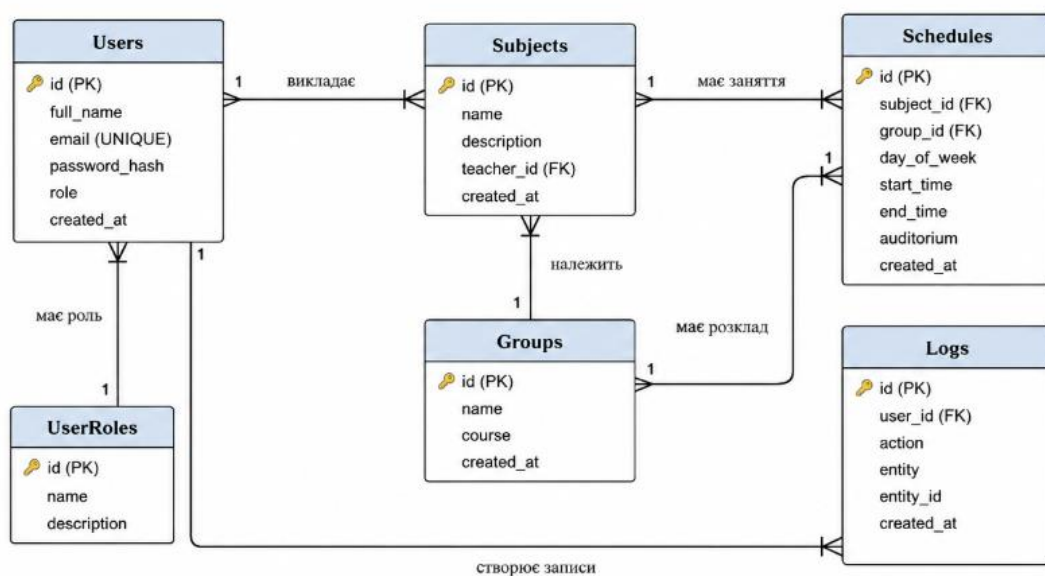
### Додаток Г. Схема бази даних (ER-діаграма)

У даному додатку представлено схему бази даних інформаційної системи управління освітнім процесом у вигляді ER-діаграми. Дана діаграма відображає основні сутності системи, їх атрибути та логічні зв'язки між ними.

База даних спроектована з урахуванням принципів нормалізації, що дозволяє уникнути надлишковості даних та забезпечити їх цілісність. Основними сутностями є користувачі, навчальні дисципліни та розклад занять. Між сутностями встановлено зв'язки типу «один-до-багатьох», що забезпечує правильну організацію структури даних.

#### Рисунок Г.1 — ER-діаграма бази даних

ER-діаграма відображає основні сутності системи управління освітнім процесом, їх атрибути та зв'язки між ними.



Позначення:  
 1 — один  
 ≡ — багато