Вінницький національний технічний університет
(повне найменування вищого навчального закладу)

Факультет інформаційних технологій та комп'ютерної інженерії
(повне найменування інституту, назва факультету (відділення))

Кафедра обчислювальної техніки
(повна назва кафедри (предметної, циклової комісії))

БАКАЛАВРСЬКА КВАЛІФІКАЦІЙНА РОБОТА

на тему:

«Навчально-інформаційна система з використанням C# та Next.JS»

08-54.БКР.023.00.000 ПЗ

Виконав: студент 2 курсу, групи 2КІ-23мс

спеціальності 123 — Комп'ютерна інженерія
(шифр і назва напряму підготовки, спеціальності)

_____Воробйов Я. І.
(підпис)

Керівник: ст. викл. каф. ОТ

_____Кисюк Д. В.
(підпис)

«_____» _____2025 р.

Рецензент: д. т. н., проф., зав. каф. ОТ

_____Романюк О. Н.
(підпис)

«_____» _____2025 р.

Допущено до захисту

Завідувач кафедри ОТ

д.т.н., проф. Азаров О. Д.

«10» 06 2025 р.

Вінниця ВНТУ — 2025 рік

ВІННИЦЬКИЙ НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ

Факультет інформаційних технологій та комп'ютерної інженерії

Кафедра обчислювальної техніки

Освітньо-кваліфікаційний рівень бакалавр

Галузь знань — 12 «Інформаційні технології»

Спеціальність — 123 «Комп'ютерна інженерія»

Освітньо-професійна програма «Комп'ютерна інженерія»

ЗАТВЕРДЖУЮ

Завідувач кафедри ОТ

д.т.н., проф. Азаров О. Д.

«21» березня 2025 р.

З А В Д А Н Н Я

НА БАКАЛАВРСЬКУ КВАЛІФІКАЦІЙНУ РОБОТУ СТУДЕНТУ

Воробйову Ярославу Івановичу

1 Тема роботи: Навчально-інформаційна система з використанням C# та Next.JS, керівник роботи ст. викл. каф. ОТ Кисюк Д. В., затверджена наказом вищого навчального закладу від 20 березня 2025 р. №97.

2 Термін подання студентом роботи 13 травня 2025 р.

3 Вихідні дані до роботи: клієнтська та серверна частини інформаційної системи для керування навчальною інформаційною системою, інтуїтивно зрозумілий інтерфейс, відкритий API для створення додатків для різних ОС.

4 Зміст розрахунково-пояснювальної записки: вступ, аналіз предметної області, проєктування та тестування серверної та клієнтської частини.

5 Перелік графічного матеріалу: архітектура проєкту, діаграма взаємодії навчально-інформаційної системи, схема інтеграції навчально-інформаційної системи, діаграма розгортання навчально інформаційної системи

6 Консультанти розділів роботи приведені в таблиці 1

Таблиця 1 — Консультанти роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
1-3	ст. викл. каф. ОТ Кисюк Д.В.	21.03.25	13.05.25
Нормоконтроль	ас. каф. ОТ Швець С. І.	13.05.25	30.05.25

7 Дата видачі завдання 21 березня 2025 р.

8 Календарний план виконання БКР приведений в таблиці 2.

Таблиця 2 — Календарний план виконання БКР

№	Назва етапу	Термін виконання	Підпис
1	Вибір, узгодження та затвердження теми БКР	21.03.25	Вик
2	Аналіз існуючих програмних рішень та визначення вимог системи	22.03.25— 23.03.25	Вик
4	Проєктування архітектури системи та бази даних	24.03.25— 28.03.25	Вик
5	Розробка бекенду на ASP.NET Core та Entity Framework	29.03.25— 13.04.25	Вик
6	Реалізація системи автентифікації та авторизації з JWT токенами та OAuth2 Google	14.04.25— 20.04.25	Вик
7	Реалізація API та налаштування документації	21.04.25— 22.04.25	Вик
8	Розробка фронтенду на Next.js з використанням Tailwind CSS та DaisyUI	23.04.25— 27.04.25	Вик
9	Інтеграція та тестування системи	28.04.25— 04.05.25	Вик
11	Написання документації та пояснювальної записки	05.05.25— 12.05.25	Вик
12	Підготовка супровідної документації, перевірка відповідності нормам та проходження тесту на плагіат	13.05.25	Вик

Студент Ярослав ВОРОБЙОВ

Керівник роботи ст. викл. каф. ОТ Дмитро КИСЮК

АНОТАЦІЯ

УДК 004.72

Воробйов Я. І. Навчально-інформаційна система з використанням C# та Next.JS. Бакалаврська кваліфікаційна робота зі спеціальності 123 «Комп'ютерна інженерія», освітня програма «Комп'ютерна інженерія». Вінниця: ВНТУ, 2025. 148 с.

На укр. мові. Бібліогр.: 34 назв, рис.: 53, табл. 12.

Бакалаврську кваліфікаційну роботу було розроблено для створення навчально-інформаційної системи з використанням Next.js та ASP.NET Core. Реалізовано клієнт-серверну архітектуру з Entity Framework Core для роботи з базами даних SQLite та PostgreSQL, JWT-автентифікацією та OAuth через Google. Побудовано за ієрархічним принципом (класи, предмети, завдання) з можливістю прикріплення матеріалів та відображенням через Markdown. Інтерфейс реалізовано з використанням Tailwind CSS та DaisyUI, що забезпечує адаптивність. API документовано за допомогою Swagger, що спрощує інтеграцію та подальшу розробку.

Ключові слова: навчально-інформаційна система, Next.js, ASP.NET Core, SQLite, PostgreSQL, JWT, OAuth 2.0, Swagger, Tailwind CSS, ієрархічна модель навчання.

ABSTRACT

UDC 004.72

Vorobyov Y. I. Educational and Information System Using C# and Next.JS. Bachelor's qualification work in specialty 123 «Computer Engineering», educational program «Computer Engineering». Vinnytsia: VNTU, 2025. 148 p.

In Ukrainian. Bibliography: 53 titles, figs.: 36, tables: 12.

The bachelor's qualification work was developed to create an educational information system using Next.js and ASP.NET Core. A client-server architecture with Entity Framework Core was implemented to work with SQLite and PostgreSQL databases, JWT authentication and OAuth via Google. Built on a hierarchical principle (classes, subjects, tasks) with the ability to attach materials and display via Markdown. The interface was implemented using Tailwind CSS and DaisyUI, which ensures adaptability. The API is documented using Swagger, which simplifies integration and further development.

Keywords: educational information system, Next.js, ASP.NET Core, SQLite, PostgreSQL, JWT, OAuth 2.0, Swagger, Tailwind CSS, hierarchical learning model.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СКОРОЧЕНЬ, ТЕРМІНІВ

API — Application Programming Interface

ASP.NET Core — платформа для розробки веб-додатків від Microsoft

БД — база даних

БКР — бакалаврська кваліфікаційна робота

CSS — Cascading Style Sheets

CRUD — Create, Read, Update, Delete

DaisyUI — бібліотека компонентів для Tailwind CSS

EF Core — Entity Framework Core

HTML — HyperText Markup Language

HTTP — HyperText Transfer Protocol

JSON — JavaScript Object Notation

JWT — JSON Web Token

LINQ — Language Integrated Query

OAuth2 — Open Authorization 2.0

ORM — Object-Relational Mapping

REST — Representational State Transfer

SQL — Structured Query Language

UI/UX — User Interface/User Experience

URL — Uniform Resource Locator

ВСТУП

У умовах стрімкого розвитку інформаційних технологій та цифровізації освітнього процесу зростає потреба в ефективних інструментах для організації навчання. Сучасні освітні установи все частіше звертаються до спеціалізованих програмних рішень, які дозволяють автоматизувати адміністративні процеси, забезпечити зручну взаємодію між викладачами та студентами, а також підвищити якість освітніх послуг.

Навчально-інформаційні системи стають невід'ємною частиною освітнього процесу, особливо в умовах зростання популярності дистанційного та змішаного навчання. Ці системи дозволяють структурувати навчальні матеріали, організовувати процес виконання та перевірки завдань, відстежувати прогрес студентів та забезпечувати комунікацію між усіма учасниками освітнього процесу.

Актуальність даної роботи полягає в розробці сучасної навчально-інформаційної системи, яка поєднує в собі передові технології веб-розробки, забезпечуючи високу продуктивність, масштабованість та зручний користувацький інтерфейс. Використання технологій Next.js для фронтенду та ASP.NET Core для бекенду дозволяє створити надійну та ефективну платформу, яка відповідає сучасним вимогам до вебзастосунків.

Метою бакалаврської роботи є підвищення якості організації навчального процесу, для чого спроєктовано та розроблено навчально-інформаційну систему, що дозволяє ефективно організовувати навчальний процес за ієрархічною моделлю, забезпечувати зручну взаємодію між викладачами та студентами, а також автоматизувати рутинні процеси в освітньому середовищі.

Для досягнення поставленої мети необхідно вирішити наступні **завдання**:

— проаналізувати існуючі рішення та визначити ключові вимоги до системи;

— спроєктувати архітектуру клієнт-серверного застосунку з використанням сучасних технологій;

— розробити серверну частину на базі ASP.NET Core з підтримкою різних СУБД;

— створити клієнтську частину з використанням фреймворку Next.js та бібліотек для сучасного користувацького інтерфейсу;

— впровадити систему автентифікації та авторизації з підтримкою різних методів входу;

— реалізувати ієрархічну модель організації навчальних матеріалів;

— протестувати розроблену систему та оцінити її ефективність.

Об'єкт дослідження — це процес організації та управління навчальною діяльністю в освітніх установах з використанням сучасних веб-технологій та інформаційних систем.

Предметом дослідження є методи та засоби розробки навчально-інформаційних систем з використанням фреймворків Next.js та ASP.NET Core.

Новизна результатів полягає у використанні комплексного підходу до створення навчально-інформаційної системи з об’єднанням у собі технології ASP.NET Core, ASP.NET Identity та SignalR для забезпечення ефективної взаємодії між учасниками освітнього процесу та автоматизації управління навчальними матеріалами.

Апробація отриманих результатів відбулася під час розробки та тестування функціонального прототипу системи, що підтвердив ефективність рішень.

Практична цінність роботи полягає в створенні повнофункціонального програмного продукту, який може бути впроваджений в освітніх установах різного рівня для оптимізації навчального процесу та підвищення його ефективності.

Матеріали роботи були представлені та **апробовані**[1]:

Навчально-інформаційна система з використанням C# та Next.JS. [Електронний ресурс] / Кисюк Д. В., Воробйов Я. І.. // Матеріали Всеукраїнської науково-практичної інтернет-конференції «Молодь в науці: дослідження, проблеми, перспективи (МН-2025)», Вінниця, 16 червня 2025р. Режим доступу: https://conferences.vntu.edu.ua/index.php/mn/mn2025/paper/view/ 25271

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ ІНФОРМАЦІЙНОЇ СИСТЕМИ ОБЛІКУ НАВЧАЛЬНО-ІНФОРМАЦІЙНОЇ СИСТЕМИ

1.1 Загальна характеристика предметної області

Навчально-інформаційні системи — це спеціалізовані програмні комплекси для автоматизації освітнього процесу, що поєднують функціонал систем управління навчанням (LMS) та інформаційних систем освітніх закладів.

Основними компонентами предметної області є:

— управління користувачами та ролями;

— структурування навчального контенту (клас, предмет, тема, заняття);

— організація навчального процесу (планування, формування розкладу тощо);

— створення та розповсюдження матеріалів;

— оцінювання знань;

— комунікація;

— інтеграція з іншими системами;

— безпека даних.

Сучасні тенденції в розвитку навчально-інформаційних систем включають персоналізацію навчання, гейміфікацію, використання елементів штучного інтелекту та розширення можливостей для колаборативної роботи.

1.2 Аналіз існуючих рішень

Існують різноманітні навчально-інформаційні системи, що використовуються у світовій та вітчизняній освітній практиці. Розглянемо найбільш поширені рішення, щоб визначити їхні переваги, недоліки та функціональні можливості.

Google Classroom — популярна навчально-інформаційна система від Google, що інтегрується з іншими їхніми сервісами (Drive, Docs, Forms, Calendar). Переваги: інтуїтивний інтерфейс, безкоштовність для освітніх установ,

мультиплатформність. Недоліки: негнучка система оцінювання, обмежене структурування матеріалів, залежність від екосистеми Google (рисунок 1.1).

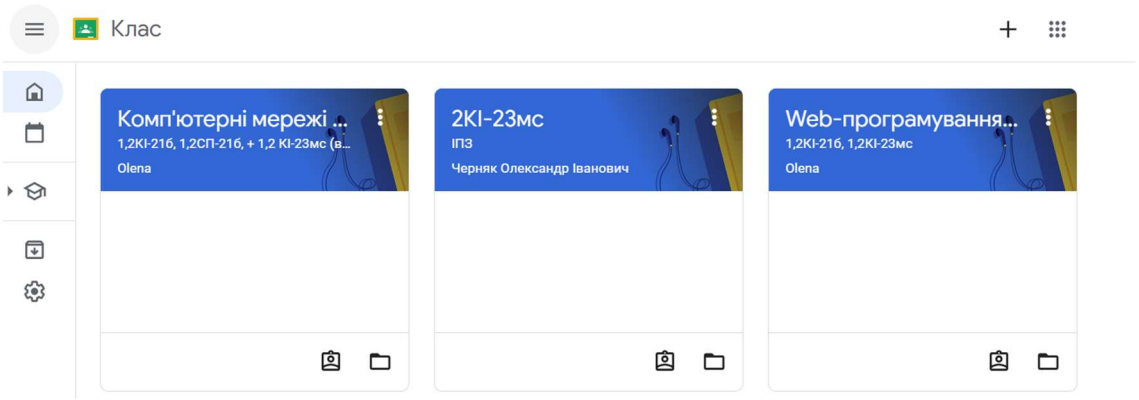


Рисунок 1.1 — Головне меню Google Classroom

Microsoft Teams for Education від Microsoft спеціалізується на дистанційному навчанні. Сильні сторони: відеоконференції, колаборативна робота, інтеграція з Office 365. Недоліки: складний інтерфейс, високі вимоги до інтернет-з'єднання, фокус на комунікації замість структурування навчальних матеріалів (рисунок 1.2).

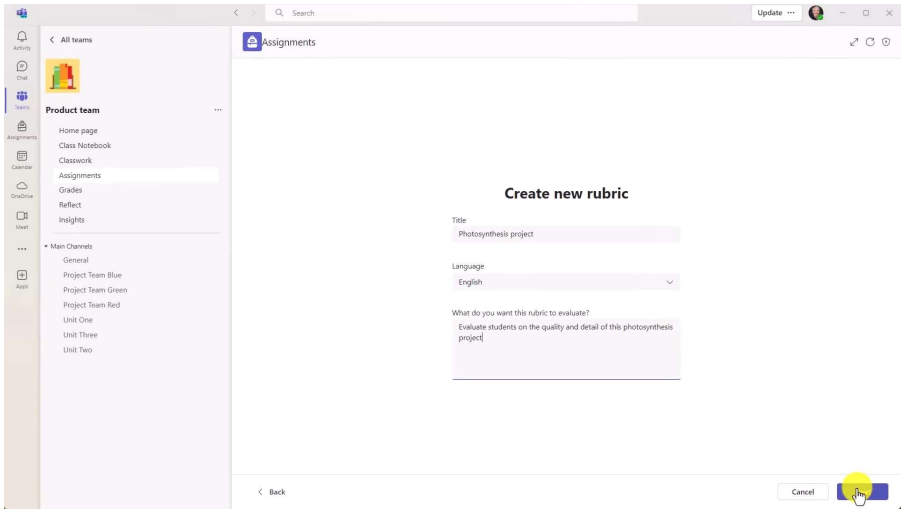
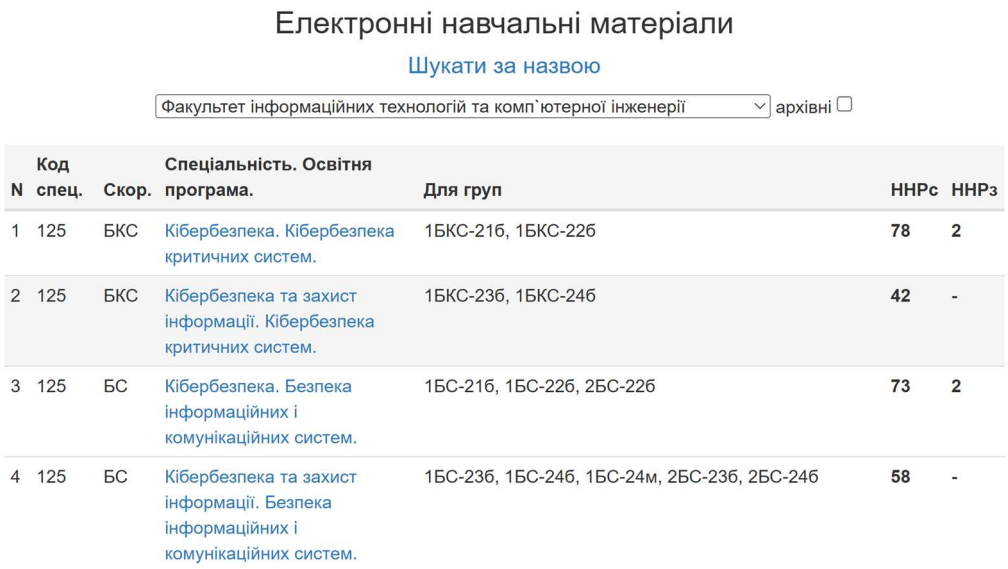


Рисунок 1.2 — Створення нової рубрики в Microsoft Teams for Education

JetIQ — власна система для Вінницького національного технічного університету з модулями електронного деканату, журналу, бібліотеки, тестування. Переваги: адаптація до українського освітнього процесу, комплексна автоматизація (рисунок 1.3).



N	Код спец.	Скор.	Спеціальність. Освітня програма.	Для груп	ННРс	ННРз
1	125	БКС	Кібербезпека. Кібербезпека критичних систем.	1БКС-21б, 1БКС-22б	78	2
2	125	БКС	Кібербезпека та захист інформації. Кібербезпека критичних систем.	1БКС-23б, 1БКС-24б	42	-
3	125	БС	Кібербезпека. Безпека інформаційних і комунікаційних систем.	1БС-21б, 1БС-22б, 2БС-22б	73	2
4	125	БС	Кібербезпека та захист інформації. Безпека інформаційних і комунікаційних систем.	1БС-23б, 1БС-24б, 1БС-24м, 2БС-23б, 2БС-24б	58	-

Рисунок 1.3 — Перегляд електронних навчальних матеріалів в JetIQ

Таблиця 1.1 — Порівняльна характеристика різних навчально-інформаційних систем

Характеристика	Google Classroom	Microsoft Teams for Education	JetIQ
Доступність	Безкоштовно для освітніх установ	Безкоштовно з обмеженнями, повна версія платна	Ліцензія для навчальних закладів
Інтеграція	Екосистема Google	Екосистема Microsoft	Обмежена інтеграція з зовнішніми сервісами
Ієрархія навчальних матеріалів	Обмежена (курси, матеріали, завдання)	Середня (команди, канали, матеріали)	Розвинена (факультет, кафедра, дисципліна, тема)
Оцінювання	Базова система оцінювання	Розширена, з інтеграцією з Forms	Комплексна система з різними типами оцінювання
Комунікація	Коментарі та оголошення	Потужні інструменти (чати, відеоконференції, канали)	Обмежені можливості
Адаптивний інтерфейс	Так	Так	Частково
API для інтеграції	Обмежене	Розширене	Обмежене

1.3 Обґрунтування необхідності розробки нової системи

Аналіз існуючих навчально-інформаційних систем виявив ряд обмежень, які впливають на ефективність їх використання в сучасному освітньому процесі. Незважаючи на функціональність Google Classroom, Microsoft Teams for Education та JetIQ, жодне з цих рішень не забезпечує оптимального поєднання всіх необхідних характеристик для повноцінного задоволення потреб сучасного освітнього середовища.

Основні недоліки існуючих систем, які обґрунтовують необхідність розробки нового рішення:

— обмежені можливості для ієрархічної організації навчального контенту;

— недостатня гнучкість та адаптивність;

— залежність від конкретних екосистем;

— складність налаштування та адміністрування;

Google Classroom пропонує лише дворівневу структуру (курс, матеріали/завдання), що недостатньо для складної організації навчального процесу, при цьому Microsoft Teams орієнтований на комунікацію та має обмежені можливості для структурування матеріалів. JetIQ, хоча і має розвинену ієрархію, але страждає від складності інтерфейсу, що ускладнює навігацію. Існуючі системи часто мають фіксований функціонал, який складно адаптувати під специфічні потреби різних навчальних закладів; обмежені можливості для розширення функціоналу та інтеграції з іншими системами знижують їх ефективність у динамічному освітньому середовищі. Google Classroom тісно інтегрований з екосистемою Google, а Microsoft Teams — з продуктами Microsoft, що створює залежність від одного постачальника та може викликати проблеми з приватністю даних та доступністю.

Багато існуючих систем вимагають спеціальних знань для налаштування та адміністрування, що ускладнює їх впровадження та використання у навчальних закладах з обмеженими технічними ресурсами.

1.4 Платформа ASP.NET Core

ASP.NET Core є сучасною, високопродуктивною, кросплатформеною версією платформи ASP.NET від Microsoft. Ця технологія представляє собою відкрите програмне забезпечення для розробки вебдодатків та вебсервісів, що може працювати на операційних системах Windows, Linux та macOS [5].

Ключовими перевагами ASP.NET Core, які обумовили вибір цієї технології для розробки серверної частини навчально-інформаційної системи, є:

— висока продуктивність: оптимізована архітектура та компіляція коду забезпечують швидке виконання та обробку запитів;

— модульність: система побудована на принципах модульності, що дозволяє використовувати лише необхідні компоненти;

— кросплатформеність: можливість розгортання застосунків на різних операційних системах, що забезпечує гнучкість у виборі інфраструктури;

— вбудована підтримка контейнерів: інтеграція з Docker для спрощення розгортання та масштабування;

— уніфікована модель програмування: використання єдиного шаблону MVC для веб-сторінок та API;

— вбудована підтримка DI (Dependency Injection): полегшує тестування та підтримку модульності коду;

У розробленій системі ASP.NET Core використовується для створення RESTful API, що забезпечує взаємодію між клієнтською частиною та сервером, а також для реалізації бізнес-логіки та роботи з базами даних.

1.5 HTTP-методи

HTTP протокол визначає набір методів, які використовуються для взаємодії з ресурсами на сервері [24]. Ці методи є основою для створення RESTful API та реалізації CRUD операцій (Create, Read, Update, Delete), що дозволяють виконувати базові дії з даними (рисунок 1.4).

HTTP метод	CRUD операція	Призначення
GET	Read	Отримання даних без змін на сервері
POST	Create	Створення нових ресурсів на сервері
PUT/PATCH	Update	Оновлення існуючих ресурсів на сервері
DELETE	Delete	Видалення ресурсів з сервера

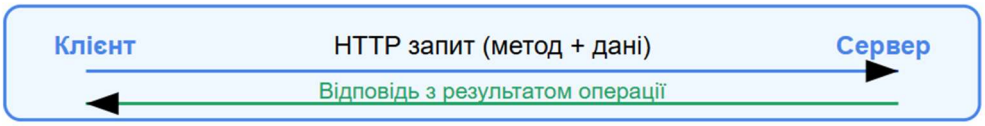


Рисунок 1.4 — HTTP-методи та їх відповідність CRUD операціям

Основні HTTP методи включають:

— GET — використовується для читання інформації, наприклад, отримання списку класів або деталей конкретного завдання;

— POST — застосовується для додавання нових даних, таких як створення нового класу, предмета чи користувача;

— PUT — метод замінює всі поточні дані ресурсу новими, що передаються у запиті;

— PATCH — на відміну від PUT, дозволяє модифікувати лише окремі поля ресурсу без необхідності передавати всі дані;

— DELETE — використовується для вилучення класів, предметів, завдань або користувачів;

— OPTIONS — корисний для виявлення можливостей API, визначення підтримуваних методів запитів, перевірки обмежень CORS, отримання метаінформації про ресурс та автоматизації клієнтських інтерфейсів без необхідності розгорнутої документації;

— HEAD — може використовуватися для перевірки наявності ресурсу та отримання метаданих.

Таблиця 1.2 — Характеристики HTTP-методів

HTTP метод	CRUD операція	Призначення	Особливості
GET	Read	Отримання даних	Не змінює стан сервера і може багаторазово викликатися з однаковим результатом
POST	Create	Створення ресурсу	Змінює стан сервера і повторні виклики створюють нові ресурси
PUT	Update	Повне оновлення	Повторні виклики дають однаковий результат без додаткових змін
PATCH	Update	Часткове оновлення	Змінює лише зазначені властивості ресурсу
DELETE	Delete	Видалення ресурсу	Після успішного видалення повторні виклики не змінюють стан сервера
OPTIONS	-	Інформація про методи	Не змінює стан сервера
HEAD	-	Метадані ресурсу	Не змінює стан сервера

Ці методи є основою для створення RESTful API та реалізації CRUD операцій (Create, Read, Update, Delete).

1.6 Системи управління базами даних

Бази даних є невід'ємною частиною сучасних інформаційних систем, забезпечуючи надійне та структуроване зберігання даних. Вони поділяються на два основні типи: реляційні та нереляційні.

Реляційні бази даних організовують дані у вигляді таблиць з рядками та стовпцями, пов'язаних між собою за допомогою ключів. Вони використовують SQL

(Structured Query Language) для маніпуляції даними та забезпечують атомарність, узгодженість, ізольованість та довговічність операцій (ACID). Реляційні БД ідеально підходять для структурованих даних з чіткими зв'язками.

Нереляційні бази даних (NoSQL) не використовують табличну структуру та пропонують альтернативні моделі зберігання даних: документи, ключ-значення, графи або колонки. Вони зазвичай краще масштабуються горизонтально та забезпечують гнучкіші схеми даних, що особливо корисно при роботі з неструктурованими або частково структурованими даними.

Системи управління базами даних у проєкті

У розробленій навчально-інформаційній системі використовуються дві реляційні СУБД:

SQLite — компактна вбудована реляційна база даних, яка не потребує окремого сервера та зберігає всю базу даних у єдиному файлі. SQLite ідеально підходить для локальної розробки, тестування та невеликих застосунків [17]. Її особливостями є:

— відсутність необхідності в окремому сервері;

— нульова конфігурація — працює «з коробки»;

— повна підтримка ACID-транзакцій;

— крос-платформність та портативність;

— низькі вимоги до ресурсів;

PostgreSQL — потужна об'єктно-реляційна СУБД з відкритим кодом [9], що забезпечує високу надійність, масштабованість та відповідність стандартам. Її ключові особливості:

— повна підтримка ACID та складних транзакцій;

— розширена підтримка JSON та інших нереляційних структур даних;

— потужні індекси та оптимізація запитів;

— вбудована підтримка реплікації та шардингу;

— розширюваність через власні типи даних та функції;

— висока відмовостійкість та захист даних.

Таблиця 1.3 — Порівняння SQL-баз даних з NoSQL

Характеристика	SQLite	PostgreSQL	MongoDB
Тип	Реляційна	Реляційна (об'єктно-реляційна)	Нереляційна (документоорієнтована)
Модель розгортання	Вбудована (файлова)	Клієнт-сервер	Клієнт-сервер
Мова запитів	SQL	SQL (з розширеннями)	MongoDB Query Language
Схема даних	Фіксована	Фіксована (з підтримкою JSON)	Гнучка (схема на запис)
Масштабованість	Низька	Висока	Дуже висока
Підтримка транзакцій	Повна (ACID)	Повна (ACID)	Обмежена (ACID на рівні документа)
Продуктивність для читання	Висока для невеликих БД	Середня	Дуже висока
Продуктивність для запису	Низька при паралельних операціях	Середня	Дуже висока
Типові випадки використання	Локальний розвиток, малі сайти	Корпоративні системи, аналітика, системи з вимогами до цілісності	Великі обсяги даних, IoT, реальний час, мобільні програми
Підтримка JSON	Обмежена	Повна	Нативна (BSON)
Горизонтальне масштабування	Не підтримується	Підтримується через розширення	Вбудоване (шардінг)

1.7 Документація API

Swagger (OpenAPI) — це потужний інструмент для документування, тестування та візуалізації RESTful API [13]. У розробленій навчально-інформаційній системі Swagger відіграє важливу роль, забезпечуючи зрозумілий та інтерактивний опис доступних кінцевих точок, методів та моделей даних.

Swagger забезпечує стандартизований підхід до опису API, що дозволяє як розробникам, так і кінцевим користувачам легко зрозуміти функціональність та можливості API. Основні переваги використання Swagger у проєкті:

— автоматична документація: swagger автоматично генерує та оновлює документацію на основі коду API, що забезпечує її актуальність та відповідність реальній реалізації;

— інтерактивний інтерфейс: Swagger UI надає зручний веб-інтерфейс для перегляду та тестування API безпосередньо з браузера, що значно спрощує процес розробки та налагодження;

— спрощення тестування: можливість виконувати запити до API та переглядати відповіді безпосередньо з інтерфейсу Swagger спрощує тестування та перевірку правильності роботи API;

— стандартизація API: використання Swagger допомагає дотримуватися найкращих практик проєктування API та забезпечує уніфікований підхід до структури кінцевих точок;

— полегшення інтеграції: чітка та зрозуміла документація API спрощує інтеграцію з іншими системами та розробку клієнтських застосунків;

— генерація клієнтського коду: Swagger дозволяє автоматично генерувати клієнтські бібліотеки для різних мов програмування на основі специфікації API.

У ASP.NET Core інтеграція Swagger реалізується за допомогою пакетів Swashbuckle.AspNetCore (рисунок 1.5), який забезпечує:

— автоматичне сканування контролерів та дій для виявлення кінцевих точок;

— автоматичне генерування документації на основі XML-коментарів до методів контролерів.

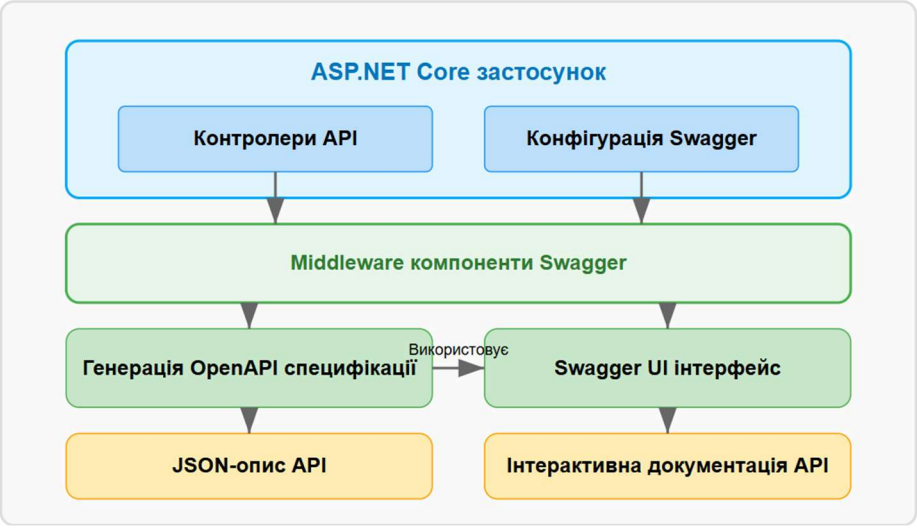


Рисунок 1.5 — Принцип роботи Swagger

Візуальний вигляд інтерфейсу Swagger демонструє документацію кінцевої точки авторизації, де представлено POST-запит до логіна з тілом запиту у форматі JSON, що містить поля для електронної пошти та паролю користувача (рисунок 1.6).

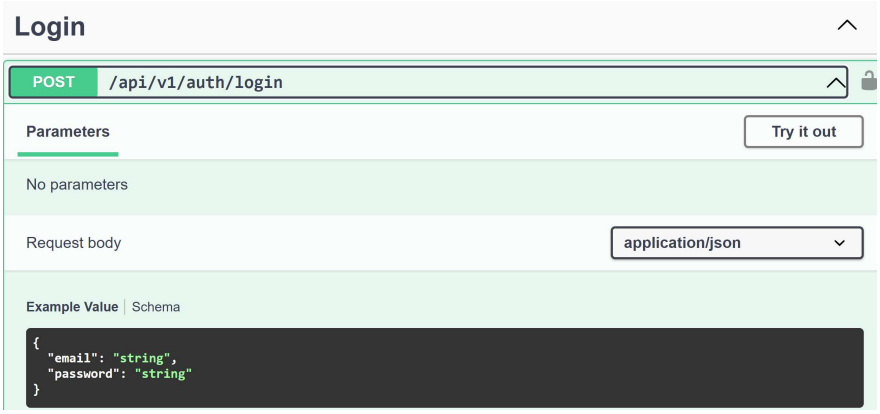


Рисунок 1.6 — Середовище Swagger

1.8 Бібліотеки для Frontend-розробки

Next.js — це потужний фреймворк для розробки веб-застосунків на основі React.js, створений компанією Vercel [4]. Він значно розширює можливості React, додаючи функціонал для серверного рендерингу, генерації статичних сайтів, оптимізації та спрощення процесу розробки (рисунок 1.7).

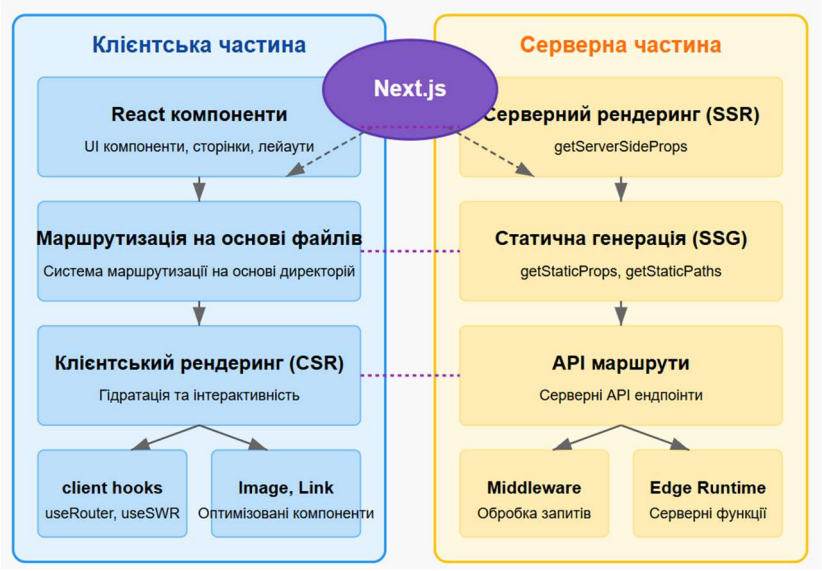


Рисунок 1.7 — Архітектура Next.js

Ключові особливості Next.js:

— маршрутизація на основі файлової системи — структура каталогів і файлів у директорії pages (або app у новіших версіях) автоматично відображається на URL-шляхи застосунку;

— вбудована оптимізація зображень — компонент Image автоматично оптимізує зображення, обробляє різні розміри для різних пристроїв та реалізує «ліниве» завантаження;

— API-маршрути — можливість створювати серверні кінцеві точки безпосередньо в проєкті, розміщуючи їх у директорії pages/api;

— автоматичне розділення коду — JavaScript-код автоматично розбивається на оптимальні пакети, які завантажуються лише коли потрібні;

— підтримка TypeScript — вбудована підтримка забезпечує типобезпеку під час розробки та автоматичне визначення типів;

— середовище розробки з гарячим перезавантаженням — зміни в коді автоматично відображаються в браузері без необхідності перезавантажувати сторінку;

— серверний рендеринг (SSR) та статична генерація (SSG) — підтримка різних стратегій рендерингу для оптимальної продуктивності та SEO.

Ключові особливості Next.js:

— маршрутизація на основі файлової системи;

— вбудована оптимізація зображень;

— API-маршрути;

— автоматичне розділення коду;

— підтримка TypeScript;

— середовище розробки з гарячим перезавантаженням;

— серверний рендеринг (SSR) та статична генерація (SSG).

Next.js реалізує інтуїтивно зрозумілу систему маршрутизації, де структура каталогів і файлів у директорії app автоматично відображається на URL-шляхи застосунку. Бібліотека надає компонент Image, який автоматично оптимізує

зображення, обробляє різні розміри для різних пристроїв та реалізує «ліниве» завантаження для покращення продуктивності. Next.js дозволяє створювати серверні кінцеві точки. безпосередньо в проєкті, розміщуючи їх у директорії pages/api, що спрощує розробку повноцінних застосунків з фронтендом та бекендом в одній кодовій базі. Автоматично розбиває JavaScript-код на оптимальні пакети, які завантажуються лише коли потрібні, що значно підвищує швидкість першого завантаження та загальну продуктивність. Має вбудовану підтримку TypeScript [11], що забезпечує типобезпеку під час розробки та автоматичне визначення типів. Під час розробки зміни в коді автоматично відображаються в браузері без необхідності перезавантажувати сторінку.

Next.js підтримує різні способи рендерингу сторінок:

— Server-Side Rendering (SSR) — генерація HTML на сервері для кожного запиту;

— Static Site Generation (SSG) — генерація HTML під час збірки;

— Incremental Static Regeneration (ISR) — оновлення статичних сторінок у фоновому режимі;

— Client-Side Rendering (CSR) — традиційний підхід React.

Приклад маршрутизації зображено на рисунку 1.8.

Файл в проєкті	URL адреса
app/page.js	/
app/about/page.js	/about
app/blog/[slug]/page.js	/blog/мій-пост
app/api/users/route.js	/api/users
app/products/[category]/[id]/page.js	/products/одяг/123

Рисунок 1.8 — Маршрутизація в Next.js

Переваги Next.js порівняно з чистим React:

— спрощена архітектура;

— серверний рендеринг з коробки;

— покращена продуктивність;

— вбудована маршрутизація;

— менше конфігурації;

— готові рішення для поширених задач;

— розширена екосистема;

— зручна розробка API.

Next.js надає готову структуру для проєкту, вирішуючи багато архітектурних питань, які в чистому React розробник повинен вирішувати самостійно. Хоча React може використовувати серверний рендеринг, в Next.js це реалізовано «з коробки» без необхідності додаткових налаштувань. Завдяки серверному рендерингу, статичній генерації та автоматичному розділенню коду, Next.js забезпечує кращу продуктивність та SEO порівняно зі стандартними SPA на Reac [16]t. Файлова система маршрутизації значно спрощує навігацію між сторінками порівняно з бібліотеками на кшталт React Router. Next.js має вбудовані налаштування Webpack, Babel та інших інструментів, що звільняє розробника від необхідності налаштовувати їх вручну. Бібліотека надає вбудовані рішення для оптимізації зображень, міжнародної локалізації, аналітики та інших поширених вимог веб-розробки. Next.js має потужну екосистему плагінів, прикладів та інтеграцій з популярними сервісами та інструментами. Файлова структура API-маршрутів дозволяє легко створювати та підтримувати серверні кінцеві точки в тому ж проєкті.

1.9 Tailwind CSS та DaisyUI

Tailwind CSS — це сучасний утилітарний CSS-фреймворк, який кардинально змінює підхід до стилізації веб-застосунків [7]. На відміну від традиційних фреймворків, які пропонують готові компоненти з попередньо визначеним

дизайном, Tailwind надає набір низькорівневих утилітарних класів, що безпосередньо відображають CSS-властивості.

Ці класи застосовуються безпосередньо в HTML-розмітці, що дозволяє швидко створювати унікальні дизайни без написання власних CSS-стилів. Tailwind використовує підхід «utility-first», де дизайн будується шляхом комбінування невеликих, однофункціональних класів, таких як flex, pt-4, text-center та rotate-90 (рисунок 1.9).

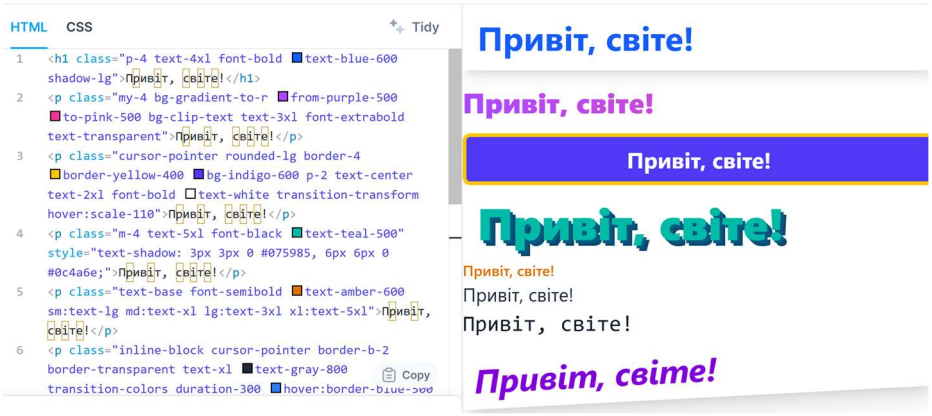


Рисунок 1.9 — Приклад використання tailwind css з «utility-first» підходом

Фреймворк включає широкий спектр попередньо визначених утиліт для роботи з кольорами, типографікою, відступами, гнучкими макетами, сітками та багатьма іншими аспектами дизайну. Це дозволяє створювати складні інтерфейси без необхідності перемикатися між HTML та CSS файлами, що значно пришвидшує процес розробки.

Tailwind також пропонує потужну систему конфігурації, яка дозволяє налаштувати та розширити базовий набір утиліт відповідно до вимог проєкту, включаючи визначення власних кольорів, розмірів, відступів та інших параметрів. Важливою особливістю Tailwind є його оптимізація для продакшн: під час збірки невикористані класи автоматично видаляються за допомогою PurgeCSS, що забезпечує мінімальний розмір CSS-файлу.

DaisyUI є компонентною бібліотекою [12], побудованою на основі Tailwind CSS, яка розширює його можливості, надаючи готові компоненти інтерфейсу користувача (рисунок 1.10).

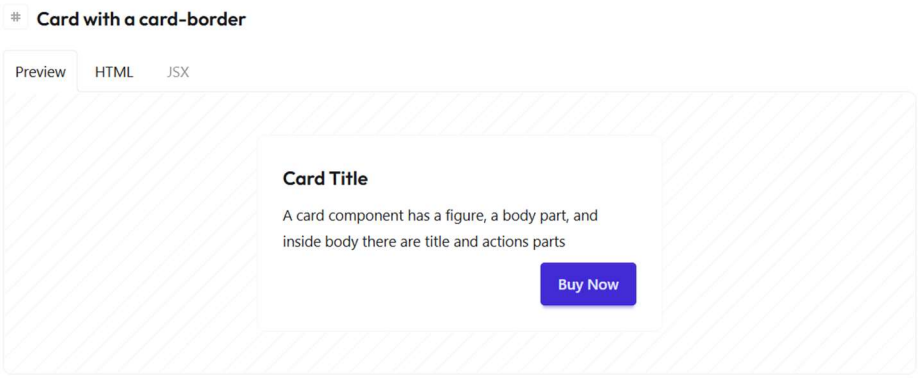


Рисунок 1.10 — Приклад створення картки в DaisyUI

DaisyUI зберігає перевагу утилітарного підходу Tailwind, але додає рівень абстракції у вигляді семантичних компонентних класів.

Замість комбінування багатьох утилітарних класів Tailwind, розробники можуть використовувати єдиний клас компонента, наприклад, btn для кнопок або card для карток. DaisyUI містить понад 40 інтерактивних компонентів користувацького інтерфейсу, включаючи кнопки, картки, модальні вікна, навігаційні панелі, форми та багато іншого.

Особливою перевагою DaisyUI є потужна система тем, яка підтримує не лише автоматичне перемикання між світлою та темною темами відповідно до системних налаштувань користувача, але й надає можливість створення та використання власних кастомізованих тем. Бібліотека включає декілька попередньо налаштованих тем, що відображають різні стилістичні напрямки та колірні схеми (рисунок 1.11).

Користувач може обирати між наявними темами або створювати власні, визначаючи основні кольори, відтінки, радіуси заокруглення та інші параметри дизайну. Ця функціональність особливо корисна для створення брендованих інтерфейсів або спеціалізованих тем для різних категорій користувачів, наприклад, із порушеннями зору або специфічними потребами відображення.

Адаптація до переваг користувача відбувається автоматично: інтерфейс може реагувати на системні налаштування пристрою або на користувацький вибір теми в самому застосунку. Це забезпечує персоналізований досвід користування без додаткових зусиль з боку розробників.

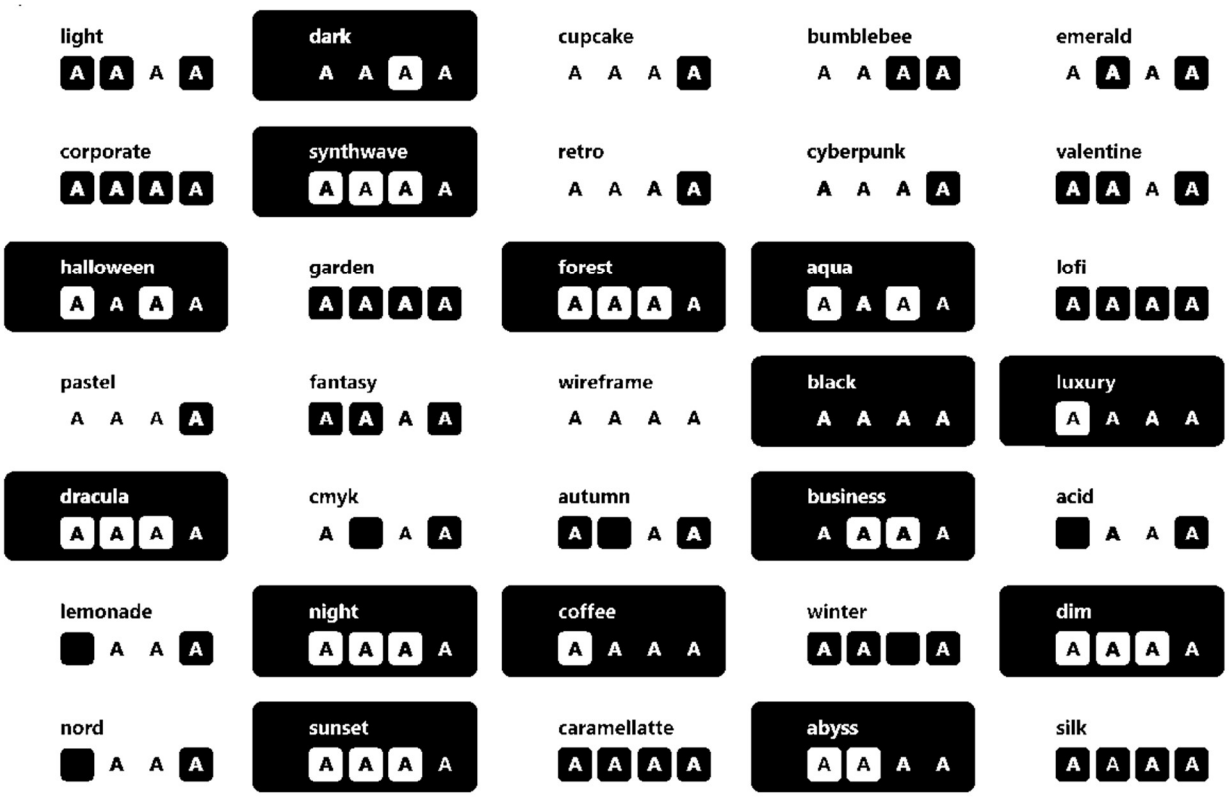


Рисунок 1.11 — Колірні схеми на DailyUI

DaisyUI працює як плагін для Tailwind, не заміняючи його, а доповнюючи. Це означає, що розробники можуть використовувати як компоненти DaisyUI, так і базові утиліти Tailwind в одному проєкті, комбінуючи швидкість розробки з готовими компонентами та гнучкість дрібного налаштування за допомогою утилітарних класів.

Використання Tailwind CSS разом з DaisyUI надає ряд переваг для розробки веб-інтерфейсів: пришвидшення процесу розробки завдяки готовим компонентам та класам; підтримка адаптивного дизайну без додаткових зусиль; автоматична адаптація до світлої та темної тем; консистентність інтерфейсу через використання стандартизованих компонентів; зменшення розміру CSS-файлів завдяки видаленню невикористаних стилів; швидкий старт розробки без необхідності розробляти власні компоненти з нуля.

Ця комбінація ідеально підходить для проєктів, де важливі швидкість розробки, консистентність дизайну, адаптивність інтерфейсу до різних пристроїв та персоналізація через систему тем.

Таблиця 1.4 — Порівняння Tailwind CSS та Bootstrap

Характеристика	Tailwind CSS	Bootstrap
Підхід	Утилітарний	Компонентний
Гнучкість дизайну	Можливість створення унікальних інтерфейсів	Обмежена — всі сайти мають схожий вигляд
Швидкість кастомізації	Швидка, безпосередньо в HTML	Повільніша, часто вимагає перевизначення CSS
Розмір CSS-файлу	Менший (після очищення невикористаних класів)	Більший
Крива навчання	Середня — потрібно вивчити багато утилітарних класів	Низька — простіше почати з готових компонентів
Унікальність дизайну	Висока — легко створювати нестандартні інтерфейси	Низька — сайти часто виглядають як «ще один Bootstrap-сайт»
Продуктивність розробки	Висока для унікальних дизайнів	Висока для стандартних інтерфейсів
Інтеграція з JavaScript	Не включає JS-компоненти, потрібна окрема інтеграція	Включає готові JS-компоненти
Розмір бандлу	Менший завдяки PurgeCSS	Більший, включає невикористані стилі
Підтримка адаптивного дизайну	Вбудована, гнучка система	Вбудована, але менш гнучка
Відповідність сучасним тенденціям	Висока — популярний у нових проєктах	Середня — більш поширений у старших проєктах
Можливість розробки власних тем	Легко з DaisyUI, повний контроль	Можливо, але складніше реалізувати

1.10 OAuth 2.0

OAuth 2.0 — відкритий протокол авторизації, що надає третім сторонам обмежений доступ до ресурсів користувача без передачі облікових даних [3]. Він базується на делегуванні доступу, де користувач дозволяє застосунку отримувати дані з зовнішнього сервера.

OAuth 2.0 включає чотири ключові ролі:

— власник ресурсу — користувач, який надає дозвіл;

— клієнт — застосунок, який хоче отримати доступ;

— сервер авторизації — сервер, який видає токени доступу;

— сервер ресурсів — сервер, який зберігає захищені ресурси.

Процес авторизації складається з кількох важливих етапів. Спочатку клієнт запитує дозвіл у власника ресурсу, перенаправляючи користувача на сервер

авторизації, де відбувається аутентифікація. Після успішної аутентифікації власник надає дозвіл (авторизаційний грант), який клієнт обмінює на токен доступу. Цей токен являє собою обмежений в часі та правах доступу ключ, що дозволяє клієнту отримувати захищені ресурси без використання облікових даних користувача.

OAuth також підтримує інтеграцію з Google, що дозволяє користувачам входити в сторонні застосунки за допомогою своїх Google-акаунтів, забезпечуючи спрощений вхід, підвищену безпеку, двофакторну автентифікацію та централізоване управління доступом (рисунок 1.12).

— Переваги використання OAuth 2.0 включають:

— підвищену безпеку за рахунок уникнення передачі паролів між системами;

— обмежений за часом і правами доступ до ресурсів;

— можливість для користувача відкликати доступ у будь-який момент;

— масштабованість і гнучкість протоколу для різних сценаріїв використання;

— широку підтримку в індустрії, включаючи всі основні постачальники ідентифікації.

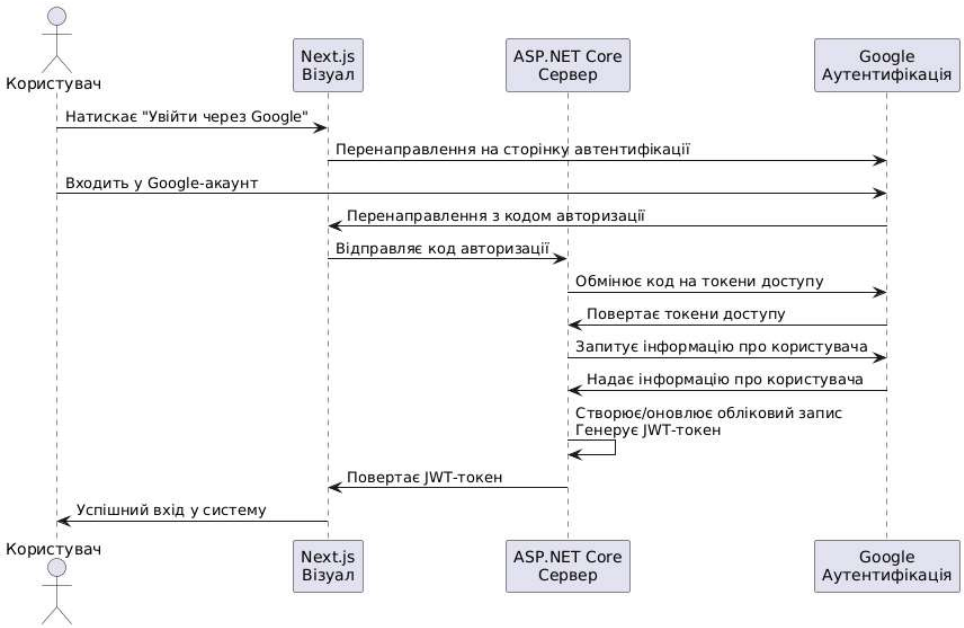


Рисунок 1.12 — Процес авторизації за допомогою OAuth2.0

2 ОБҐРУНТУВАННЯ ВИБОРУ ТЕХНОЛОГІЙ

2.1 Обгрунтування архітектурних підходів для веб-застосунків

Під час розробки навчально-інформаційної системи важливим кроком є вибір відповідної архітектури, яка забезпечить необхідну функціональність, масштабованість, підтримуваність та продуктивність [23]. Сучасні підходи до проєктування веб-застосунків включають:

— монолітна архітектура — традиційний підхід, де всі компоненти системи об'єднані в єдину кодову базу. Перевагами цього підходу є простота розробки та розгортання, проте він має обмеження в масштабуванні та гнучкості;

— мікросервісна архітектура — підхід, при якому застосунок розбивається на набір незалежних сервісів, кожен з яких відповідає за окрему бізнес-функцію. Перевагами є висока масштабованість та гнучкість, але такий підхід додає складності в розробці та розгортанні;

— цибулева архітектура (Onion Architecture) — архітектурний патерн, який організовує код навколо домену, підкреслюючи розділення відповідальності шляхом створення шарів, що залежать тільки від внутрішніх шарів.

Для реалізації навчально-інформаційної системи було обрано цибулеву архітектуру з наступних причин:

— незалежність від зовнішніх факторів — бізнес-логіка системи не залежить від конкретних технологій баз даних, фреймворків або зовнішніх сервісів, що забезпечує гнучкість при майбутніх змінах технологічного стеку;

— тестованість — розподіл відповідальності між шарами дозволяє писати модульне тестування для бізнес-логіки без залежності від інфраструктури;

— підтримуваність коду — архітектура забезпечує чітку структуру проєкту, що полегшує розуміння коду новими розробниками та спрощує внесення змін;

— принцип інверсії залежностей — внутрішні шари визначають інтерфейси, які реалізуються зовнішніми шарами, що забезпечує слабке зв'язування компонентів системи;

— оптимальна складність — на відміну від мікросервісної архітектури, цибулева архітектура не вимагає складної інфраструктури для розгортання, залишаючись при цьому достатньо гнучкою для освітньої системи;

— масштабованість бізнес-логіки — можливість легкого додавання нових функціональних можливостей без порушення існуючої структури системи.

2.2 Цибулева архітектура

Цибулева архітектура, запропонована Джеффрі Палермо, організовує код у концентричні шари, де зовнішні шари можуть залежати від внутрішніх, але не навпаки. Це дозволяє створити застосунок, стійкий до змін в інфраструктурі та технологіях, оскільки бізнес-логіка не залежить від зовнішніх деталей (рисунок 2.1).

Рисунок 2.1 — Цибулева архітектура

Основні шари цибулевої архітектури:

— доменне ядро (Domain Model) — центральний шар, який містить бізнес-сутності та їх відношення.

— доменні сервіси (Domain Services) — шар, що містить бізнес-правила та логіку, яка не відповідає конкретній сутності;

— прикладні сервіси (Application Services) — шар, який координує взаємодію між різними доменними сервісами та забезпечує інтерфейс для зовнішніх шарів;

— інфраструктура (Infrastructure) — зовнішній шар, що відповідає за взаємодію з базами даних, зовнішніми API та іншими системами;

— презентація (Presentation) — шар, що відповідає за взаємодію з користувачем (у нашому випадку це API для фронтенду та інтерфейс на Next.js).

Для реалізації цибулевої архітектури в ASP.NET Core проєкт був організований з використанням наступних компонентів:

Domain — проєкт, що містить доменні моделі, інтерфейси репозиторіїв та доменні сервіси:

— Entities — класи, що представляють основні бізнес-сутності (ClassEntity, SubjectEntity, AssignmentEntity, UserEntity);

— Interfaces — інтерфейси для репозиторіїв та сервісів (IClassRepository, ISubjectRepository тощо);

— Services — доменні сервіси, що реалізують бізнес-логіку.

Application — проєкт, що містить прикладні сервіси та об'єкти передачі даних (DTO):

— DTOs — класи для передачі даних між шарами (ClassDto, SubjectDto тощо);

— Interfaces — інтерфейси прикладних сервісів;

— Services — реалізації прикладних сервісів;

— Validators — класи валідації вхідних даних.

Infrastructure — проєкт, що містить реалізацію інфраструктурних компонентів:

— Data — реалізація доступу до даних за допомогою Entity Framework Core;

— Repositories — реалізація репозиторіїв;

— ExternalServices — інтеграція з зовнішніми сервісами (Google OAuth, електронна пошта тощо);

— Authentication — компоненти для автентифікації та авторизації.

WebApi — проєкт, що містить контролери API та конфігурацію застосунку:

— Controllers — контролери REST API;

— Filters — фільтри ASP.NET Core для обробки запитів;

— Middleware — програмні компоненти для обробки HTTP-запитів;

— Configuration — налаштування застосунку та DI-контейнера.

2.3 Порівняння альтернатив для серверної частини

При виборі технологічної платформи для серверної частини навчально-інформаційної системи було проведено ґрунтовний аналіз сучасних фреймворків для розробки веб-застосунків. Основними кандидатами для вибору були ASP.NET Core, Node.js (Express/Nest.js), Spring Boot (Java), Django/Flask (Python) та Laravel (PHP). Порівняння цих технологій було здійснено за рядом ключових критеріїв, важливих для реалізації проєкту (таблиця 2.1).

Таблиця 2.1 — Порівняльна таблиця серверної частини

Критерій	ASP.NET Core	Node.js	Spring Boot (Java)	Django/Flask (Python)	Laravel (PHP)
Продуктивність	Висока	Висока для асинхронних операцій	Висока	Середня	Середня
Масштабованість	Висока	Висока	Висока	Середня	Середня
Типізація	Статична (C#)	Динамічна (JS) / Статична (TS)	Статична (Java)	Динамічна (Python)	Динамічна (PHP)
ORM підтримка	Entity Framework Core	Sequelize, TypeORM	Hibernate, JPA	Django ORM, SQLAlchemy	Eloquent
Безпека	Висока	Середня (залежить від реалізації)	Висока	Висока	Середня
Документація	Відмінна	Добра	Відмінна	Добра	Добра

Продовження таблиці 2.1

Крива навчання	Середня/Висока	Низька/Середня	Висока	Низька/ Середня	Низька/ Середня
Підтримка спільноти	Активна	Дуже активна	Активна	Активна	Активна
Асинхронність	Відмінна	Відмінна	Добра	Обмежена	Обмежена
Контейнеризація	Відмінна	Добра	Відмінна	Добра	Середня

ASP.NET Core був обраний як оптимальна технологія для серверної частини проєкту з наступних причин:

— Висока продуктивність та масштабованість

— Статична типізація та потужна екосистема .NET

— Entity Framework Core та інтеграція з різними СУБД

— Вбудована підтримка JWT-автентифікації та OAuth

— Відмінна документація та активна спільнота

— Відповідність принципам цибулевої архітектури

ASP.NET Core демонструє непогані показники продуктивності серед веб-фреймворків, що критично важливо для навчально-інформаційної системи, яка може мати значне навантаження під час пікового використання. Використання мови C# із статичною типізацією забезпечує раннє виявлення помилок, покращує підтримку з боку IDE та полегшує рефакторинг коду.

Екосистема .NET надає багатий набір бібліотек та інструментів для розробки, тестування та розгортання застосунків.

Наявність потужного ORM-фреймворку, який підтримує як SQLite (для розробки), так і PostgreSQL (для промислового середовища), значно спрощує роботу з даними та забезпечує гнучкість при виборі СУБД. ASP.NET Core має розвинуті механізми для реалізації автентифікації та авторизації, що є критично важливим для навчально-інформаційної системи, де потрібен контроль доступу до різних ресурсів.

Microsoft забезпечує детальну документацію для ASP.NET Core, а активна спільнота розробників створює численні бібліотеки, статті та навчальні матеріали,

що спрощує процес розробки та вирішення проблем. ASP.NET Core добре підходить для реалізації цибулевої архітектури завдяки підтримці залежностей інверсій, розділенню на проєкти та модульності.

2.4 Технології роботи з даними

Для роботи з даними були розглянуті різні технології, доступні в екосистемі .NET: EF Core, Dapper, NHibernate, ADO.NET та пряма робота з SQL (таблиця 2.2).

Таблиця 2.2 — Порівняльна таблиця для роботи з даними

Критерій	Entity Framework Core	Dapper	NHibernate	ADO.NET	Пряма робота з SQL
Рівень абстракції	Високий	Низький/ Середній	Високий	Низький	Дуже низький
Продуктивність	Середня/ Висока	Дуже висока	Середня	Висока	Дуже висока
Функціональність	Дуже висока	Базова	Висока	Базова	Базова
Підтримка Code First	Так	Ні	Так	Ні	Ні
Автоматична міграція	Так	Ні	Обмежена	Ні	Ні
Відстеження змін	Так	Ні	Так	Ні	Ні
Lazy-завантаження	Так	Ні	Так	Ні	Ні
Інтеграція з LINQ	Повна	Обмежена	Часткова	Ні	Ні
Крива навчання	Середня	Низька	Висока	Низька	Низька
Підтримка спільноти	Активна	Активна	Середня	Середня	-

Entity Framework Core був обраний як оптимальна технологія для роботи з даними з наступних причин:

— підтримка Code First підходу;

— система міграцій;

— повна інтеграція з LINQ;

— підтримка відстеження змін та лінивого завантаження;

— сумісність з різними СУБД;

— оптимізація продуктивності;

— інтеграція з ASP.NET Core Identity;

— розширена підтримка навігаційних властивостей;

— можливості транзакційної обробки;

— масштабованість та надійність.

Entity Framework Core дозволяє визначати модель даних у коді та автоматично генерувати відповідну схему бази даних, що ідеально відповідає принципам доменно-орієнтованого проєктування та цибулевої архітектури [8]. Цей підхід значно спрощує розробку, дозволяючи зосередитися на бізнес-логіці, а не на деталях реалізації бази даних .

Автоматична генерація міграцій та можливість їх застосування забезпечує контрольовані зміни структури бази даних, що є важливим для еволюціонуючої системи. Міграції можна версіонувати, тестувати та розгортати як частину процесу безперервної інтеграції, забезпечуючи надійність оновлень схеми бази даних.

Можливість написання запитів за допомогою LINQ забезпечує більшу надійність коду та краще розуміння запитів до БД. Розробники можуть використовувати звичну синтаксичну конструкцію C#, яка перетворюється на ефективні SQL-запити. Це не лише підвищує продуктивність розробки, але й зменшує кількість помилок, пов'язаних з SQL-ін'єкціями.

Функції відстеження змін та лінивого завантаження значно спрощують роботу з об'єктами доменної моделі та підвищують ефективність запитів. Відстеження змін автоматично визначає, які об'єкти були змінені, додані або видалені, що спрощує операції збереження. Ліниве завантаження дозволяє

завантажувати пов'язані дані лише тоді, коли вони потрібні, оптимізуючи використання пам'яті та продуктивність.

Entity Framework Core має провайдери для різних систем управління базами даних, включаючи SQLite та PostgreSQL, які використовуються в проєкті. Це забезпечує гнучкість у виборі СУБД без зміни коду доступу до даних, що особливо важливо для різних середовищ розробки, тестування та промислової експлуатації.

Фреймворк пропонує численні можливості для оптимізації продуктивності, включаючи компіляцію запитів, групове завантаження даних та кешування результатів. Ці функції дозволяють створювати ефективні запити, що важливо для навчальної системи з потенційно великими обсягами даних та періодами пікового навантаження.

Безшовна інтеграція з ASP.NET Core Identity спрощує реалізацію системи автентифікації та авторизації, що є критичним компонентом навчально-інформаційної системи. Entity Framework Core дозволяє легко розширювати стандартні моделі користувачів та ролей для задоволення специфічних потреб проєкту.

Розширена підтримка навігаційних властивостей забезпечує зручне моделювання складних зв'язків між сутностями, що важливо для представлення ієрархічних структур в освітньому контексті, таких як курси, уроки, завдання та матеріали.

Entity Framework Core забезпечує потужні можливості для транзакційної обробки даних, гарантуючи цілісність даних навіть при складних операціях. Це особливо важливо для функцій, які вимагають атомарних операцій, таких як створення нового класу з набором уроків або оцінювання студентів.

Як частина екосистеми .NET, Entity Framework Core демонструє високу масштабованість та надійність, що підтверджується його використанням у великих корпоративних системах. Це дозволяє бути впевненим у тому, що технологія задовольнить потреби зростаючої навчальної платформи в довгостроковій перспективі.

2.5 Порівняльний аналіз підходів до автентифікації

При розробці системи було проведено аналіз різних підходів до автентифікації: сесійна автентифікація, токен-базована автентифікація (включаючи JWT), OAuth та відкриті ключі (таблиця 2.3).

Таблиця 2.3 — Порівняння технологій автентифікації

Критерій	Сесійна автентифікація	JWT (JSON Web Tokens)	OAuth 2.0	Автентифікація через API-ключі
Принцип роботи	Зберігання сесій на сервері	Токени з інформацією	Делегування 3-ю стороною	Секретні ключі у запиті
Масштабованість	Обмежена	Висока	Висока	Висока
Безпека	Висока	Висока	Дуже висока	Середня
Складність реалізації	Середня	Низька/ Середня	Висока	Низька
Підтримка SSO	Обмежена	Можлива	Нативна	Обмежена
Стійкість до CSRF	Потребує додаткового захисту	Стійка	Стійка	Залежить від реалізації
Контроль сесії з боку сервера	Повний	Обмежений	Частковий	Обмежений
Можливість відкликання	Легко	Складно	Підтримується	Можливо через скасування ключа
Навантаження на сервер	Високе	Низьке	Середнє	Дуже низьке
Підтримка в ASP.NET Core	Вбудована	Вбудована	Вбудована	Потребує кастомної реалізації

У результаті аналізу, для основної автентифікації користувачів було обрано технологію JSON Web Tokens (JWT) [10].

JWT реалізована у вигляді комплексної системи з такими типами токенів:

— Access Token — короткостроковий токен (15–30 хв), що містить дані користувача та його права, використовується для авторизації API-запитів і зберігається в пам’яті клієнта;

— Refresh Token — довгостроковий токен (7–30 днів), що дозволяє отримувати нові access токени без повторного входу, зберігається в HTTP-only cookies або в localStorage;

Access та refresh токени в системі автентифікації на основі JWT діють спільно: access token використовується для тимчасового доступу до ресурсів, а refresh token — для оновлення access token без повторної авторизації. Такий підхід забезпечує як безпеку, так і зручність безперервної роботи користувача в системі (рисунок 2.2).

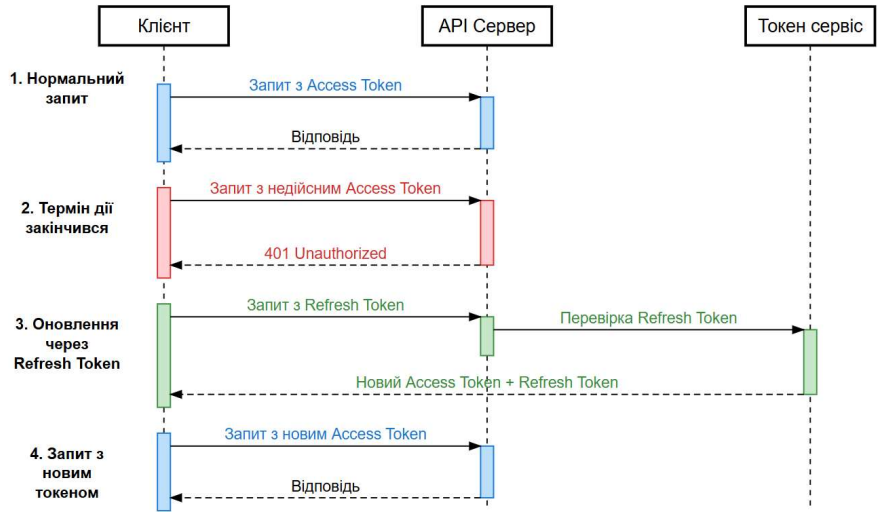


Рисунок 2.2 — Взаємодія між refresh- та access-токенів

Основні переваги використання JWT у навчально-інформаційній системі:

— зниження навантаження на базу даних за рахунок відсутності запитів для перевірки сесії при кожному зверненні до API;

— підтримка різних клієнтів (веб-застосунок, мобільні додатки, сторонні системи) завдяки стандартизованому формату токенів;

— безпечне розділення механізмів автентифікації та авторизації через використання різних типів токенів з відповідними термінами дії;

— гнучкий контроль над доступом завдяки можливості зберігання ролей та прав у JWT claims;

— можливість швидкого відкликання доступу через механізм короткострокових access-токенів та блекліст для критичних випадків.

3 РЕАЛІЗАЦІЯ СЕРВЕРНОЇ ЧАСТИТИ

3.1 Моделювання сутностей у БД

Для реалізації навчального вебсервісу було спроєктовано набір сутностей, що відображають основні логічні одиниці системи: навчальні класи, предмети, повідомлення, завдання, матеріали, результати виконаних завдань тощо (таблиця 3.1).

Таблиця 3.1 — Таблиця сутностей

Назва сутності	Атрибути
Навчальний клас	Id, Назва, Опис, Власник, Id власника, Дата створення, URL, Предмети
Навчальний предмет	Id, Назва, Опис, Власник, Id власника, Дата створення, URL, Номер аудиторії, Id класу, Клас, Користувачі
Базовий допис	Id, Назва, Тіло, Дата створення, Id створюючого користувача
Повідомлення	Id, Назва, Тіло, Дата створення, Id створюючого користувача, Важливість
Завдання	Id, Назва, Тіло, Дата створення, Id створюючого користувача, Дата виконання, Максимальна оцінка
Результат завдання	Id, Id завдання, Id користувача, Оцінка, Дата виконання, Дата перевірки, Користувач, Ідентифікатор перевіряючого, Статус завантажених матеріалів
Матеріал	Id, Назва матеріалу, Дата створення, Дата зміни, URL, Захист, Id власника
Зв’язок між предметом і користувачем	Id, Id користувача, Id предмета, Користувач, Предмет
Зв’язок між контейнером і дописом	Id, Id контейнера, Id допису, Контейнер, Допис
Захист матеріалу	Тип захисту матеріалу (Приватний, Публічний)

Ці сутності реалізовані у вигляді C#-класів з використанням підходу Code First, який є частиною ORM-технології Entity Framework Core.

Code First — це підхід, при якому спочатку розробляється модель даних у вигляді класів, після чого за допомогою міграцій база даних створюється або оновлюється автоматично відповідно до цієї моделі. Це дає змогу уникнути ручного створення схеми бази даних та підтримувати її в актуальному стані через зміну коду.

Для конфігурації полів (наприклад, позначення ключів, зв’язків, обов’язкових полів) використовуються атрибути з простору імен System.ComponentModel.DataAnnotations, що дозволяє чітко визначити структуру та обмеження для кожної сутності.

Клас EduMaterialInfo представляє сутність "Матеріали" у системі та включає властивості для зберігання та управління навчальними матеріалами. Властивість Id позначена атрибутом KeyAttribute як первинний ключ таблиці типу Guid. Властивість OwnerId з атрибутом RequiredAttribute є обов'язковим полем, що зберігає ідентифікатор власника. Навігаційна властивість Owner типу ApplicationUser пов'язана з OwnerId за допомогою атрибута ForeignKeyAttribute. Властивість MaterialName з атрибутом RequiredAttribute зберігає назву навчального матеріалу. Властивості Created та Modified типу DateTime відстежують час створення та останньої модифікації матеріалу. Властивість Url зберігає посилання на фізичне розташування матеріалу. Властивість Protection типу MaterialProtection з атрибутом RequiredAttribute визначає рівень захисту матеріалу та контролює доступ до нього.

У об'єктно-орієнтованому програмуванні абстрактні класи часто використовуються для створення загальних шаблонів, які можуть бути розширені іншими класами. Абстрактний клас сам по собі не може бути інстанційований, але він надає загальну структуру, яку можуть успадковувати інші класи. У даному випадку, абстрактний клас EduContainer використовується як базовий клас для сутностей EduClass та EduSubject.

Абстрактний клас EduContainer містить базові властивості для освітніх контейнерів: ідентифікатор Id з атрибутом KeyAttribute як первинний ключ, обов'язкове поле Name з атрибутом RequiredAttribute для зберігання назви контейнера, поле Description для опису, обов'язковий ідентифікатор власника OwnerId з атрибутом RequiredAttribute, навігаційну властивість Owner типу ApplicationUser пов'язану з OwnerId за допомогою атрибута ForeignKeyAttribute, поле Created для відстеження дати створення та поле Url для зберігання посилання.

Клас EduClass успадковує всі властивості від абстрактного класу EduContainer та додає колекцію Subjects типу ICollection<EduSubject>, що дозволяє зберігати перелік предметів, які належать до конкретного класу.

Клас EduSubject також успадковує властивості від EduContainer та розширює їх додатковими полями: необов'язковим полем RoomNumber для вказання номера кімнати, де проводяться заняття; необов'язковим полем ClassId, яке може містити ідентифікатор класу; навігаційною властивістю Class типу EduClass з атрибутом ForeignKeyAttribute, пов'язаною з ClassId; колекцією Users типу ICollection<ApplicationUser> для зберігання користувачів, пов'язаних з предметом.

Table-Per-Hierarchy (TPH) — це патерн відображення ієрархії наслідування в реляційній базі даних, за якого вся ієрархія класів (базовий клас та всі похідні класи) зберігається в одній таблиці. Для визначення конкретного типу сутності використовується спеціальна колонка-дискримінатор.

Патерн Table-Per-Hierarchy (TPH) пропонує просте рішення для відображення об'єктної ієрархії в реляційній базі даних. Суть цього підходу полягає у створенні єдиної універсальної таблиці, яка вміщує всі сутності з ієрархії наслідування. Така таблиця об'єднує всі атрибути — як спільні, успадковані від базового класу, так і специфічні для кожного з похідних типів (таблиця 3.1).

Метод HasDiscriminator використовується для реалізації патерну наслідування в Entity Framework Core шляхом конфігурації у перевизначеному методі OnModelCreating класу контексту бази даних. У даному випадку відбувається налаштування сутності EduContainer через виклик методу Entity, де додається дискримінатор за допомогою методу HasDiscriminator з параметром типу

string та назвою колонки EduType, який вказує, що розрізнення типів сутностей буде здійснюватися за допомогою колонки-дискримінатора відповідного типу. Далі відбувається встановлення конкретних значень для похідних класів: метод HasValue вказує, що для сутностей типу EduClass та EduSubject у колонці-дискримінаторі будуть зберігатися значення Class та Subject відповідно. Такий підхід забезпечує ефективне зберігання різних типів сутностей у одній таблиці та коректне відновлення ієрархії об'єктів при зчитуванні даних з бази, що дозволяє підтримувати цілісність об'єктної моделі та оптимізувати структуру бази даних.

Структура навчально-інформаційної системи показано на рисунку 3.1.

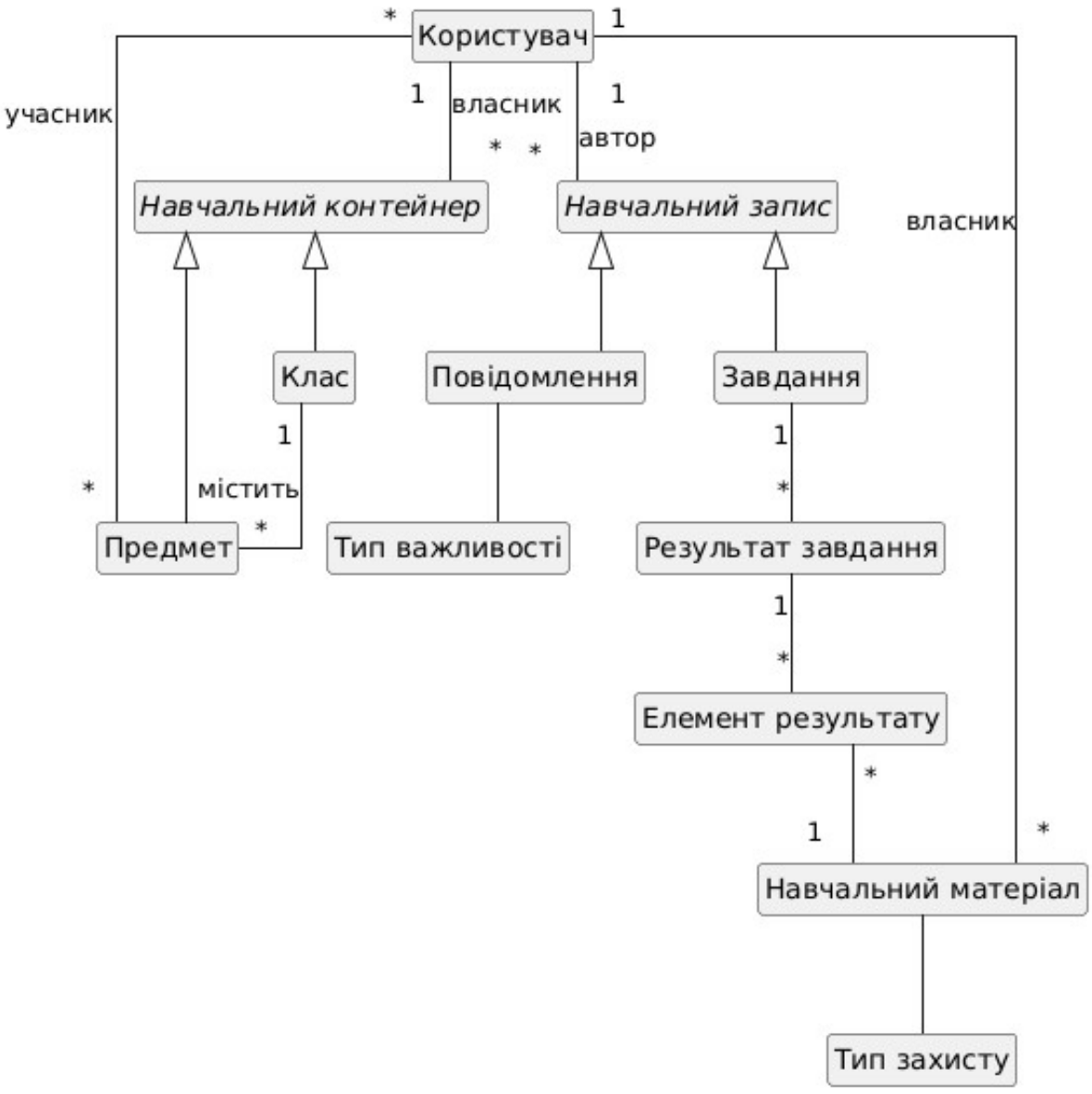


Рисунок 3.1 — Структура навчально-інформаційної системи

3.2 Використання ASP.NET Identity для управління користувачами

ASP.NET Identity — потужний фреймворк для управління користувачами, що надає широкі можливості для автентифікації, авторизації та керування обліковими записами [26]. Основна перевага ASP.NET Identity полягає в тому, що він надає розширювану та гнучку архітектуру, яка дозволяє адаптувати систему автентифікації під конкретні потреби проєкту.

Стандартні типи даних ASP.NET Identity було розширено для задоволення специфічних потреб системи. Зокрема, для ідентифікації користувачів було використано тип даних Guid замість стандартного числового типу.

Клас ApplicationDbContext успадковує функціональність IdentityDbContext з використанням узагальнення, де вказуються типи для користувача (ApplicationUser), ролі (ApplicationRole) та ідентифікатора (Guid), що дозволяє замінити стандартний числовий тип ідентифікатора на глобально унікальний ідентифікатор для більшої надійності та уникнення колізій при розподіленому розгортанні системи, а також приймає параметр options типу DbContextOptions для конфігурації підключення до бази даних.

Використання Guid (Globally Unique Identifier) як типу даних для ідентифікаторів користувачів надає кілька істотних переваг:

— глобальна унікальність — Guid забезпечує практично гарантовану унікальність у будь-якому контексті, що особливо важливо при можливому масштабуванні системи чи об'єднанні баз даних;

— безпека — на відміну від послідовних ідентифікаторів, Guid не є передбачуваним, що підвищує безпеку системи, запобігаючи потенційним атакам, пов'язаним із підбором ідентифікаторів;

— розподілена генерація — створення Guid не вимагає централізованого координування, що спрощує розподілену архітектуру системи;

— підтримка універсальних інтерфейсів — використання одного типу ідентифікаторів (Guid) для всіх сутностей системи спрощує проєктування загальних інтерфейсів та сервісів.

ASP.NET Identity створює набір взаємопов'язаних таблиць у базі даних для зберігання інформації про користувачів, ролі, права доступу та інші аспекти ідентифікації (таблиця 3.2).

Таблиця 3.2 — Таблиця сутностей для підтримки ASP.NET Identity

Назва таблиці	Призначення
AspNetUsers	Зберігає основну інформацію про користувачів: ідентифікатор, ім'я користувача, електронну пошту, хеш пароля, статус підтвердження email тощо.
AspNetRoles	Містить ролі, які можуть бути призначені користувачам (наприклад, «Студент», «Викладач», «Адміністратор»).
AspNetUserRoles	Таблиця зв'язку багато-до-багатьох між користувачами та ролями.
AspNetUserClaims	Зберігає додаткові твердження (claims) для користувачів, які використовуються для авторизації та зберігання додаткової інформації.
AspNetUserLogins	Інформація про зовнішні логіни (Google, Facebook тощо), пов'язані з користувачами.
AspNetUserTokens	Зберігає токени для користувачів (наприклад, токени підтвердження email, скидання пароля, а в нашому випадку також JWT токени).

Кожна з цих таблиць виконує свою роль у загальній системі автентифікації та авторизації:

— автентифікація: перевірка ідентичності користувача (таблиці AspNetUsers, AspNetUserLogins, AspNetUserTokens);

— авторизація: визначення прав доступу користувача (таблиці AspNetRoles, AspNetUserRoles, AspNetUserClaims);

— управління обліковими записами: операції з відновлення паролю, підтвердження електронної пошти тощо (таблиця AspNetUserTokens).

3.3 Контекст даних

Центральним компонентом для роботи з даними в системі є клас ApplicationDbContext, який успадковується від IdentityDbContext, що надає доступ до сутностей ASP.NET Identity.

Цей клас виконує кілька ключових функцій:

— інтеграція ASP.NET Identity — успадкування від IdentityDbContext забезпечує доступ до всіх таблиць ідентифікації;

— визначення доменних сутностей — через властивості DbSet<T> визначаються сутності предметної області, які будуть відображені у таблицях бази даних;

— конфігурація наслідування — у методі OnModelCreating налаштовується патерн Table-Per-Hierarchy (TPH) для сутностей EduContainer та EduPost, що забезпечує ефективне зберігання ієрархій класів у базі даних;

— конфігурація перетворень типів — автоматична конфігурація перетворення перелічуваних типів (enum) у цілочисельні значення для зберігання в базі даних.

Така структура контексту даних дозволяє ефективно об'єднати функціональність ASP.NET Identity з доменною моделлю навчально-інформаційної системи.

3.4 Реалізація JWT автентифікації

Для забезпечення безпечної та ефективної автентифікації в системі було реалізовано механізм JWT (JSON Web Tokens). Основна логіка генерації токенів інкапсульована в класі JwtTokenGenerator.

Клас JwtTokenGenerator містить приватне поле _credentials типу SigningCredentials, яке зберігає криптографічні дані для підпису токенів, забезпечуючи їх захист від підробки. Приватний метод GenerateToken приймає колекцію тверджень claims, що містять інформацію про користувача, параметр lifetime для встановлення терміну дії токена та параметр tokenType для визначення типу токена, після чого повертає згенерований JWT-токен у вигляді рядка. Публічний метод GenerateAccessToken приймає об'єкт користувача типу ApplicationUser та генерує короткостроковий токен доступу з необхідними даними користувача. Публічний метод GenerateRefreshToken приймає ідентифікатор користувача типу Guid та створює довгостроковий токен оновлення для можливості отримання нового токена доступу без повторної автентифікації.

Даний клас відповідає за створення таких типів JWT токенів, кожен з яких має своє призначення:

— Access Token — короткострокові токени (термін дії 60 хвилин), містять інформацію про користувача, ролі та використовуються для доступу до захищених ресурсів API (реалізовується за допомогою методу GenerateAccessToken);

— Refresh Token — довгострокові токени (термін дії 1 день), які використовуються для отримання нових access токенів без необхідності повторної автентифікації (генерується за допомогою методу GenerateRefreshToken).

Реалізація JwtTokenGenerator використовує криптографічний алгоритм HMAC SHA-256 для підписання токенів, що забезпечує їх захист від підробки. При ініціалізації класу створюються криптографічні креденціали на основі секретного ключа, які потім використовуються для генерації підпису токена [10].

Для генерації токенів доступу клас використовує дані користувача, включаючи його унікальний ідентифікатор, електронну пошту та роль, що дозволяє системі здійснювати авторизацію на основі цих параметрів. Refresh токени містять мінімально необхідну інформацію — лише ідентифікатор користувача, що підвищує безпеку системи.

Архітектура класу також передбачає можливість розширення для підтримки додаткових типів токенів у майбутньому, про що свідчить закоментований метод для генерації сесійних токенів у вихідному коді.

Кожен JWT токен складається з трьох основних частин, розділених крапками:

— заголовок, що містить тип токена та алгоритм підпису;

— корисне навантаження, що містить інформацію про користувача, час кінця токена;

— підпис, що створюється шляхом шифрування заголовка та корисного навантаження з використанням секретного ключа.

Кожна секція шифрується за допомогою base64 (рисунок 3.2).

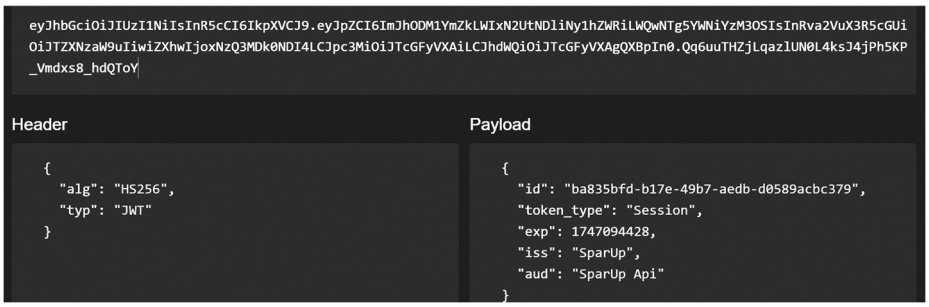


Рисунок 3.2 — Декодування JWT-токена

3.5 Реалізація сервісу електронних повідомлень

3.5.1 Налаштування пароля для додатків у Gmail

Для надсилання системних повідомлень рекомендовано створити окремий обліковий запис електронної пошти. Це забезпечує розділення системних та особистих комунікацій, полегшує моніторинг та підтримку, а також підвищує рівень безпеки системи (рисунок 3.3).

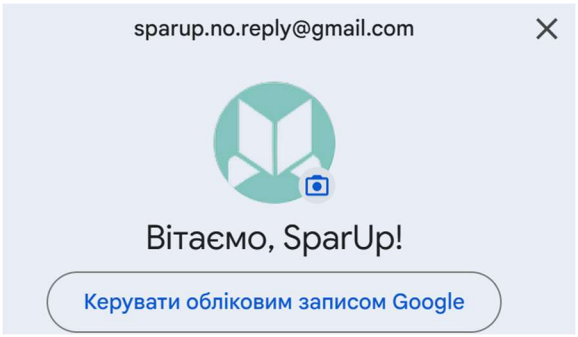


Рисунок 3.3 — Окремий обліковий запис для сервісу

Для використання облікового запису Gmail як відправника системних повідомлень необхідно налаштувати спеціальний пароль для додатків, оскільки стандартна автентифікація часто блокується для сторонніх сервісів [2].

Процес створення такого пароля починається з увімкнення двофакторної автентифікації у налаштуваннях безпеки облікового запису Google. Після активації двоетапної перевірки та її підтвердження через SMS або інший спосіб, користувач отримує доступ до розділу «Паролі додатків». У цьому розділі необхідно обрати тип та назву додатку для сервісу, після чого система згенерує 16-символьний пароль. Отриманий пароль слід додати до захищеного сховища конфігурацій

системи та забезпечити доступ до нього через механізм впровадження залежностей (рисунок 3.4).

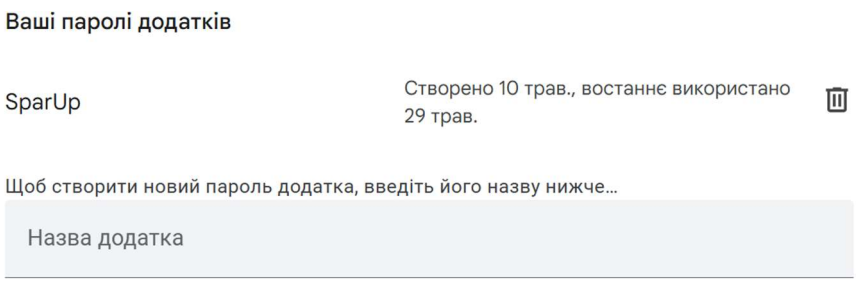


Рисунок 3.4 — Паролі додатків сервісу SparUp

3.5.2 Програмна реалізація сервісу електронних повідомлень

Для забезпечення надсилання електронних повідомлень розроблено два взаємодоповнюючі класи: SmtpConfigurationOptions для зберігання параметрів підключення та EmailService для безпосередньої роботи з електронною поштою.

Клас SmtpConfigurationOptions містить набір властивостей для зберігання конфігураційних параметрів підключення до SMTP-сервера: властивість SmtpServer типу string для зберігання адреси або доменного імені поштового сервера, властивість SmtpPort типу int для зберігання номера порту для підключення до SMTP-сервера, властивість Email типу string для зберігання електронної адреси, від імені якої надсилатимуться повідомлення, та властивість Password типу string для зберігання пароля, необхідного для автентифікації на поштовому сервері.

Клас EmailService приймає в конструкторі параметр optionProvider типу IOptions, що має параметр типу SmtpConfigurationOptions, який використовується для отримання конфігураційних параметрів з системи ін'єкції залежностей ASP.NET Core. Клас містить приватну властивість Options, яка надає доступ до значення конфігурації через optionProvider.Value. Публічний метод SendEmailAsync приймає параметр eMessage типу EmailMessage, що містить інформацію про лист (отримувачів, тему, тіло повідомлення), та асинхронно

відправляє електронне повідомлення, використовуючи налаштування з конфігураційних параметрів.

Клас SmtpConfigurationOptions призначений для зберігання конфігураційних параметрів підключення до SMTP-сервера, що включають адресу сервера, порт, електронну адресу відправника та пароль для автентифікації.

Основний функціонал надсилання повідомлень реалізовано в класі EmailService, який отримує конфігурацію через механізм впровадження залежностей, формує повідомлення відповідно до MIME-стандарту, встановлює з'єднання з SMTP-сервером з використанням захищеного протоколу, автентифікується на сервері за допомогою облікових даних, відправляє повідомлення та коректно завершує з'єднання. Для роботи з SMTP використовується бібліотека MailKit, що забезпечує надійне та безпечне підключення до поштових серверів з підтримкою сучасних протоколів шифрування.

Розроблений сервіс електронних повідомлень використовується у кількох ключових компонентах системи: підтвердження реєстрації нових користувачів, відновлення пароля, інформування про важливі події в системі, надсилання повідомлень про нові завдання та оцінки. На рисунку 3.5 представлено приклад надісланого системою електронного повідомлення для підтвердження реєстрації користувача. Повідомлення містить унікальне посилання для активації облікового запису, а також базову інформацію про систему. Використання спеціалізованого сервісу електронних повідомлень забезпечує надійну комунікацію з користувачами та підвищує загальну функціональність навчальної платформи.

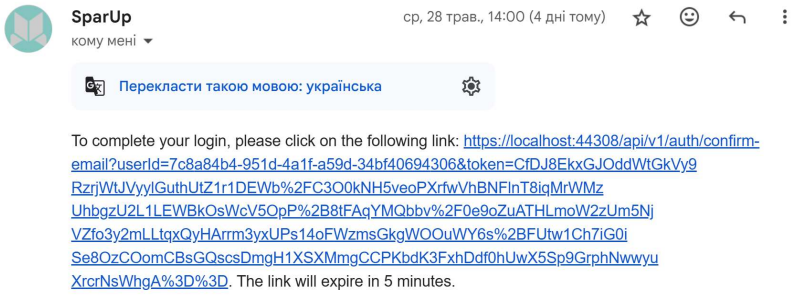


Рисунок 3.5 — Повідомлення від сервісу для підвердження реєстрації користувача

3.6 Інтеграція системи автентифікації з Google OAuth 2.0

3.6.1 Налаштування проекту в Google Cloud Platform

Для розширення можливостей автентифікації користувачів та підвищення зручності використання системи вирішено реалізувати інтеграцію з протоколом OAuth 2.0 компанії Google. Це дозволяє користувачам реєструватися та входити в систему за допомогою своїх облікових записів Google, що є особливо актуальним для освітнього середовища, де Google Workspace часто використовується як основний інструмент.

Процес інтеграції розпочинається з налаштування проєкту в Google Cloud Platform. Створено новий проект у консолі Google Cloud, після чого відбувається налаштування OAuth-сервісу (рисунок 3.6). У розділі «API & Services», «Credentials» створюються нові OAuth 2.0 Client ID для веб-додатку. Також додаються дозволені URI перенаправлення (redirect URIs), включаючи шлях /oauth/google/callback для обробки відповіді від Google після автентифікації (рисунок 3.6).

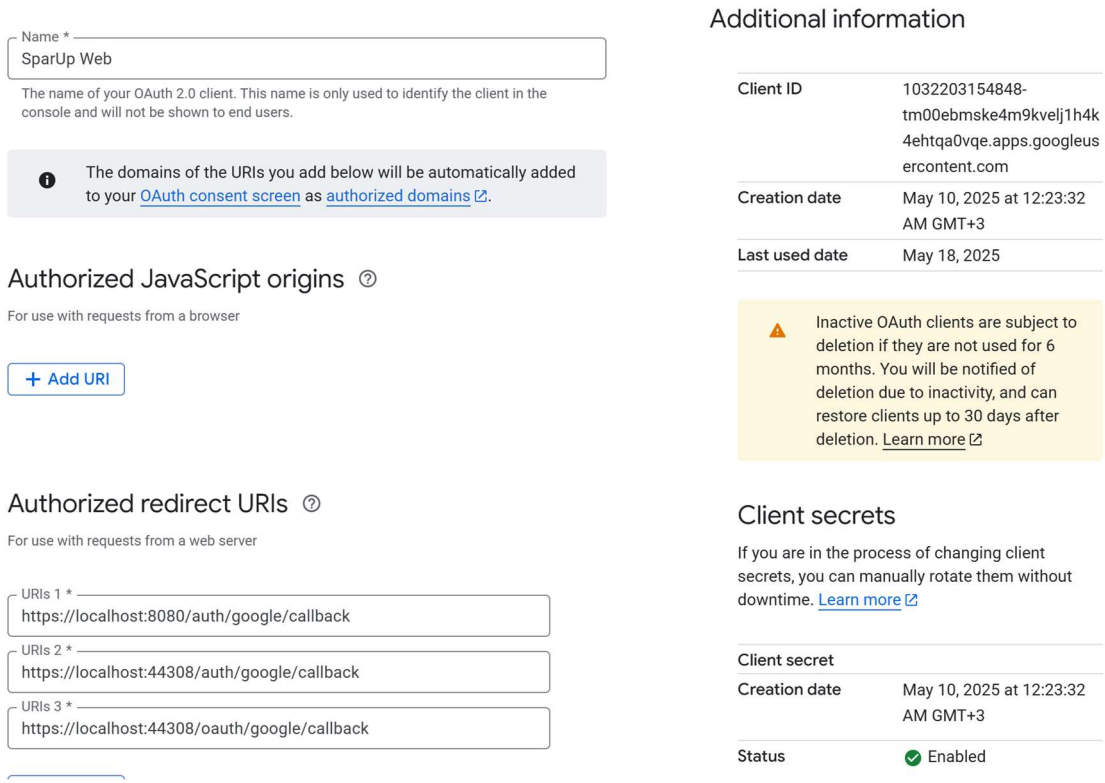


Рисунок 3.6 — Налаштування проєкту для входу через Google OAuth 2.0

Сконфігуровано екран згоди користувача (OAuth consent screen) з визначенням назви додатку, логотипу та запитуваних областей доступу (рисунок 3.7). У результаті отримано Client ID та Client Secret — критично важливі дані для подальшої інтеграції з API Google (рисунок 3.6).

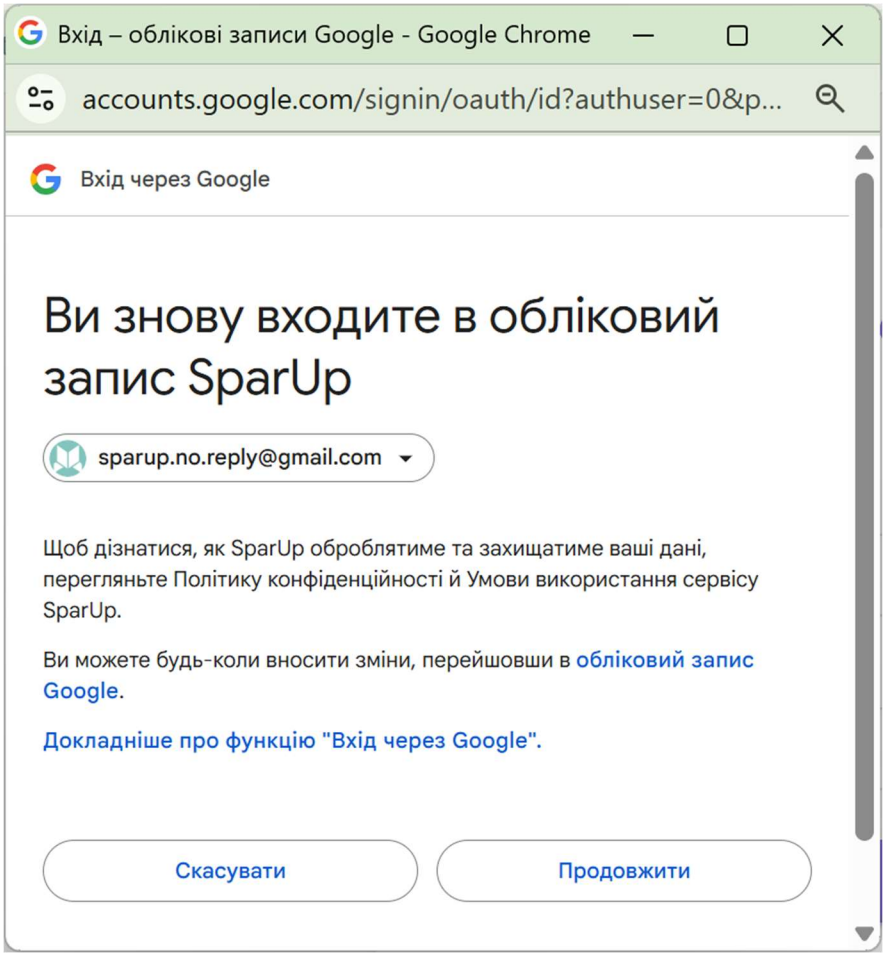


Рисунок 3.7 — Екран згоди користувача

3.6.2 Конфігурація та реалізація автентифікації в ASP.NET Core

Для інтеграції з Google OAuth у проекті використовується пакет Microsoft.AspNetCore.Authentication.Google, який надає готові компоненти для роботи з протоколом OAuth 2.0.

Cookies містить в собі стан автентифікації користувача, а для викликів автентифікації використовується провайдер Google. Також встановлено шлях зворотного виклику та включено збереження токенів для подальшого використання.

Налаштування автентифікації відбувається через метод AddAuthentication, де встановлюються параметри options, які визначають DefaultScheme як Cookies для використання cookie-автентифікації за замовчуванням та DefaultChallengeScheme як Google для переспрямування на Google при необхідності автентифікації. Метод AddCookie додає обробник для схеми Cookies, а метод AddGoogle налаштовує провайдер Google з відповідними параметрами options: ClientId для ідентифікації додатку в Google, ClientSecret для безпечної автентифікації додатку, CallbackPath для встановлення шляху зворотного виклику після автентифікації та параметр SaveTokens для збереження токенів доступу та оновлення.

Для обробки процесу автентифікації через Google було розроблено спеціалізований контролер OAuthController, який реалізує логіку взаємодії з API Google за протоколом OAuth 2.0.

Контролер OAuthController має атрибут ApiController для підтримки API-функціональності та атрибут Route для визначення базового шляху "oauth/google". Конструктор контролера приймає параметри: userManager типу UserManager, що має параметр типу ApplicationUser для управління користувачами, tokenGenerator типу JwtTokenGenerator для генерації JWT-токенів та emailService типу EmailService для надсилання електронних повідомлень. Метод UseGoogleAuth з атрибутом HttpGet ініціює процес автентифікації через Google та приймає параметр refresh для визначення необхідності оновлення токенів. Метод Callback з атрибутом HttpGet обробляє зворотний виклик від Google після успішної автентифікації користувача.

Процес автентифікації через Google розпочинається з перенаправлення користувача на сторінку входу Google через кінцеву точку sign-in. Ця кінцева точка використовує метод ChallengeAsync для показу діалогу згоди та для отримання раніше наданих дозволів. У параметрах автентифікації зберігається URL для подальшого перенаправлення, який використовується після успішної автентифікації.

Після автентифікації Google перенаправляється спочатку на власну кінцеву точку, яка вказана в параметрах налаштування, а звідти — на кінцеву точку

callback, де система підтвердює успішність процесу, отримує електронну пошту та ім'я користувача з токена, і шукає відповідний обліковий запис у базі даних. Якщо користувача не знайдено, створюється новий обліковий запис з генерованим паролем та надсилається електронний лист з інформацією про вхід у систему.

3.7 Налаштування та підключення до БД

Для забезпечення надійного збереження даних у системі використано технологію Entity Framework Core, яка надає зручний об'єктно-реляційний мапінг (ORM) для роботи з різними системами управління базами даних. Одним з ключових аспектів розробки є забезпечення гнучкості при переході між середовищами розробки та виробництва, для чого реалізовано підтримку різних СУБД.

Для середовища розробки та тестування обрано SQLite — легку вбудовану реляційну БД, яка не потребує окремого сервера та дозволяє швидко розгорнути середовище для тестування. Підключення SQLite здійснюється через пакет Microsoft.EntityFrameworkCore.Sqlite.Core, який надає всі необхідні функції для взаємодії з базою даних.

Для налаштування EF Core застосовується метод розширення UseSqlite, який приймає параметри підключення до БД.

Реєстрація контексту БД у системі ін'єкції залежностей відбувається через метод AddDbContext, що приймає делегат для налаштування параметрів контексту options, де викликається метод UseSqlite з рядком підключення, який містить локальний шлях до файлу БД, що визначає розташування та ім'я файлу SQLite.

У результаті такого налаштування створюється файл БД app.db, який спочатку містить лише службові таблиці Sqlite. Для створення необхідних таблиць на основі моделей даних можна використовувати механізм міграцій, але для першого запуску або швидкого тестування достатньо викликати метод EnsureCreated, який автоматично створює схему БД на основі наявних моделей (рисунок 3.8).

Ініціалізація БД при запуску додатку відбувається шляхом створення області видимості сервісів через метод CreateScope для управління життєвим циклом об'єктів, отримання екземпляру контексту БД db через метод GetRequiredService, що має параметр типу ApplicationDbContext, та виклику методу EnsureCreated для властивості Database, який перевіряє наявність БД і створює її з усіма необхідними таблицями згідно з моделями, якщо база ще не існує.

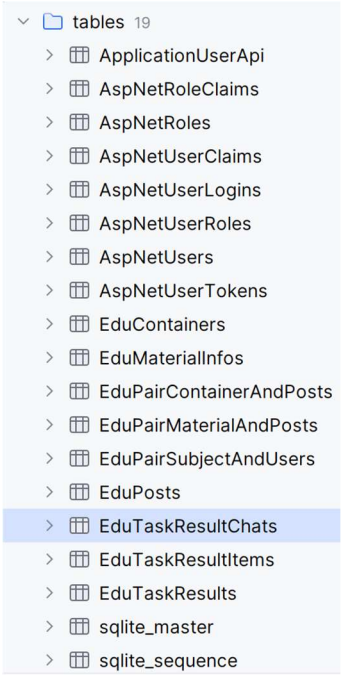


Рисунок 3.8 — Створені таблиці після виклику методу EnsureCreated

Варто зазначити, що метод EnsureCreated підходить для початкової розробки та тестування, але для виробничого середовища та складних змін схеми даних рекомендується використовувати повноцінні міграції.

Для виробничого середовища обрано PostgreSQL — потужну відкриту об'єктно-реляційну систему управління базами даних, яка забезпечує високу продуктивність, надійність та масштабованість. PostgreSQL підтримує складні запити, транзакції, зовнішні ключі та інші функції, необхідні для стабільної роботи системи в умовах реального використання. Для визначення, яку базу даних застосовувати в конкретному середовищі, впроваджено умовну конфігурацію, яка перевіряє поточне середовище виконання та відповідно налаштовує контекст бази даних.

3.8 Впровадження Swagger для документації API

Для забезпечення зручної документації та тестування API розробленої системи було інтегровано Swagger (OpenAPI) з використанням пакетів Swashbuckle [13]. Ця інтеграція дозволяє автоматично генерувати інтерактивну документацію на основі контролерів API та їх методів, що значно спрощує розробку, тестування та подальшу підтримку системи [21].

Ключовим аспектом впровадження Swagger стало налаштування API-версіонування, що забезпечує можливість одночасної підтримки різних версій інтерфейсу. Це особливо важливо для систем, де необхідно зберігати зворотну сумісність для існуючих клієнтів при впровадженні нових функцій. Вказавши в конфігурації формат версіонування та стандартну версію, система автоматично розподіляє кінцеві точки за відповідними групами в документації.

Конфігурація версіонування для Swagger включає виклик методу AddApiVersioning з параметром options, де встановлюються основні налаштування: DefaultApiVersion зі значенням нового об'єкта ApiVersion з параметрами 1 та 0, що визначає версію API за замовчуванням як 1.0; AssumeDefaultVersionWhenUnspecified зі значенням true для використання версії за замовчуванням, коли конкретна версія не вказана; ReportApiVersions зі значенням true для включення інформації про підтримувані версії у відповіді. Ланцюжок методів продовжується викликом AddApiExplorer з параметром options, де встановлюється GroupNameFormat зі значенням «'v'VVV» для форматування імен груп у Swagger UI як «v1», «v2» тощо, та SubstituteApiVersionInUrl зі значенням true для автоматичної заміни версії в URL-адресах.

Значну увагу приділено інтеграції механізмів автентифікації з інтерфейсом Swagger. Налаштовано підтримку JWT-токенів, що дозволяє розробникам та тестувальникам авторизуватися безпосередньо з інтерфейсу документації та тестувати захищені кінцеві точки без необхідності використання додаткових інструментів. Це суттєво прискорює цикл розробки та тестування API.

Налаштування Swagger з підтримкою JWT-автентифікації відбувається через метод AddSwaggerGen з параметром c, де використовується метод

AddSecurityDefinition з параметрами «Bearer» та об'єкт OpenApiSecurityScheme з властивостями Name зі значенням «Authorization», Type для типу схеми безпеки, Scheme зі значенням «Bearer», BearerFormat зі значенням «JWT», In для розміщення токена в заголовку та Description з інструкцією для введення токена. Також використовується метод AddSecurityRequirement з об'єктом OpenApiSecurityRequirement, який пов'язує схему безпеки «Bearer» з API-операціями через Reference.

Для забезпечення високої якості документації впроваджено кілька спеціалізованих фільтрів та конвертерів. Особливо важливим є налаштування коректного відображення спеціальних типів даних, таких як DateOnly, та впровадження підтримки поліморфізму для складних ієрархій класів. Завдяки фільтру JsonPolymorphicSchemaFilter система коректно відображає в документації різні типи публікацій (сповіщення, завдання), що базуються на спільному батьківському класі, з чітким розмежуванням їх специфічних властивостей та призначення.

Підвищенню зрозумілості та зручності використання API сприяє налаштування прикладів запитів та відповідей для кожної кінцевої точки. Застосування фільтрів прикладів дозволяє автоматично генерувати зразки даних, які демонструють коректний формат запитів та очікуваний формат відповідей. Це суттєво знижує поріг входження для нових розробників, які починають працювати з API системи (рисунок 3.9).

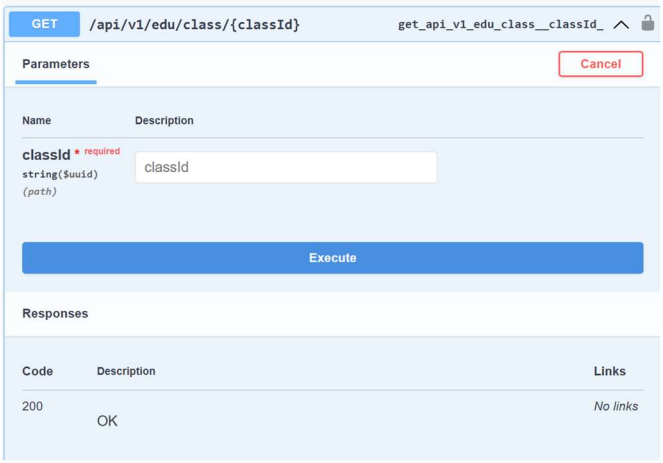


Рисунок 3.9 — Створення запиту отримання інформації про клас у Swagger

Важливим аспектом безпеки є умовне підключення Swagger UI лише в середовищі розробки. Це запобігає доступу до документації API у виробничому середовищі, що могло б розкрити деталі внутрішньої структури системи потенційним зловмисникам. При цьому інтерфейс Swagger UI налаштовано таким чином, щоб максимально спростити навігацію та роботу з документацією: включено поглиблені посилання, відображення ідентифікаторів операцій, фільтрацію ендпоінтів та валідацію вхідних даних.

Таким чином, впровадження Swagger не лише забезпечує автоматичну генерацію технічної документації API, але й створює повноцінне інтерактивне середовище для вивчення та тестування функціональності системи. Це значно полегшує інтеграцію клієнтських додатків, прискорює процес розробки та покращує комунікацію між різними командами, які працюють над проєктом.

3.9 Структура контролерів

У розробленій системі контролери API виступають основним інтерфейсом взаємодії клієнтських додатків із серверною частиною. Розглянемо структуру типового контролера на прикладі EduClassController, який відповідає за операції з навчальними класами.

Приклад контролера EduClassController демонструє використання різних атрибутів для налаштування маршрутизації, версіонування та безпеки. Клас позначений атрибутом ApiController, який маркує його як контролер API та забезпечує автоматичну валідацію моделей та форматування відповідей. Атрибут ApiVersion з параметром «1» вказує на версію API, що дозволяє підтримувати кілька версій інтерфейсу паралельно. Атрибут Route з шаблоном «api/v{version:apiVersion}/edu/class» визначає базовий маршрут для всіх кінцевих точок з динамічним сегментом для версії. Конструктор контролера приймає параметр context типу ApplicationDbContext для доступу до бази даних.

Метод GetClassInfoAsync демонструє використання атрибутів для конфігурації окремої кінцевої точки: атрибут HttpGet з параметром «{classId:guid}» визначає HTTP-метод та шаблон маршруту з параметром типу Guid; атрибут

Authorize з параметрами AuthenticationSchemes зі значенням JwtBearerDefaults.AuthenticationScheme та Policy зі значенням MyPolicy.Access обмежує доступ лише для автентифікованих користувачів з валідним JWT-токеном та відповідною політикою; атрибут ApplicationUser є користувацьким атрибутом для спрощеного доступу до даних поточного користувача.

Атрибути контролера та їх призначення:

— ApiControllerAttribute — маркує клас як контролер API, забезпечуючи автоматичну валідацію моделей, прив'язку параметрів та форматування відповідей;

— ApiVersionAttribute — вказує версію API для контролера, що дозволяє підтримувати кілька версій API паралельно та забезпечує сумісність;

— RouteAttribute — визначає базовий маршрут для всіх кінцевих точок контролера з динамічним сегментом для версії API;

— HttpGetAttribute — визначає HTTP метод GET, а також визначає маршрут кінцевої точки з параметром (у нашому випадку з параметром classId типу GUID);

— AuthorizeAttribute — обмежує доступ до ендпоінтів лише для автентифікованих користувачів з валідним JWT-токеном та відповідною політикою доступу;

— ApplicationUserAttribute — користувацький атрибут для спрощеного доступу до даних поточного користувача.

Також для роботи з базою даних та іншими сервісами в контролерах необхідно використовувати механізм впровадження залежностей, який дозволяє отримати доступ до потрібних ресурсів без створення тісних зв'язків між компонентами системи. Цей підхід не лише спрощує тестування, але й забезпечує більш гнучку архітектуру, де компоненти можна легко замінювати або модифікувати.

У розробці застосовуються три основні типи впровадження залежностей:

— впровадження через конструктор — залежності передаються через параметри конструктора, що гарантує їх наявність при створенні об'єкта;

— впровадження через властивості — залежності встановлюються через публічні властивості, що підходить для опціональних залежностей;

— впровадження через методи — залежності передаються безпосередньо в методи, що їх використовують.

У розробленій системі переважно використовується впровадження через конструктор як найбільш надійний підхід, що відповідає принципам SOLID [22]. Контекст бази даних ApplicationDbContext додається в контролер для забезпечення доступу до даних, а реєстрація сервісів здійснюється в методі ConfigureServices класу Startup або безпосередньо при конфігурації додатку з використанням методів розширення.

3.10 Створення запитів з використанням Linq

Для ефективної взаємодії з базою даних у системі активно використовується Language Integrated Query (LINQ) — засіб мови C#, що дозволяє писати типобезпечні запити до різних джерел даних. У поєднанні з Entity Framework Core LINQ забезпечує зручний та гнучкий підхід до вибірки, фільтрації та перетворення даних, що суттєво підвищує продуктивність розробки та оптимізує роботу з базою даних.

У основі LINQ лежить інтерфейс IQueryable, який забезпечує можливість побудови та виконання запитів на різних джерелах даних. Entity Framework Core реалізує цей інтерфейс для своїх DbSet, що дозволяє конструювати складні запити, які транслюються в SQL на стороні постачальника даних. Ключовою особливістю IQueryable є підтримка дерева виразів (Expression Trees) — структур даних, що представляють код як об'єкти. Це дозволяє аналізувати, модифікувати та транслювати LINQ-запити в SQL-вирази без їх негайного виконання.

Одним із типових сценаріїв використання LINQ у системі є отримання списку ідентифікаторів постів, пов'язаних з певним контейнером (класом). Запит складається з кількох операцій: звертання до потрібного набору даних через контекст, фільтрація записів за ідентифікатором контейнера, проєкція результатів для отримання лише ідентифікаторів постів та асинхронне виконання запиту з матеріалізацією результату в список.

LINQ-запит для отримання ідентифікаторів постів починається зі звернення до властивості EduPairContainerAndPosts контексту БД, що представляє колекцію пар зв'язків між контейнерами та постами. Метод Where застосовується для фільтрації записів за ідентифікатором контейнера. Метод Select виконує проєкцію відфільтрованих результатів, трансформуючи кожен елемент у значення властивості, що дозволяє отримати лише потрібні ідентифікатори. Метод ToListAsync асинхронно виконує запит до бази даних та матеріалізує результати у список об'єктів типу Guid, який присвоюється змінній postIds.

Linq-запит перетворюється на дерево виразів, де кожен компонент запиту представлений як окремий вузол (рисунок 3.10). Метод Where трансформується у вузол виклику методу з предикатом, що містить операцію порівняння властивості ContainerId з параметром ідентифікатора. Метод Select стає вузлом проєкції, що вказує на вибір лише властивості EduPostId. Entity Framework Core аналізує це дерево і генерує оптимізований SQL-запит, який виконується на сервері бази даних без попереднього завантаження всіх записів. Викликом ToListAsync() запит матеріалізується, результати завантажуються в пам'ять програми як список ідентифікаторів, готовий для подальшої обробки.

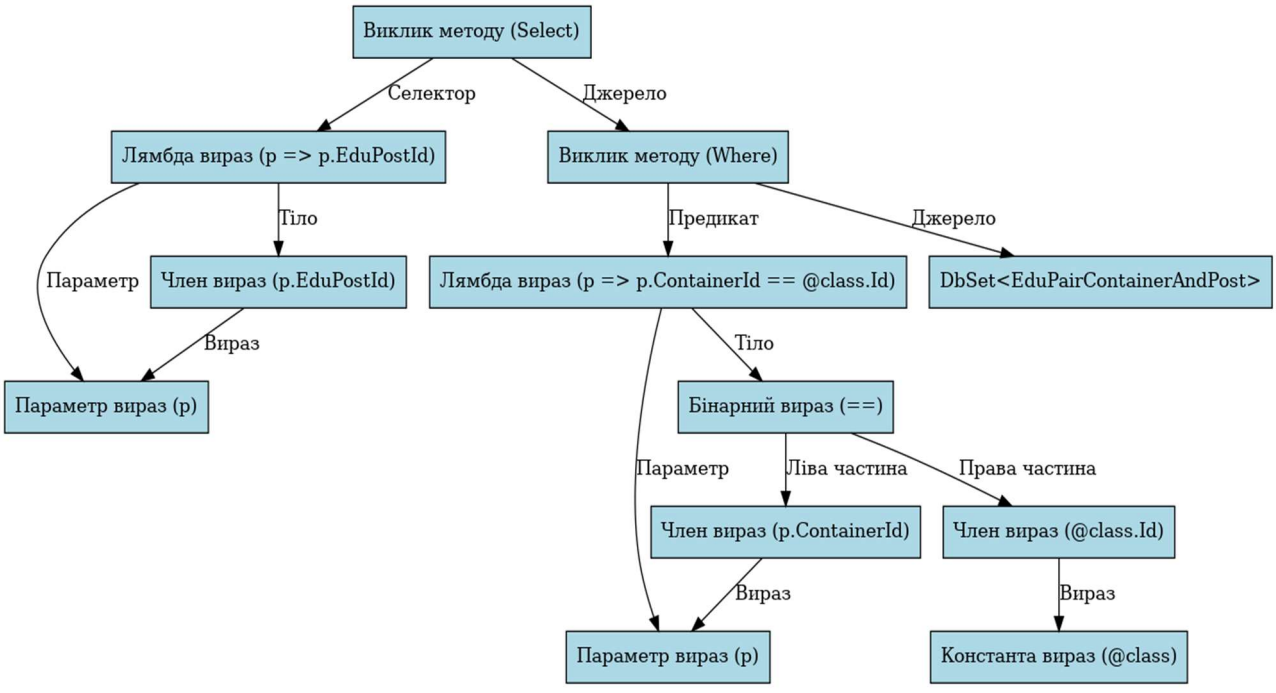


Рисунок 3.10 — Дерево виразів запиту для отримання ідентифікаторів постів

Отримані ідентифікатори постів далі використовуються для вибірки деталей кожного поста з відповідних таблиць. Це дозволяє оптимізувати запити, вибираючи лише ті дані, які дійсно потрібні, замість завантаження всіх записів та фільтрації на стороні додатку.

Результат запиту повертається клієнту у форматі JSON, що містить структуровану інформацію про контейнер та пов'язані з ним пости різних типів. Структура відповіді включає кореневий об'єкт з полями data, success та version, де поле success відображає успішність виконання запиту, а version вказує на версію API. Всередині об'єкта data міститься інформація про контейнер з його унікальним ідентифікатором id та масивом posts, який включає всі пов'язані пости. Кожен пост представлений об'єктом з властивостями: id для унікального ідентифікатора поста, title для заголовка, publicated для дати та часу публікації, type для визначення типу поста (наприклад, «task» або «notification») та додаткові специфічні для типу поля, такі як dueDate для завдань, що може бути null, якщо термін виконання не встановлено. Такий формат забезпечує клієнтським додаткам структуровану інформацію для відображення постів різних типів з усіма необхідними деталями.

3.11 Реалізація системи обміну повідомленнями

3.11.1 Реалізація чату з використанням REST API

Традиційний підхід до реалізації функціональності чату базується на використанні HTTP-запитів та REST API. У розробленій системі цей підхід реалізовано через відповідні кінцеві точки для отримання та надсилання повідомлень.

Для отримання історії повідомлень використовується метод GET, який дозволяє клієнту запитувати повідомлення для конкретного завдання та студента. Кінцева точка має шлях «task/{taskId:guid}/student/{studentId:guid}/chat», де параметри taskId та studentId представляють унікальні ідентифікатори завдання та студента відповідно. Метод GetChatHistoryAsync захищений атрибутом Authorize з параметрами AuthenticationSchemes зі значенням JwtBearerDefaults.AuthenticationScheme та Policy зі значенням MyPolicy.Access, що

обмежує доступ тільки для автентифікованих користувачів з відповідними правами. Всередині методу виконується пошук результату завдання за вказаними ідентифікаторами, потім отримуються всі повідомлення, пов'язані з цим результатом, з сортуванням за часом створення для правильного відображення хронології спілкування, після чого формується структурована відповідь з повідомленнями у форматі JSON для передачі клієнту.

Для надсилання нових повідомлень використовується метод POST, який приймає ідентифікатор студента, ідентифікатор завдання та текст повідомлення. Кінцева точка має шлях "student/{studentId:guid}/chat" з параметром studentId та приймає додаткові параметри postId та об'єкт dto типу AddCommentDto через тіло запиту, де містяться текст повідомлення та інші необхідні дані. Метод SendMessageToStudentAsync також захищений атрибутами Authorize для обмеження доступу та ApplicationUser для отримання інформації про поточного користувача. Всередині методу спочатку виконується перевірка прав доступу поточного користувача для взаємодії з вказаним студентом та завданням, потім відбувається пошук існуючого результату завдання для цього студента у базі даних, і якщо такий не знайдено, створюється новий запис результату з відповідними зв'язками. Після цього створюється нове повідомлення з текстом з dto, встановлюється автор повідомлення на основі поточного користувача, прив'язується повідомлення до результату завдання, зберігаються всі зміни у базі даних, і формується відповідь з результатом операції, що включає створене повідомлення та статус успішності.

Важливою особливістю реалізації є автоматичне створення контексту чату при першому повідомленні. Це спрощує процес ініціації комунікації, оскільки не потрібно окремо створювати чат перед початком обміну повідомленнями. Коли викладач відправляє перше повідомлення студенту щодо конкретного завдання, система автоматично створює необхідні структури даних (рисунок 3.11).

REST-підхід для реалізації чату має свої переваги та обмеження. З одного боку, він добре інтегрується з загальною архітектурою системи, простий у реалізації та надійний у доставці повідомлень. З іншого боку, відсутність механізму

повідомлень у реальному часі змушує клієнтів періодично опитувати сервер для отримання нових повідомлень, що створює додаткове навантаження та затримки.

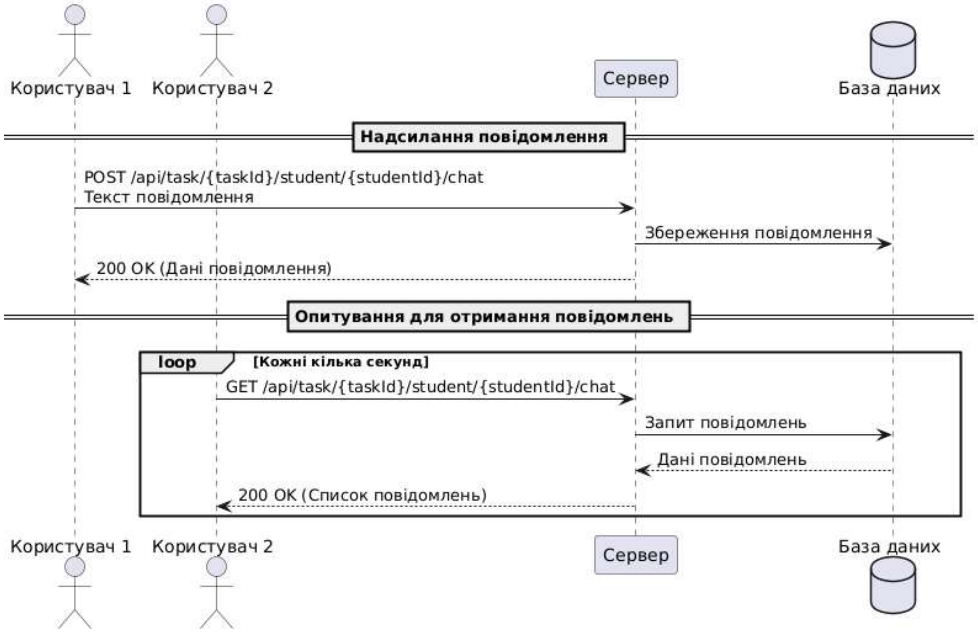


Рисунок 3.11 — Схема комунікації через REST API

3.11.2 Реалізація чату з використанням SignalR

Для подолання обмежень REST-підходу та забезпечення комунікації в реальному часі в системі реалізовано функціональність чату з використанням технології SignalR, яка базується на WebSockets.

Налаштування SignalR виконується при конфігурації сервісів додатку через метод AddSignalR з параметром options, де встановлюється властивість EnableDetailedErrors зі значенням true для отримання детальної інформації про помилки під час розробки та налагодження, що полегшує пошук та виправлення проблем в реальному часі.

SignalR надає потужний механізм для двосторонньої комунікації між клієнтом і сервером. На сервері створено спеціальний клас ChatHub, який обробляє з'єднання клієнтів, управляє групами користувачів та забезпечує передачу повідомлень. Клас успадковує базовий клас Hub, що надає основну функціональність для роботи з підключеннями, та приймає в конструкторі параметр context типу ApplicationDbContext для доступу до бази даних. Клас

захищений атрибутом Authorize, що обмежує доступ до хабу лише автентифікованим користувачам з відповідними правами.

Метод Send приймає параметр message типу string та виконує операції: отримує дані користувача з контексту підключення, визначає контекст чату, створює та зберігає нове повідомлення, відправляє його всім клієнтам у відповідній групі, забезпечуючи миттєву доставку повідомлення.

Метод OnConnectedAsync викликається при підключенні нового клієнта, отримує параметри підключення, визначає цільового користувача, знаходить або створює контекст чату та додає клієнта до групи SignalR з унікальним ідентифікатором, що відповідає комбінації завдання та студента.

Метод OnDisconnectedAsync викликається при відключенні клієнта, приймає параметр exception для визначення причини та видаляє клієнта з усіх груп, забезпечуючи коректне управління ресурсами системи.

Кожен чат (визначений унікальною комбінацією завдання та студента) представлений як окрема група в SignalR, що дозволяє точно адресувати повідомлення лише тим користувачам, які повинні їх отримати.

Реєстрація хабу в системі маршрутизації відбувається при налаштуванні конвеєра обробки запитів через метод MapHub, який приймає тип хабу ChatHub та шлях «/api/v1/ws/chat», за яким клієнти будуть підключатися до хабу через WebSocket-з'єднання.

Особливістю реалізації є збереження повідомлень у БД при використанні SignalR, що забезпечує постійність даних та доступ до історії для користувачів, які не були онлайн під час комунікації (рисунок 3.12).

3.11.3 Порівняльний аналіз підходів до реалізації чату

Для кількісного порівняння ефективності розглянутих підходів проведено аналіз мережевого трафіку при різних сценаріях використання. На таблиці 3.2 показано приблизне порівняння обсягу трафіку за 15 хвилин активного використання чату з повідомленнями розміром 250 байт, що надсилаються кожні 15 секунд.

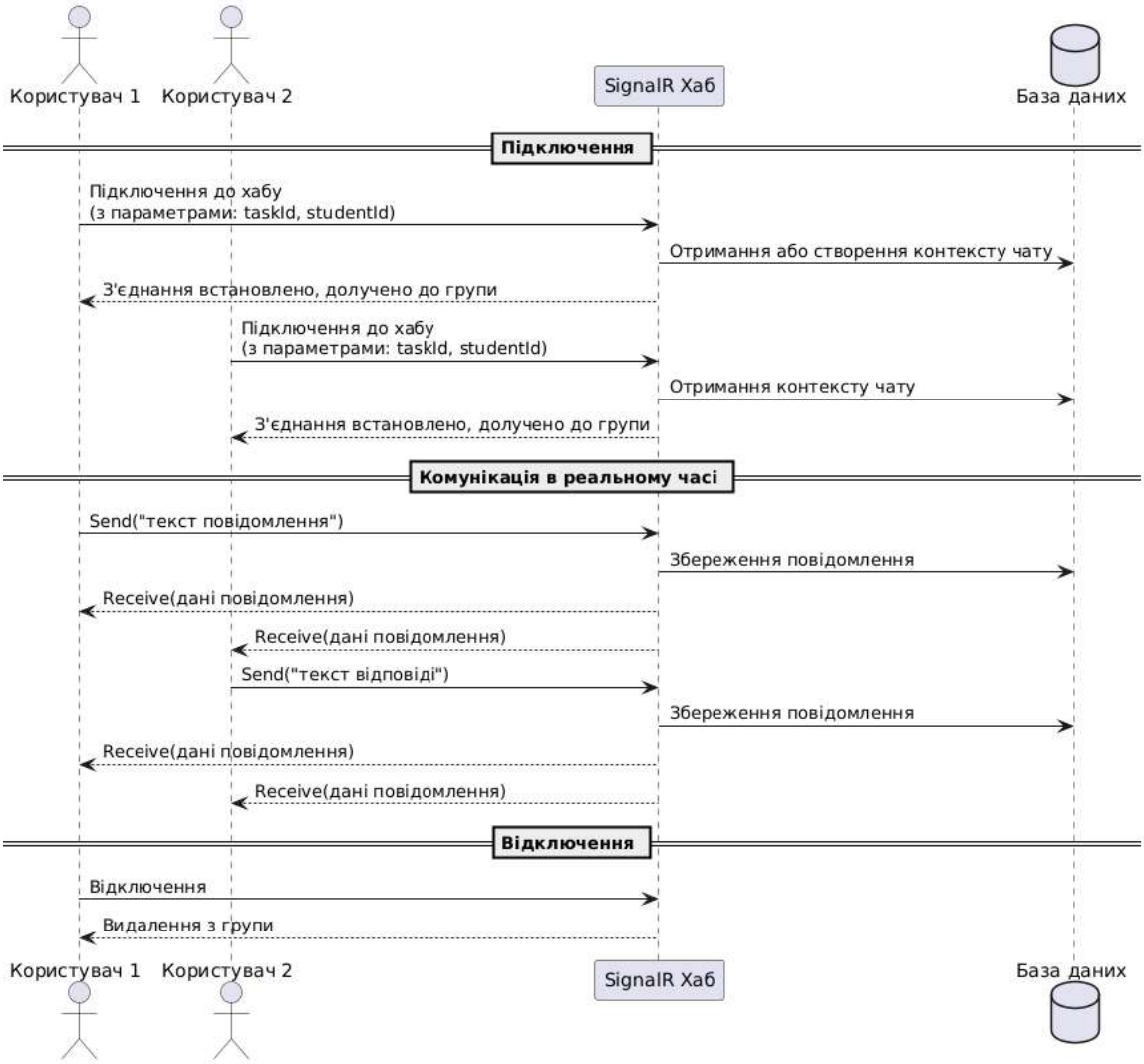


Рисунок 3.12 — Схема комунікації через SignalR

Таблиця 3.2. Порівняння мережевого трафіку при різних підходах до реалізації чату

Параметр	REST API (3 сек)	REST API (10 сек)	SignalR (WebSockets)
Кількість HTTP-запитів	300	90	1 (+ повідомлення)
Заголовки HTTP, байт	300 · 800 = 240 000	90 · 800 = 72 000	800 + 60 · 50 = 3 800
Корисне навантаження, байт	60 · 250 = 15 000	60 · 250 = 15 000	60 · 250 = 15 000
Додатковий трафік для з'єднання, байт	0	0	15 · 60 · 11 = 9 900
Загальний трафік, байт	255 000	87 000	28 700
Середня затримка доставки, мс	1 500	5 000	<100

Аналіз даних таблиці показує, що використання SignalR забезпечує значне зменшення мережевого трафіку порівняно з REST API, особливо при частому опитуванні. Навіть з урахуванням додаткового трафіку для підтримки з'єднання, загальний обсяг трафіку при використанні WebSockets майже в 3 рази менший порівняно з REST API з опитуванням кожні 10 секунд, і приблизно в 9 разів менший порівняно з опитуванням кожні 3 секунди.

Варто зазначити, що для оптимізації підходу з REST API можна впровадити додаткові механізми, такі як:

— умовні запити з використанням HTTP-заголовка ETag;

— стиснення відповідей за допомогою gzip або brotli;

— довготривалі опитування (long polling) замість звичайного опитування;

— інкрементальне отримання даних (лише нових повідомлень).

Однак навіть з цими оптимізаціями, WebSockets все одно забезпечує більш реактивний досвід та ефективніше використання мережевих ресурсів. Особливо значною є різниця в затримці доставки повідомлень: менше 100 мс для WebSockets від 1,5 до 5 секунд для REST API з опитуванням, що критично важливо для комфортного спілкування в реальному часі.

Крім кількісних показників, важливо враховувати якісні аспекти користувацького досвіду. Комунікація в реальному часі створює ефект присутності та миттєвого відгуку, що значно покращує взаємодію між викладачами та студентами. Це особливо важливо для дистанційного навчання, де ефективна комунікація є ключовим фактором успішності освітнього процесу.

У результаті аналізу прийнято рішення використовувати гібридний підхід: основна функціональність чату реалізована через SignalR для забезпечення комунікації в реальному часі, але також підтримується REST API для отримання історії повідомлень та забезпечення сумісності з клієнтами, які не підтримують WebSockets.

Такий підхід забезпечує оптимальний баланс між ефективністю комунікації, сумісністю з різними клієнтами та використанням мережевих ресурсів, що є

критично важливим для навчальної системи, де комунікація між викладачами та студентами повинна бути максимально ефективною та зручною.

У даному розділі розглянуто розробку серверної частини навчально-інформаційної системи. Реалізовано моделювання сутностей бази даних з використанням Entity Framework Core, що забезпечило ефективну взаємодію з даними. Впроваджено ASP.NET Identity для управління користувачами та JWT-автентифікацію для захисту API-ендпоінтів. Інтеграція з Google OAuth 2.0 розширила можливості автентифікації, а реалізація сервісу електронних повідомлень забезпечила комунікацію з користувачами.

Особлива увага приділена ефективній роботі з даними через LINQ-запити та документуванню API за допомогою Swagger. Для системи обміну повідомленнями порівняно підходи REST API та SignalR, на основі чого обрано гібридне рішення, що забезпечує оптимальний баланс між ефективністю та сумісністю.

3.12 Забезпечення безпеки API через шифрування ключів доступу

Для підвищення рівня безпеки системи та надання можливості створення автоматизованих додатків, які взаємодіють з API навчальної платформи, розроблено механізм управління API-ключами з використанням криптографічного шифрування. Ця функціональність дозволяє користувачам отримувати унікальні ключі доступу для програмної взаємодії з системою без необхідності передачі основних облікових даних.

Основу механізму складає сутність ApplicationUserApi, яка зберігає зашифрований ключ API, вектор ініціалізації для шифрування та дату створення ключа. Важливою особливістю реалізації є те, що сирий ключ API ніколи не зберігається в базі даних — натомість зберігається лише його зашифрована версія, яка може бути розшифрована тільки з використанням майстер-ключа, що зберігається в захищеній конфігурації системи.

Для управління API-ключами розроблено два сервісних класи: ApiUserManager та ApiKeyEncryptionService. Перший відповідає за високорівневі

операції з ключами, а другий реалізує криптографічні функції шифрування та розшифрування з використанням алгоритму AES.

Метод GenerateRandomKeyAsync приймає параметр userId типу Guid та виконує пошук існуючого запису для вказаного користувача або створює новий, генерує новий випадковий вектор ініціалізації та шифрує новий ключ за допомогою криптографічного сервісу, зберігає зашифрований ключ та вектор ініціалізації в базі даних, після чого повертає оригінальний незашифрований ключ користувачу для використання в API-запитах.

Метод ValidateKeyAsync приймає параметри api типу ApplicationUserApi та guid типу Guid, розшифровує збережений в api ключ за допомогою збереженого вектора ініціалізації та порівнює розшифрований ключ з наданим guid, повертаючи результат порівняння, що вказує на успішність або невдачу валідації.

При генерації нового ключа доступу спочатку створюється випадковий вектор ініціалізації, а потім генерується випадковий GUID, який шифрується з використанням майстер-ключа та вектора ініціалізації. Зашифрований ключ та вектор ініціалізації зберігаються в базі даних, а оригінальний GUID повертається користувачу (рисунок 3.13). Таким чином, навіть при компрометації бази даних зловмисник не зможе отримати доступ до API без знання майстер-ключа.

Валідація ключа відбувається шляхом розшифрування збереженого значення та порівняння з ключем, наданим у запиті. Це забезпечує надійну автентифікацію клієнтських додатків без необхідності передачі чутливих даних користувача.

Реалізований механізм API-ключів надає ряд переваг:

— забезпечує додатковий рівень безпеки, оскільки основні облікові дані користувача не використовуються при програмній взаємодії;

— дозволяє створювати автоматизовані рішення та інтеграції з іншими системами;

— надає можливість обмеження прав доступу для API-ключів порівняно з повним доступом користувача;

— забезпечує можливість відкликання окремих ключів без необхідності зміни основного пароля користувача.

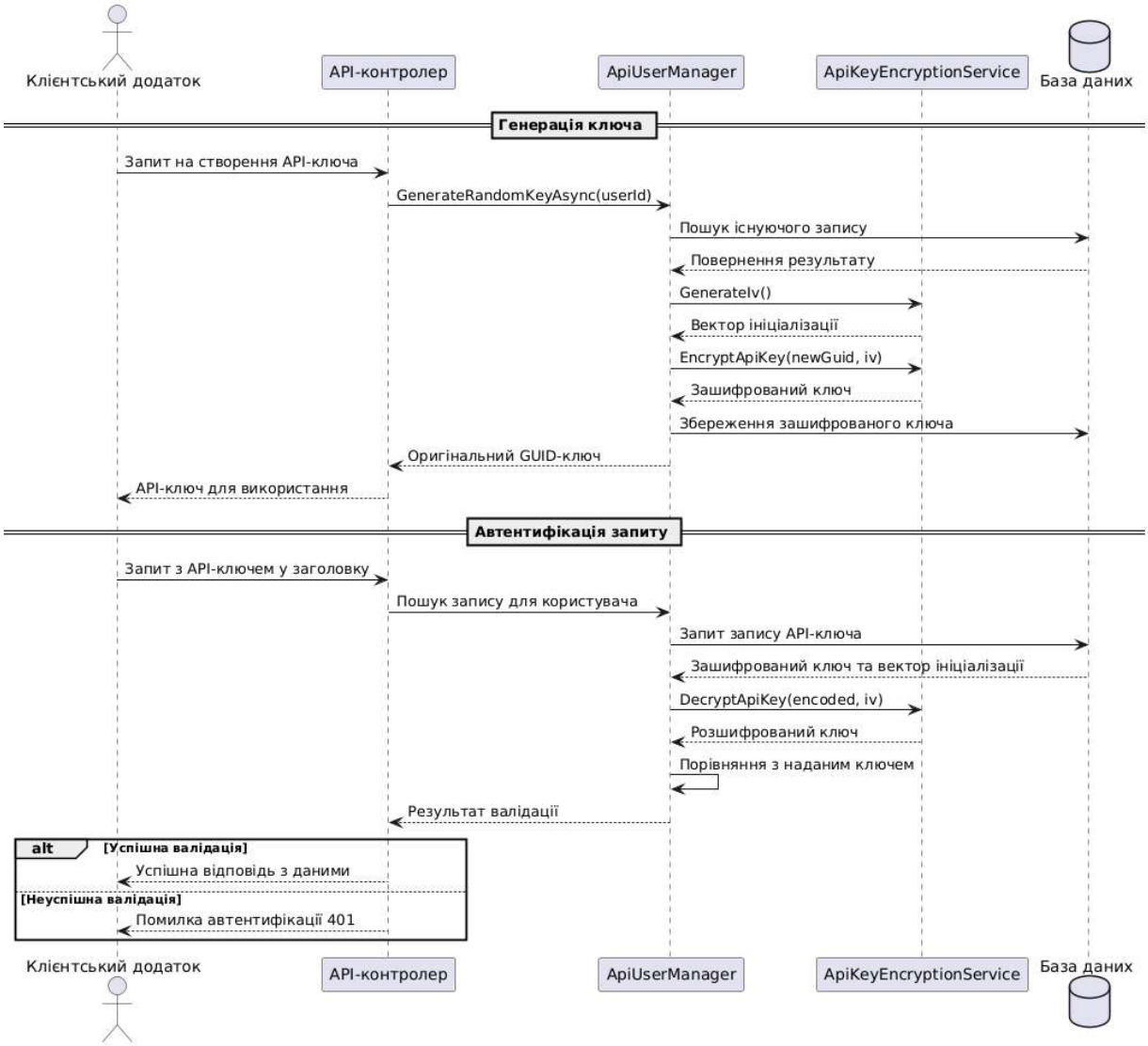


Рисунок 3.13 — Процес автентифікації за API-ключем

4 РЕАЛІЗАЦІЯ КЛІЄНТСЬКОЇ ЧАСТИНИ НАВЧАЛЬНО-ІНФОРМАЦІЙНОЇ СИСТЕМИ

4.1 Вибір технологічного стеку для клієнтської частини

Розробку клієнтської частини навчально-інформаційної системи виконано на базі фреймворку Next.js, який розширює можливості React.js додатковими функціями серверного рендерингу (SSR), статичної генерації сторінок (SSG) та інкрементальної статичної регенерації (ISR) [20].

Для стилізації компонентів застосовано Tailwind CSS версії 4.1.6 з бібліотекою компонентів DaisyUI версії 5.0.35, що забезпечує набір готових UI-компонентів та узгоджену систему дизайну. Управління станом запитів до API реалізовано через React Query версії 5.75.7, що дозволяє ефективно кешувати дані та обробляти стани завантаження.

Відображення та обробку Markdown-контенту в системі забезпечують бібліотеки react-markdown, markdown-it, marked та remark-gfm, що дозволяє підтримувати різні формати розмітки документів. Для безпечного відображення HTML-контенту використано dompurify, rehype-sanitize та rehype-raw.

Для роботи з JWT-токенами інтегровано бібліотеку jwt-decode версії 4.0.0. Візуальне представлення іконок реалізовано через react-icons версії 5.5.0, що надає доступ до різних наборів іконок у форматі React-компонентів.

Для комунікації в реальному часі впроваджено клієнт SignalR, що забезпечує двосторонній зв'язок між клієнтською та серверною частинами додатку.

Процес встановлення залежностей включає декілька команд npm install для різних груп бібліотек: Tailwind CSS з DaisyUI для стилізації та дизайн-системи; React Query для управління станом запитів та кешування даних; бібліотеки для роботи з Markdown-розміткою та безпечного відображення HTML; jwt-decode для обробки JWT-токенів; react-icons для системи іконок; та клієнт SignalR для реалтайм-комунікації.

4.2 Структура та організація клієнтської частини додатку

Клієнтську частину навчально-інформаційної системи реалізовано відповідно до архітектурних принципів фреймворку Next.js з використанням сучасних підходів до організації коду. Структуру проекту побудовано з урахуванням максимальної модульності, повторного використання компонентів та чіткого розділення відповідальності між різними частинами додатку.

Ключовими директоріями проекту є:

— app — основна директорія додатку Next.js, що містить файлову структуру маршрутизації згідно з підходом App Router;

— components — багаторазово використовувані React-компоненти, організовані за функціональним призначенням;

— context — контексти React для управління глобальним станом додатку;

— hooks — користувацькі React-хуки для інкапсуляції логіки компонентів;

— lib — службові функції та утиліти;

— message — файли локалізації для підтримки багатомовності;

— modals — компоненти модальних вікон для інтерактивної взаємодії з користувачем;

— services — сервіси для взаємодії з API та іншими зовнішніми системами.

Особливе значення має директорія app, яка відповідно до підходу App Router у Next.js визначає структуру маршрутизації додатку [15]. Кожна піддиректорія в ній відповідає окремому маршруту або групі маршрутів, а файли page.tsx визначають вміст сторінок.

Організація проекту відповідає сучасним підходам до структурування React-додатків з використанням Next.js. Маршрутизація на основі файлової системи (App Router) значно спрощує навігацію по проекту та розуміння зв'язків між різними частинами додатку.

Архітектура додатку побудована з дотриманням принципу розділення відповідальності:

— компоненти відповідають за відображення даних та взаємодію з користувачем;

— контексти забезпечують доступ до глобального стану;

— хуки інкапсулюють логіку взаємодії з API та управління станом;

— сервіси надають абстракції для взаємодії з зовнішніми системами;

— модальні вікна реалізують інтерактивну взаємодію з користувачем для виконання конкретних дій.

Важливою особливістю розробленої структури є організація мок-API-маршрутів в директорії app/api, що дозволяє реалізувати мок-серверні функції безпосередньо в межах Next.js додатку без необхідності розгортання окремого API-сервера. Це суттєво спрощує розробку та розгортання додатку, особливо для початкових етапів реалізації.

Розділення компонентів на спільні (у директорії components) та специфічні для окремих секцій (у піддиректоріях app/*/components) дозволяє ефективно управляти повторним використанням коду та уникнути дублювання функціональності.

Така організація коду забезпечує високу підтримуваність проєкту, полегшує навігацію по кодовій базі та спрощує подальше розширення функціональності системи за рахунок чіткого розділення відповідальності між різними модулями проєкту.

4.3 Реалізація HTTP-клієнта для взаємодії з REST API

Для ефективної взаємодії з серверним REST API розроблено спеціалізований HTTP-клієнт на основі бібліотеки axios [25]. Клієнт реалізує механізм автоматичного оновлення токенів авторизації та обробки помилок автентифікації[18]. Основними функціональними компонентами клієнта є:

— механізм кешування JWT-токенів у пам'яті для оптимізації продуктивності;

— функції валідації та декодування токенів з використанням бібліотеки jwt-decode;

— система автоматичного оновлення токенів при закінченні терміну їх дії;

— перехоплювачі запитів та відповідей для інтеграції авторизаційної логіки;

— механізм уникнення race condition при одночасному оновленні токенів;

— система сповіщення компонентів додатку про стан авторизації.

Конфігурацію HTTP-клієнта здійснено з базовими параметрами, такими як базова URL-адреса API, заголовки за замовчуванням та таймаут запитів. У разі отримання від сервера коду помилки 401 (Unauthorized), клієнт автоматично намагається оновити токен і повторити запит (рисунок 4.1, 4.2).

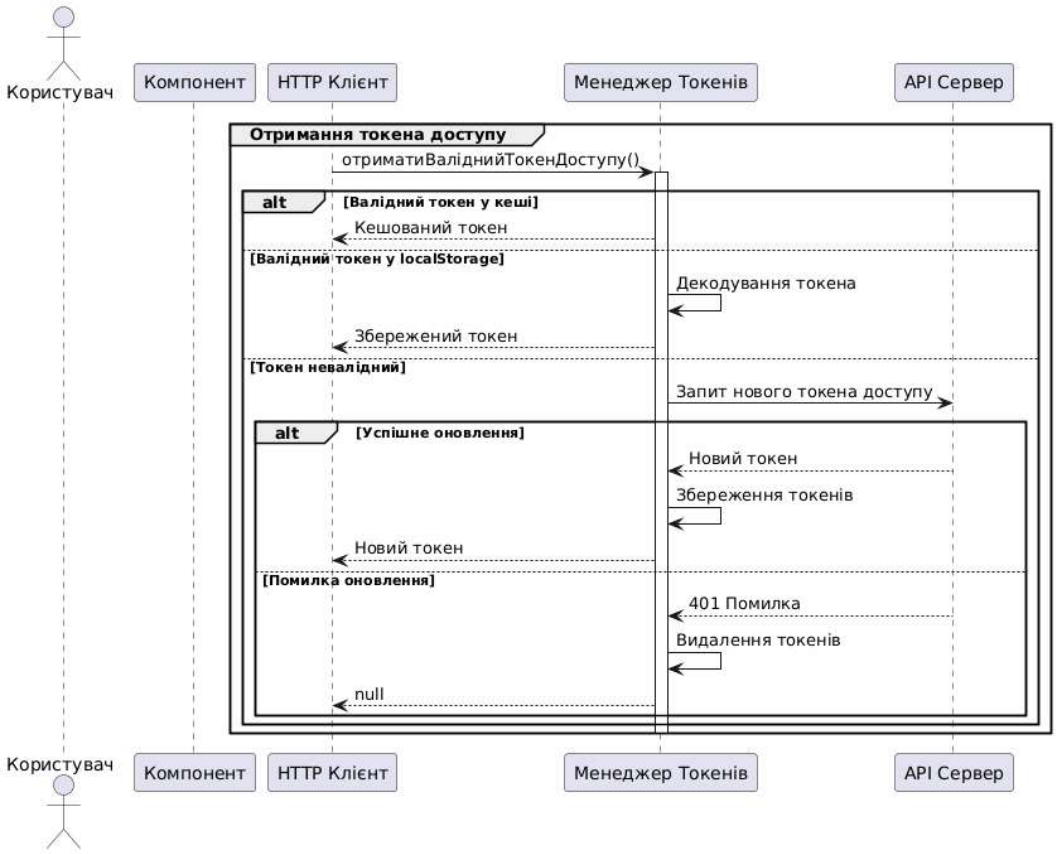


Рисунок 4.1 — Процес отримання токена доступу

Конфігурація HTTP-клієнта включає імпорт бібліотеки axios, визначення константи API_URL на основі змінної середовища NEXT_PUBLIC_API_URL з резервним значенням «/api», та створення екземпляра axios з базовими налаштуваннями: baseURL для встановлення базової URL-адреси API, headers з Content-Type зі значенням «application/json» для визначення формату даних, та timeout зі значенням 10000 мілісекунд для обмеження часу очікування відповіді.

Перехоплювач запитів налаштовано через метод interceptors.request.use, який приймає асинхронну функцію з параметром config для доступу до конфігурації запиту, отримує валідний токен доступу та додає його до заголовків запиту, забезпечуючи автентифікацію для кожного запиту до API.

Перехоплювач відповідей реалізовано через метод interceptors.response.use з двома функціями: перша просто повертає успішну відповідь, а друга асинхронно обробляє помилки, зокрема перевіряє код помилки 401, реалізує логіку запобігання повторним спробам оновлення токена та повторює оригінальний запит з новим токеном після успішного оновлення.

Модуль експортує налаштований екземпляр API-клієнта як значення за замовчуванням, а також додатково експортує константу API_URL та функцію extractIdentityFromToken для використання в інших частинах додатку.

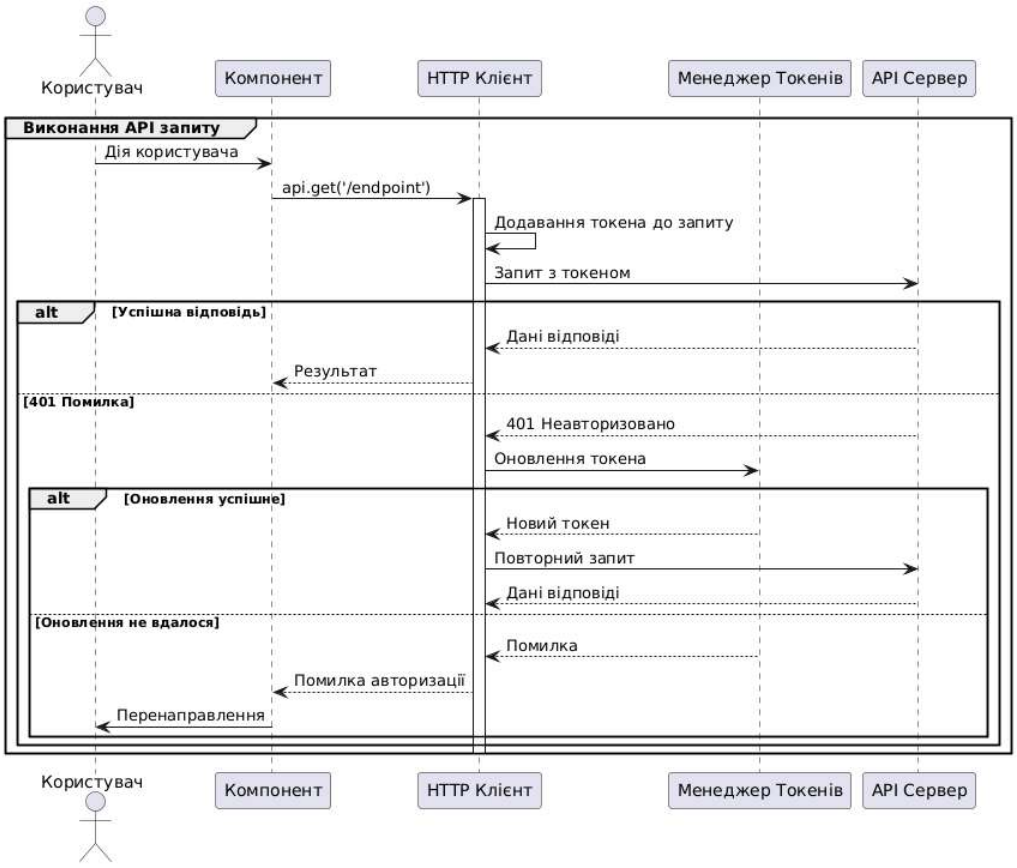


Рисунок 4.2 — Процес отримання доступу до ресурсу

Реалізація HTTP-клієнта забезпечує ефективну роботу з JWT-токенами, мінімізуючи кількість запитів до сервера через механізм кешування токенів у

пам'яті. Використання асинхронного патерну при оновленні токенів дозволяє уникнути дублювання запитів на оновлення у випадку паралельного виконання кількох запитів до API.

4.4 Реалізація контексту автентифікації AuthContext

Для забезпечення надійного управління станом автентифікації користувачів у клієнтській частині навчально-інформаційної системи розроблено спеціалізований контекст автентифікації з використанням технології React Context API. Такий архітектурний підхід дозволяє централізовано керувати процесами автентифікації та авторизації в масштабах усього додатку, забезпечуючи єдину точку доступу до інформації про стан користувача та відповідних методів управління.

Контекст автентифікації реалізовано за допомогою шаблону проектування Provider-Consumer, де AuthProvider виступає постачальником даних автентифікації, а користувацький хук useAuth забезпечує зручний доступ до цих даних у будь-якому компоненті додатку. Це суттєво спрощує архітектуру додатку, усуваючи необхідність передавати дані та функції автентифікації через ланцюжок властивостей компонентів (prop drilling).

Основними функціональними складовими контексту автентифікації є:

— стан автентифікації (isAuthenticated) для відстеження наявності активної сесії користувача;

— дані ідентифікації користувача (identity), що містять мінімально необхідну інформацію для функціонування системи;

— функція входу в систему (login), яка обробляє отримані від сервера токени, зберігає їх у локальному сховищі та оновлює стан додатку;

— функція виходу з системи (logout), що виконує очищення токенів, скидання стану автентифікації та перенаправлення на сторінку входу;

— функція вилучення ідентифікаційних даних з JWT-токена з валідацією терміну його дії.

Важливою особливістю реалізації є інтеграція з React Query через доступ до queryClient [6] для очищення кешу даних при виході з системи, що запобігає потенційним проблемам безпеки, пов'язаним зі збереженням конфіденційних даних між сесіями різних користувачів.

Для забезпечення стійкості системи до помилок автентифікації впроваджено механізм обробки подій «auth:unauthorized», які генеруються HTTP-клієнтом у випадку невдалого оновлення токенів. Такий підхід дозволяє оперативно реагувати на ситуації, коли токени стають недійсними, автоматично перенаправляючи користувача на сторінку входу.

При ініціалізації додатку контекст автоматично перевіряє наявність збережених токенів у локальному сховищі, намагаючись відновити сесію користувача без необхідності повторного введення облікових даних. Це забезпечує безперервність користувацького досвіду між сеансами роботи з додатком.

Реалізація контексту автентифікації включає інтерфейс AuthContextType, який визначає структуру контексту з чотирма ключовими елементами: властивість isAuthenticated типу boolean для відстеження стану автентифікації, властивість identity типу IdentityData або null для зберігання даних користувача, метод login, що приймає параметри access та refresh типу string для виконання входу, та метод logout без параметрів для виходу з системи.

Створення контексту відбувається через функцію createContext з початковим значенням undefined та типом AuthContextType, результат якої зберігається в константі AuthContext для подальшого використання.

Функція useAuth служить хуком для доступу до контексту в компонентах, викликає useContext з параметром AuthContext для отримання поточного значення контексту, перевіряє наявність контексту і викидає помилку, якщо контекст відсутній, та повертає значення контексту для використання в компонентах.

Компонент AuthProvider приймає параметр children типу ReactNode та реалізує логіку управління станом автентифікації, повертаючи компонент AuthContext.Provider з атрибутом value, що містить об'єкт з властивостями

isAuthenticated, identity та методами login, logout, які будуть доступні всім дочірнім компонентам через контекст.

Застосування розробленого контексту автентифікації в компонентах додатку здійснюється через хук useAuth, який надає зручний доступ до стану та методів автентифікації. Це дозволяє легко інтегрувати функціональність автентифікації в різні частини додатку, включаючи компоненти форм входу, захищені маршрути та елементи навігації, які залежать від стану автентифікації користувача.

4.5 Система сповіщень для інтерфейсу користувача

Для забезпечення ефективного зворотного зв'язку з користувачем у клієнтській частині навчально-інформаційної системи розроблено спеціалізований контекст сповіщень (AlertContext), який дозволяє відображати інформаційні повідомлення, помилки та сповіщення про успішні дії в уніфікованому форматі.

Система сповіщень реалізована за допомогою React Context API та інтегрована з компонентами UI-бібліотеки DaisyUI для забезпечення стилізованого та адаптивного відображення повідомлень. Ключовими особливостями даної реалізації є:

— підтримка різних типів сповіщень (успіх, помилка, інформація) з відповідним візуальним оформленням;

— можливість відображення як текстових повідомлень, так і складних React-компонентів;

— гнучке налаштування позиціонування сповіщень на екрані (верх, середина, низ та лівий, центральний, правий край);

— автоматичне видалення сповіщень після заданого часового інтервалу;

— можливість ручного закриття сповіщень користувачем.

Система сповіщень інкапсулює логіку управління станом повідомлень, забезпечуючи простий інтерфейс для використання в будь-якому компоненті додатку через кастомний хук useNotify.

Реалізація контексту системи сповіщень починається зі створення AlertContext за допомогою функції createContext з початковим значенням undefined

та типом AlertContextType. Компонент AlertProvider приймає параметри children типу ReactNode для дочірніх елементів, horizontal зі значенням за замовчуванням «end» та vertical зі значенням за замовчуванням «start» для визначення позиціонування сповіщень. Всередині компонента міститься логіка управління станом сповіщень, а повертається структура з провайдером AlertContext.Provider, що має атрибут value з методом notify, та контейнер div з класами для позиціонування, який містить відображення всіх активних сповіщень через компонент Alert для кожного елемента масиву notifications.

Функція useNotify служить хуком для доступу до функціональності сповіщень, викликає useContext з параметром AlertContext для отримання контексту, перевіряє наявність контексту і викидає помилку, якщо він використовується поза провайдером, та повертає контекст з методом notify для використання в компонентах.

Впровадження системи сповіщень у додатку дозволяє суттєво покращити користувацький досвід, забезпечуючи інформування про результати дій без переривання основного потоку взаємодії з інтерфейсом. Для ілюстрації практичного застосування контексту сповіщень розглянемо приклад використання в процесі автентифікації (рисунок 4.3).

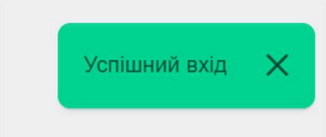


Рисунок 4.3 — Сповіщення про успішний вхід

Приклад використання системи сповіщень у процесі автентифікації демонструється через компонент Login, який використовує хуки useAuth для доступу до функціональності автентифікації, useNotify для системи сповіщень та useRouter для навігації. Метод tryLogin асинхронно виконує логіку входу: встановлює токен авторизації, отримує новий токен доступу, зберігає токени, оновлює стан автентифікації, показує сповіщення про успішний вхід через виклик notify з параметрами «Успішний вхід» та «success», та перенаправляє користувача

на сторінку додатку. У випадку помилки викликається notify з параметрами «Помилка отримання доступу» та «error», а помилка виводиться в консоль для відлагодження.

Система сповіщень є важливим елементом інтерфейсу користувача, що забезпечує інформативність та інтерактивність взаємодії. Її використання дозволяє уніфікувати механізм зворотного зв'язку в межах усього додатку, значно спрощуючи розробку та підтримку компонентів, які потребують відображення повідомлень про результати операцій.

4.6 Інтеграція React Query для ефективного управління станом запитів

Для оптимізації роботи з даними, отриманими з API, у клієнтській частині навчально-інформаційної системи впроваджено бібліотеку React Query (TanStack Query). Дана бібліотека забезпечує декларативне управління станом серверних даних, їх кешування, синхронізацію та оновлення, що суттєво спрощує взаємодію з API та покращує користувацький досвід.

Основними перевагами використання React Query у проекті є:

— автоматичне кешування результатів запитів;

— фонове оновлення даних після певного періоду застарівання (staleTime);

— автоматичні повторні спроби при невдалих запитах;

— управління станами завантаження, помилок та успішного виконання;

— можливість оптимістичних оновлень інтерфейсу;

— автоматична інвалідація та повторне отримання взаємопов'язаних даних.

Для використання React Query у всьому додатку створено спеціалізований провайдер, який інкапсулює конфігурацію та надає доступ до функціональності бібліотеки всім компонентам.

Провайдер React Query для управління станом запитів реалізовано як функціональний компонент QueryProvider, який приймає параметр children типу React.ReactNode для передачі дочірніх елементів. Всередині компонента створюється екземпляр QueryClient через useState з функцією ініціалізації, що забезпечує створення клієнта лише один раз при монтуванні компонента.

Конфігурація QueryClient включає об'єкт defaultOptions з вкладеним об'єктом queries, де встановлюються властивість staleTime зі значенням 60000 мілісекунд (1 хвилина) для визначення часу, протягом якого дані вважаються актуальними, та функція retry, яка приймає параметри failureCount та error для визначення логіки повторних спроб запитів, повертаючи false для помилок з кодом 401 (помилки авторизації) та true для інших помилок, якщо кількість невдалих спроб менше 2.

Компонент повертає структуру з провайдером QueryClientProvider, який має атрибут client зі створеним екземпляром queryClient, містить дочірні елементи та компонент ReactQueryDevtools з атрибутом initialIsOpen зі значенням false для відлагодження запитів у режимі розробки.

Конфігурація QueryClient включає встановлення базових параметрів для запитів, таких як час застарівання даних (staleTime) та логіка повторних спроб (retry). Зокрема, визначено, що дані вважаються актуальними протягом однієї хвилини, після чого відбувається їх фонове оновлення, а також встановлено логіку, за якою не виконуються повторні спроби для помилок авторизації (401), але виконується до двох повторних спроб для інших типів помилок.

Бібліотека React Query надає кілька основних хуків для роботи з даними:

— useQuery — призначений для отримання даних (GET-запити), автоматично управляє станом завантаження, кешує результати та забезпечує автоматичне оновлення;

— useMutation — використовується для модифікації даних (POST, PUT, DELETE), дозволяє оптимістично оновлювати інтерфейс та інвалідувати відповідні запити в кеші;

— useQueries — дозволяє виконувати кілька запитів паралельно з динамічною кількістю параметрів.

Для ілюстрації практичного використання React Query розглянемо приклад компонента макету додатку, який отримує дані профілю користувача.

Компонент AppSubLayout приймає параметр children та використовує хуки: useState для управління станом бічної панелі (isSidebarOpen) та активного випадаючого меню (activeDropdown); useBreadcrumbs для отримання даних

навігації; useRouter для маршрутизації; та useAuth для доступу до функції виходу з системи.

Запит на отримання даних профілю реалізовано через хук useQuery, який повертає об'єкт з трьома властивостями: data з псевдонімом profileData для отриманих даних профілю, isLoading з псевдонімом profileLoading для відстеження стану завантаження та error з псевдонімом profileError для обробки помилок. Конфігурація запиту включає queryKey у вигляді масиву [«profile», «short»] для унікальної ідентифікації запиту в кеші, та асинхронну функцію queryFn, яка виконує HTTP-запит до «/profile/me/short» через клієнт api, перевіряє успішність відповіді через response.data.success, повертає дані з response.data.data у випадку успіху, або викидає помилку з текстом з response.data.error або стандартним повідомленням, якщо дані профілю не вдалося завантажити.

У наведеному прикладі хук useQuery використовується для отримання даних профілю користувача. Кожен запит має унікальний ключ (queryKey), який використовується для ідентифікації запиту в кеші та можливості інвалідації взаємопов'язаних запитів.

Важливою особливістю другого запиту є параметр enabled, який вказує, що запит на отримання даних повинен виконуватися лише коли це потрібно.

Варто відзначити основну відмінність між використанням React Query та традиційним підходом з ручним управлінням станом запитів:

— при традиційному підході розробнику необхідно вручну управляти станами завантаження, помилок та даних, а також самостійно реалізовувати логіку кешування та оновлення;

— з React Query ці аспекти автоматизовані, що дозволяє зосередитися на бізнес-логіці додатку замість написання шаблонного коду для управління станом запитів.

Інтеграція React Query також включає використання ReactQueryDevtools, що дозволяє відлагоджувати стан запитів у браузері. Цей інструмент надає візуальний інтерфейс для перегляду активних запитів, їх стану, даних та можливостей інвалідації, що суттєво спрощує розробку та тестування додатку (рисунок 4.4).

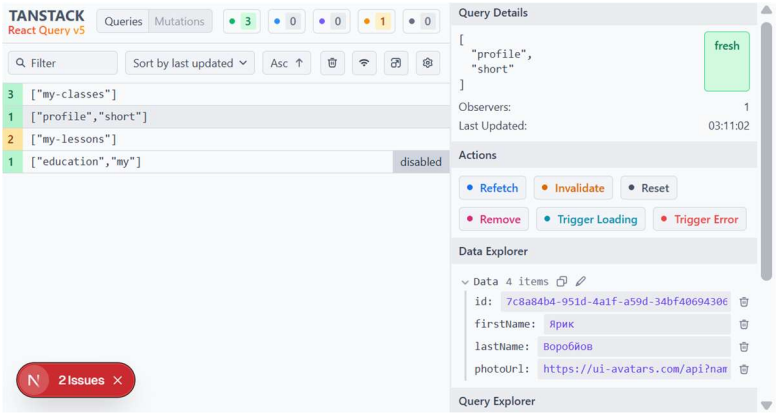


Рисунок 4.4 — Інтерфейс інструменту React Query

Для модифікації даних, наприклад при оновленні фотографії профілю, використовується хук useMutation, який дозволяє виконувати запити на зміну даних та автоматично інвалідувати відповідні запити в кеші.

4.7 Реалізація сторінки входу та інтеграція Google OAuth

Для забезпечення доступу користувачів до навчально-інформаційної системи реалізовано сторінку входу з підтримкою двох методів автентифікації: традиційного входу за допомогою електронної пошти та пароля, а також соціальної автентифікації через Google OAuth 2.0. Такий підхід дозволяє користувачам обирати найзручніший для них спосіб автентифікації, підвищуючи доступність системи (рисунок 4.5).

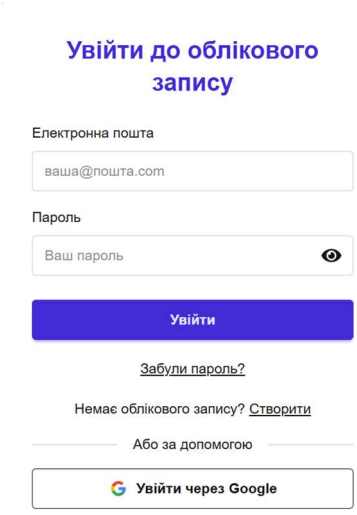


Рисунок 4.5 — Сторінка входу до облікового запису

Сторінку входу (LoginPage) розроблено з використанням компонентного підходу React.js та інтегровано з контекстом автентифікації системи через хук useAuth. Для обробки запитів авторизації використано хук useMutation, що забезпечує ефективне управління станом запиту та обробку помилок.

Основними функціональними елементами сторінки входу є:

— форма введення електронної пошти та пароля з валідацією полів;

— кнопка для перемикання видимості пароля для зручності користувача;

— механізм відображення стану запиту (завантаження) та обробки помилок;

— інтеграція з системою сповіщень для інформування про результати операцій;

— посилання для відновлення забутого пароля та створення нового облікового запису;

— компонент GoogleLoginButton для автентифікації через Google OAuth.

Сторінка входу реалізована у додатку ДОДАТИ.

Для забезпечення входу через Google OAuth 2.0 розроблено спеціалізований компонент GoogleLoginButton, який реалізує взаємодію з сервером автентифікації через механізм спливаючого вікна та обміну повідомленнями між вікнами. Цей підхід дозволяє зберегти безпеку процесу автентифікації без необхідності перезавантаження основної сторінки.

Процес автентифікації через Google OAuth 2.0 складається з наступних етапів:

— користувач натискає кнопку "Увійти через Google" на сторінці входу;

— відкривається нове вікно з запитом на авторизацію через Google;

— після успішної автентифікації сервер формує JWT-токени та передає їх назад до основного вікна через механізм postMessage;

— компонент GoogleLoginButton отримує повідомлення, витягує токен оновлення (refresh token) та передає його через callback-функцію до батьківського компонента;

— сторінка входу використовує отриманий токен для отримання токена доступу (access token) та завершення процесу автентифікації.

Такий підхід до реалізації входу через Google OAuth 2.0 забезпечує безпечний та зручний процес автентифікації, який не потребує перезавантаження сторінки та дозволяє користувачам швидко отримати доступ до системи без необхідності запам'ятовування додаткових облікових даних.

Важливою особливістю реалізації є використання механізму постійного оновлення токенів, який забезпечує безпеку системи та захист від несанкціонованого доступу. Короткий термін дії токенів доступу (access token) вимагає їх регулярного оновлення через відповідний endpoint API, що зменшує ризики, пов'язані з їх потенційним викраденням.

Інтеграція сторінки входу з контекстом автентифікації (AuthContext) та системою сповіщень (AlertContext) забезпечує централізоване управління станом автентифікації та інформування користувача про результати операцій, що підвищує якість користувацького досвіду та спрощує подальшу підтримку та розширення функціональності системи.

4.8 Система модальних вікон та інтерактивних форм

Для забезпечення ефективної взаємодії користувача з системою в клієнтській частині навчально-інформаційної платформи реалізовано гнучку систему модальних вікон, адаптованих як для десктопних, так і для мобільних інтерфейсів. Модальні вікна використовуються для створення та редагування різних сутностей системи, зокрема навчальних класів, уроків, завдань та інших елементів.

Основою реалізації модальних вікон є компонентний підхід з використанням можливостей бібліотеки DaisyUI, яка надає готові UI-компоненти з підтримкою адаптивності для різних розмірів екранів. Особливістю реалізації є використання нативного HTML-елемента dialog, який забезпечує семантично правильну структуру та доступність для асистивних технологій.

Для ілюстрації підходу до реалізації модальних вікон розглянемо компонент CreateClassModal, призначений для створення та редагування навчальних класів.

Особливістю реалізації модальних вікон з використанням DaisyUI є адаптивність до різних розмірів екранів, що забезпечується через CSS-класи modal-

bottom та sm:modal-middle. На мобільних пристроях модальне вікно відображається як «висувна панель» знизу екрана (рисунок 4.6), а на великих екранах — як традиційне центроване вікно (рисунок 4.7). Такий підхід забезпечує оптимальний користувацький досвід на різних пристроях без необхідності розробки окремих компонентів.

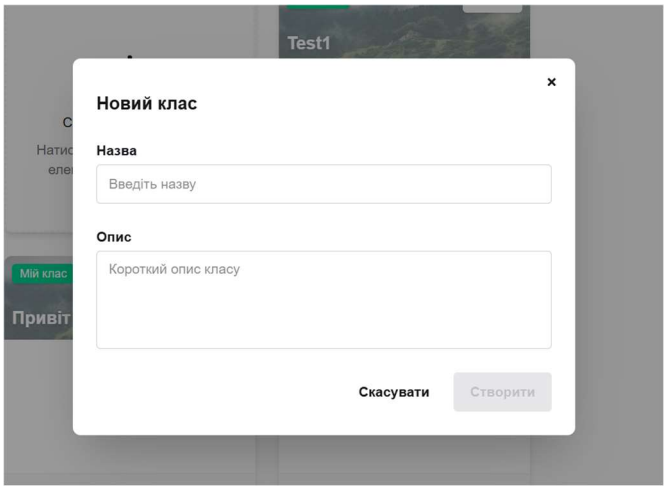


Рисунок 4.6 — Перегляд модального вікна на великому екрані

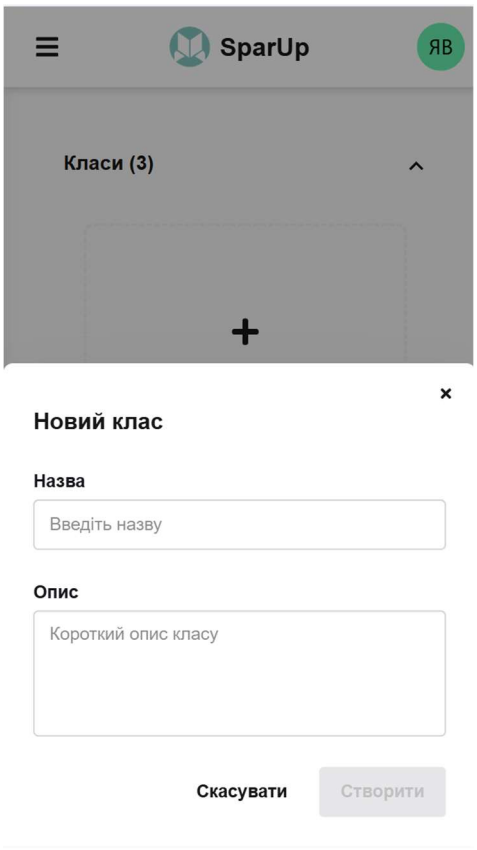


Рисунок 4.7 — Перегляд модального вікна на мобільному екрані

Важливими аспектами реалізації модальних вікон є:

— керування життєвим циклом модального вікна через властивість isOpen та методи showModal/close;

— ініціалізація початкових значень полів форми при відкритті вікна;

— обробка подій відправки форми з валідацією та зворотним зв'язком про помилки;

— блокування взаємодії з вікном під час завантаження даних;

— підтримка закриття вікна через клавішу Escape та клік за межами вікна;

— адаптивний дизайн для різних розмірів екранів.

Для використання модального вікна в компонентах додатку застосовується декларативний підхід, де властивості компонента визначають його поведінку, а функції-події забезпечують взаємодію з батьківським компонентом.

4.9 Інтеграція з SignalR для комунікації в реальному часі

У системі реалізовано два підходи до обміну повідомленнями: традиційний на основі REST API та інтерактивний з використанням SignalR. Для ізоляції логіки комунікації та спрощення інтеграції з компонентами розроблено спеціалізовані React-хуки useTaskChatRest та useTaskChat.

Перший підхід, реалізований через хук useTaskChatRest, базується на традиційній взаємодії з API через HTTP-запити. Цей підхід використовує React Query для кешування та управління станом даних, а також періодичного оновлення.

Основними компонентами REST-підходу є:

— запити GET для отримання історії повідомлень;

— запити POST для надсилання нових повідомлень;

— періодичне оновлення даних через refetchInterval;

— кешування результатів за допомогою React Query.

Недоліком цього підходу є затримка в отриманні нових повідомлень (до 60 секунд) та надлишковий мережевий трафік, оскільки кожне оновлення вимагає повного завантаження всіх повідомлень, навіть якщо нових повідомлень немає.

Для вирішення обмежень REST-підходу розроблено хук useTaskChat на основі технології SignalR, що забезпечує комунікацію в реальному часі. Цей хук встановлює постійне з'єднання з сервером через WebSockets та забезпечує миттєву доставку повідомлень.

Ключовими компонентами SignalR-підходу є:

— встановлення постійного з'єднання з сервером через WebSockets;

— автоматичне відновлення з'єднання при розривах з експоненційною затримкою;

— реєстрація обробників подій для різних станів з'єднання;

— інтеграція з React Query для управління станом повідомлень;

— обробка отримання нових повідомлень в реальному часі через метод Receive;

— відправка повідомлень через виклик методу Send.

Перевагами підходу з використанням SignalR є миттєва доставка повідомлень, зменшення мережевого трафіку (передаються лише нові повідомлення) (рисунок 4.8) та покращений користувацький досвід завдяки інтерактивності в реальному часі.

Рисунок 4.8 — Вміст трафіку під час передачі та отримання повідомлень

Для демонстрації використання розроблених хуків реалізовано компонент PrivateChat, який забезпечує відображення та надсилання повідомлень у контексті завдання (рисунок 4.9).

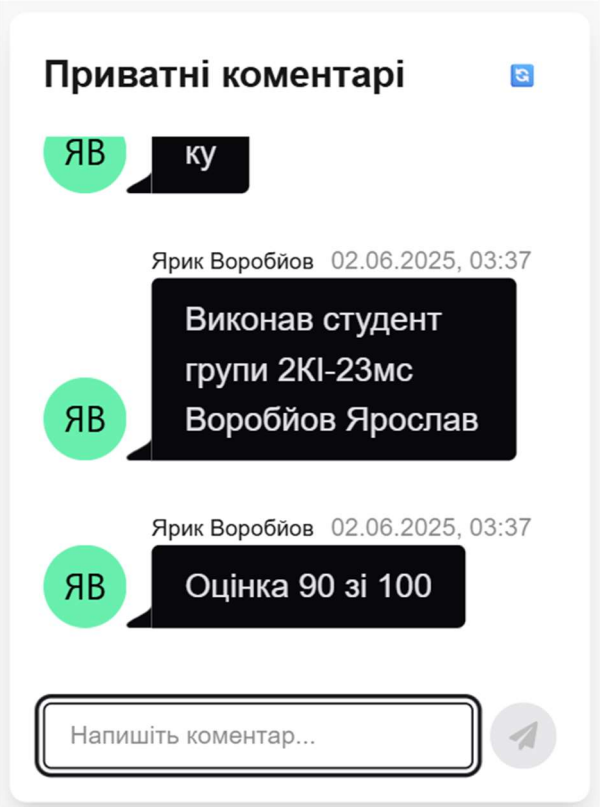


Рисунок 4.9 — Компонент PrivateChat для надсилання повідомлень

Компонент PrivateChat реалізує типовий інтерфейс чату з такими функціями:

— відображення історії повідомлень з автоматичною прокруткою до нових повідомлень;

— підтримка аватарів користувачів з автоматичною генерацією ініціалів для відсутніх зображень;

— різне оформлення для власних та чужих повідомлень;

— поле введення з підтримкою відправки через Enter;

— індикатори стану завантаження та відправки;

— інтеграція з системою сповіщень для інформування про результати операцій.

4.10 Реалізація безпечного відображення Markdown-контенту

Для забезпечення гнучкого та безпечного відображення форматованого контенту в навчально-інформаційній системі розроблено спеціалізований компонент SafeMarkdownView. Цей компонент дозволяє користувачам

використовувати синтаксис Markdown для створення структурованого тексту з підтримкою різноманітних елементів форматування, зберігаючи при цьому високий рівень безпеки для запобігання XSS-атакам та іншим вразливостям [19].

Основними функціональними можливостями компонента є:

— підтримка розширеного синтаксису Markdown з таблицями, списками та іншими елементами через GitHub Flavored Markdown (GFM);

— безпечне відображення HTML-контенту з санітизацією для запобігання ін'єкціям шкідливого коду;

— підсвічування синтаксису для блоків коду з підтримкою понад 25 мов програмування;

— адаптивне відображення контенту з урахуванням режиму (світлого чи темного);

— стилізація всіх елементів Markdown для узгодженого дизайну з загальною темою додатку.

Компонент SafeMarkdownView реалізовано з використанням бібліотеки react-markdown для парсингу та рендерингу Markdown-контенту, DOMPurify для санітизації HTML та syntax-highlighter для підсвічування синтаксису коду.

Важливим аспектом безпеки в компоненті є використання бібліотеки DOMPurify для санітизації HTML-контенту, що дозволяє запобігти можливим XSS-атакам при відображенні користувацького контенту. Функція DOMPurify.sanitize() видаляє потенційно небезпечні елементи та атрибути з HTML, залишаючи лише безпечні.

Для підсвічування синтаксису використовується компонент SyntaxHighlighter з бібліотеки react-syntax-highlighter, який підтримує широкий спектр мов програмування. Для зручності використання створено мапування скорочених назв мов до їх повних ідентифікаторів, що дозволяє користувачам використовувати звичні скорочення при вказанні мови в блоках коду.

Особливістю реалізації є підтримка двох режимів відображення (світлого та темного) з відповідною адаптацією теми підсвічування синтаксису, що забезпечує

комфортну роботу з контентом у різних умовах освітлення та узгоджується із загальною темою додатку.

Для забезпечення узгодженого стилю всіх елементів Markdown створено окремий CSS-файл (markdown-styles.css), який містить стилізацію для заголовків, списків, таблиць, блоків коду та інших елементів відповідно до загального дизайну додатку.

Для практичного застосування компонента SafeMarkdownView розглянемо його використання в модальному вікні створення публікації. У наведеному прикладі компонент SafeMarkdownView використовується на другому кроці майстра створення публікації для попереднього перегляду контенту, введеного користувачем у форматі Markdown. Такий підхід дозволяє користувачам бачити, як буде виглядати їхній контент після публікації, та вносити корективи перед остаточним збереженням (рисунок 4.11 та 12).

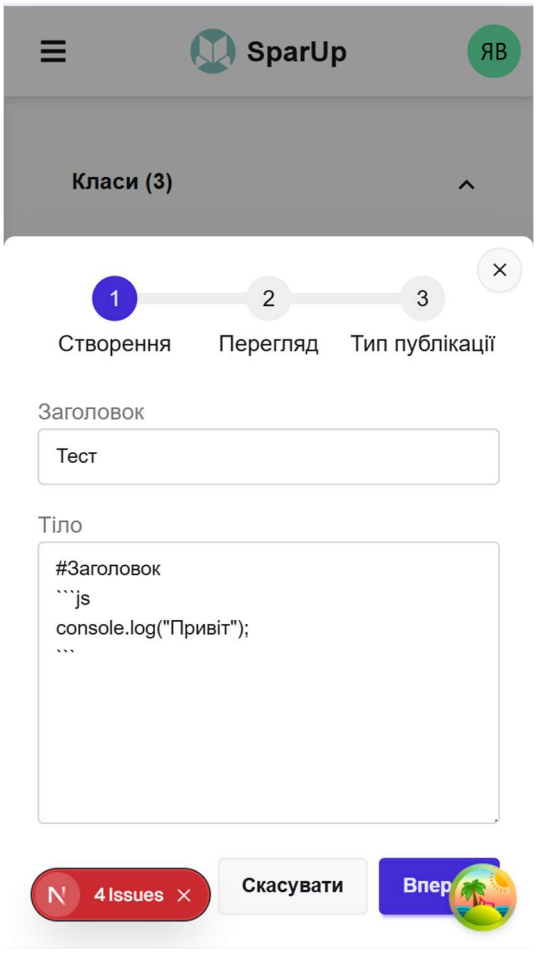


Рисунок 4.11 — Створення нового завдання та заповнення тексту

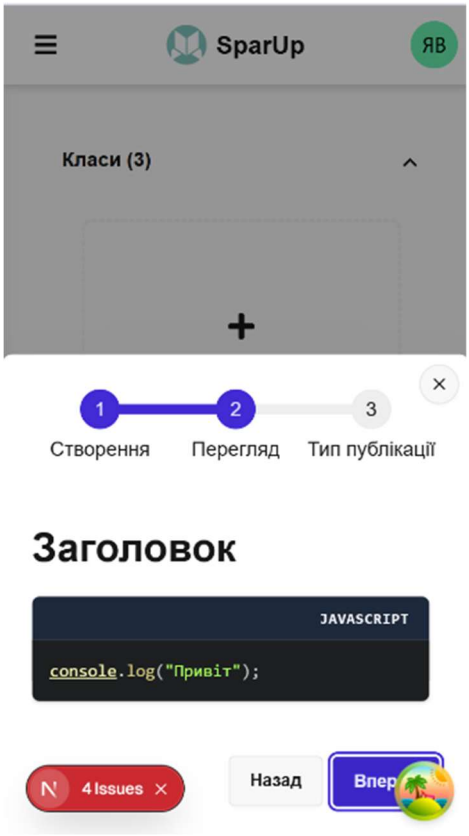


Рисунок 4.12 — Перегляд створеного завдання

Використання компонента SafeMarkdownView забезпечує багаті можливості форматування тексту для користувачів, зберігаючи при цьому високий рівень безпеки. Підтримка підсвічування синтаксису для блоків коду є особливо важливою для навчально-інформаційної системи, оскільки дозволяє викладачам включати приклади програмного коду в матеріали уроків та завдань у зручному для читання форматі.

4.11 Тестування клієнтської частини

4.11.1 Тестування процесу автентифікації

Першим етапом тестування була перевірка функціональності автентифікації. Система підтримує два способи входу: традиційний через електронну пошту і пароль та вхід через Google OAuth, що надає користувачам гнучкість у виборі методу автентифікації. При тестуванні традиційного входу перевірено як успішний сценарій з коректними обліковими даними, так і сценарій з помилковими даними.

Сторінка входу містить форму введення електронної пошти та пароля, а також кнопку для входу через Google. Інтерфейс інтуїтивно зрозумілий, з чіткими підказками та інформативними повідомленнями про помилки. При введенні коректних облікових даних система успішно автентифікує користувача (рисунок 4.13) та перенаправляє на головну сторінку додатку, де відображаються доступні навчальні матеріали та функції відповідно до ролі користувача (рисунок 4.14).

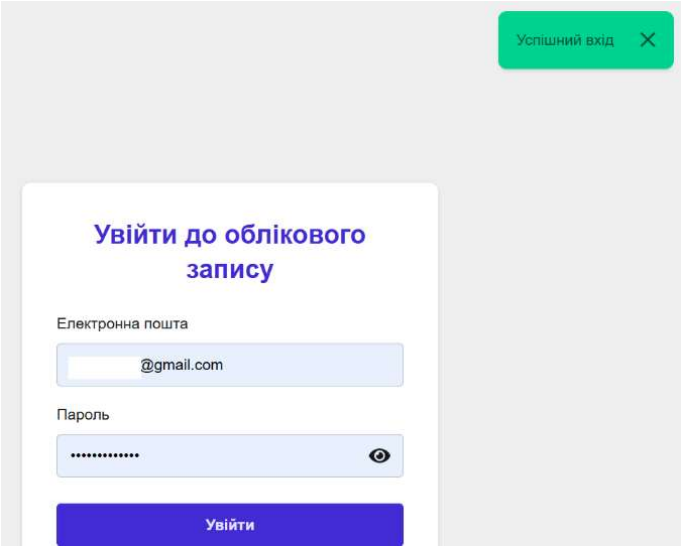


Рисунок 4.13 — Успішний вхід в систему

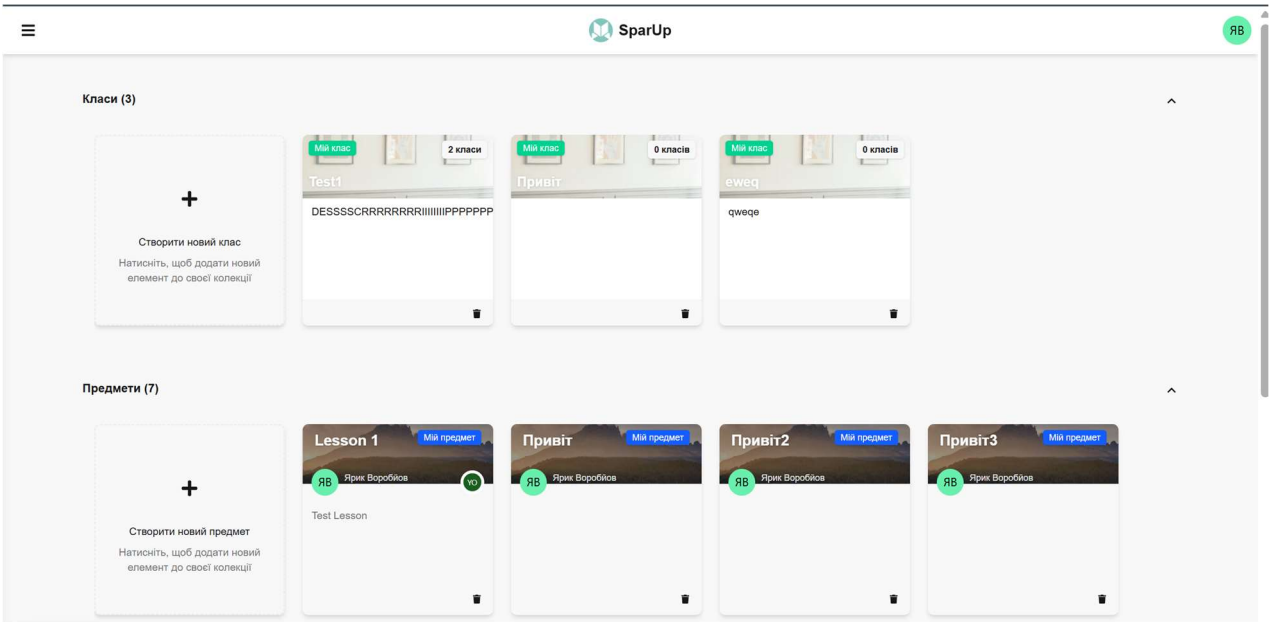


Рисунок 4.14 — Стартова сторінка додатку

При введенні некоректних облікових даних система відображає інформативне повідомлення про помилку, що дозволяє користувачу зрозуміти причину невдалого входу. Повідомлення про помилку має чіткий візуальний стиль, що привертає увагу, але не є нав'язливим. Користувач залишається на сторінці входу з можливістю виправити помилку та повторити спробу автентифікації (рисунок 4.15).

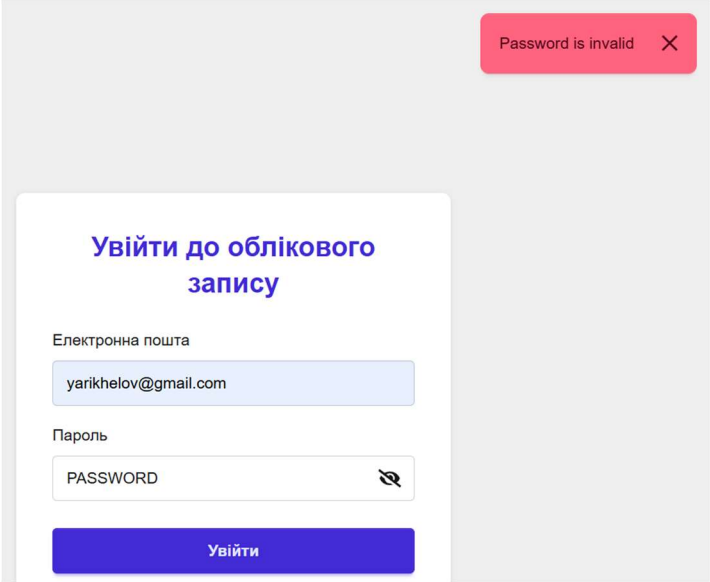


Рисунок 4.15 — Виведення помилки про некоректний пароль

Вхід через Google OAuth реалізовано з використанням спливаючого вікна, що забезпечує безпечний процес автентифікації без перенаправлення користувача з основного додатку (рисунок 3.7). Після успішної автентифікації через Google OAuth система отримує необхідні токени та перенаправляє користувача на головну сторінку додатку. Тестування показало надійну роботу механізму автентифікації через Google, включаючи правильну обробку різних сценаріїв, таких як скасування автентифікації користувачем.

4.11.2 Тестування створення та перегляду навчальних матеріалів

Важливим аспектом функціональності навчально-інформаційної системи є можливість створення та перегляду навчальних матеріалів. Тестування цих

функцій включало перевірку створення нових класів, уроків та завдань, а також перегляд створених матеріалів різними категоріями користувачів.

Для створення нового класу користувач з роллю викладача може використати відповідну кнопку на головній сторінці, що відкриває модальне вікно з формою. Форма містить поля для введення назви та опису класу, а також інших необхідних параметрів. Інтерфейс форми зрозумілий і забезпечує валідацію введених даних, запобігаючи створенню класів з неповною інформацією (рисунок 4.16 та 4.17).

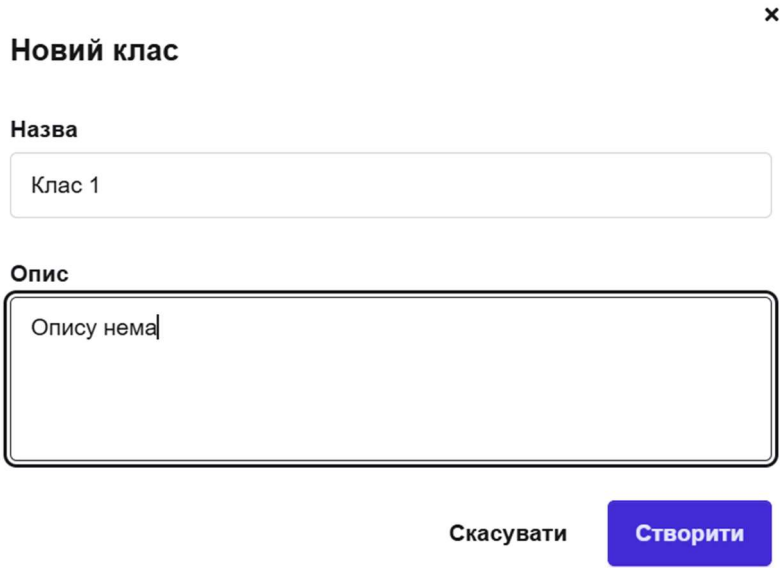


Рисунок 4.16 — Створення нового класу

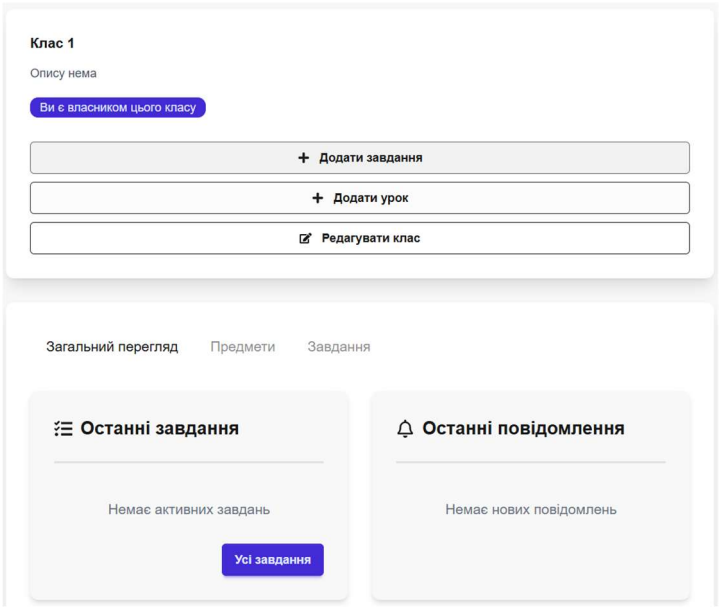


Рисунок 4.17 — Створений клас

Для створення завдання викладач може використати відповідну кнопку на сторінці класу, що відкриває модальне вікно з формою. Форма дозволяє ввести назву, опис та основний контент уроку у форматі Markdown. Використання Markdown забезпечує гнучкість у форматуванні тексту, вставці зображень, створенні таблиць та блоків коду, що особливо корисно для навчальних матеріалів технічного характеру (рисунок 4.11, 4.12 та 4.18).

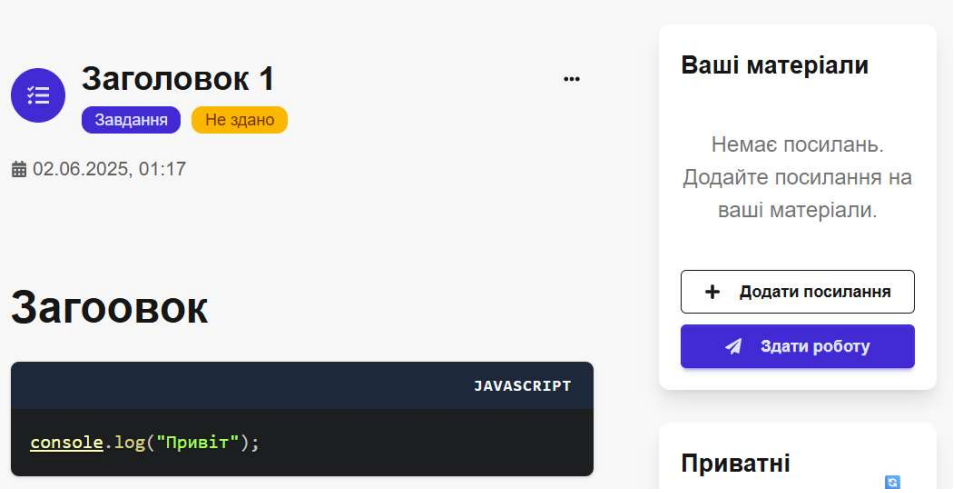


Рисунок 4.18 — Створене завдання

Тестування створення завдань показало коректну роботу відповідного модального вікна, де викладач може вказати назву, опис, термін виконання та максимальну оцінку. Особливістю системи є можливість створення різних типів завдань, включаючи завдання з автоматичною перевіркою, завдання з файловими вкладеннями та текстові завдання з ручною перевіркою.

4.11.3 Тестування системи комунікації в реальному часі

Особливу увагу приділено тестуванню системи комунікації в реальному часі, реалізованої на основі технології SignalR. Ця система забезпечує інтерактивну взаємодію між викладачами та студентами в контексті навчальних завдань, дозволяючи обмінюватися повідомленнями без необхідності оновлення сторінки.

Інтерфейс чату інтегровано в сторінку завдання, що забезпечує контекстну комунікацію безпосередньо в процесі роботи над завданням. Чат має зручний дизайн з розділенням повідомлень за авторством та відображенням часу

надсилання. Аватари користувачів додають персоналізації та допомагають швидко ідентифікувати авторів повідомлень (рисунок 4.19).

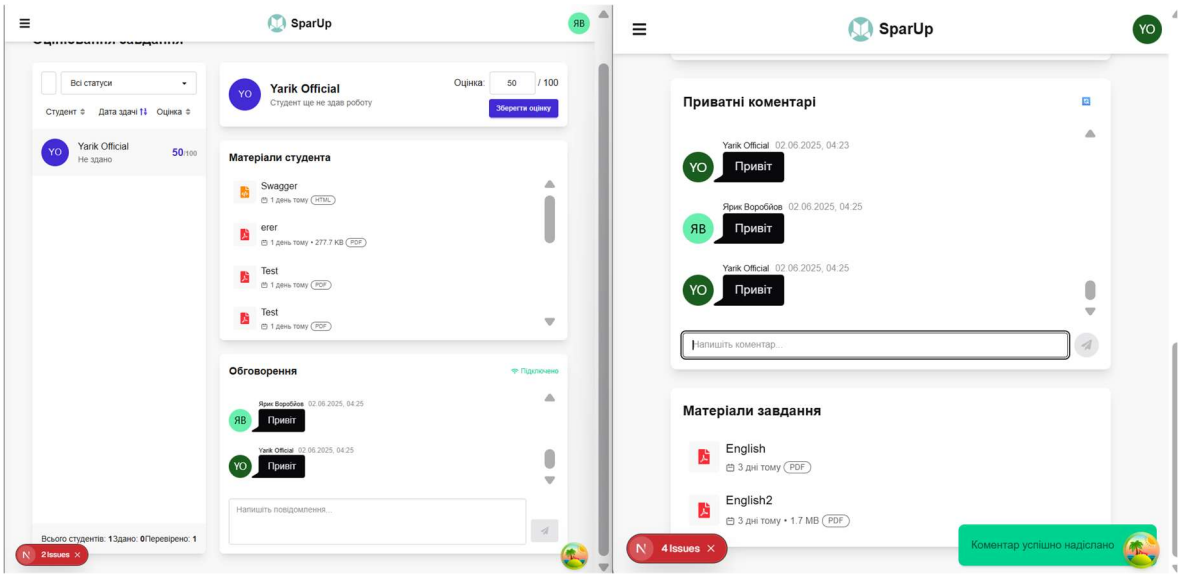


Рисунок 4.19 — Тестування відправлення повідомлень в режимі реального часу

Тестування надсилання повідомлень показало миттєву реакцію інтерфейсу — нове повідомлення відображається одразу після натискання кнопки надсилання, без помітної затримки. Система автоматично прокручує чат до останнього повідомлення, забезпечуючи зручність при активній комунікації. Поле введення підтримує відправку повідомлень через клавішу Enter, що відповідає звичним патернам взаємодії з чатами (рисунок 4.19).

4.11.4 Тестування адаптивності інтерфейсу

У сучасних умовах доступ до навчальних систем часто здійснюється з різних пристроїв, тому особливу увагу приділено тестуванню адаптивності інтерфейсу для різних розмірів екранів — від мобільних телефонів до настільних комп'ютерів.

На мобільних телефонах інтерфейс системи адаптується, забезпечуючи зручний доступ до всіх функцій без необхідності горизонтального прокручування. Елементи управління збільшуються для зручності використання на сенсорних екранах, а бічна панель навігації трансформується в випадне меню, доступне через кнопку в верхньому кутку екрану (рисунок 4.20).

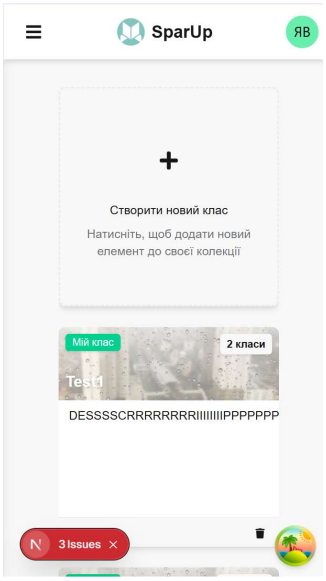


Рисунок 4.20 — Перегляд в режимі смартфона

На планшетах система використовує додатковий простір екрану для відображення більшої кількості інформації. Карточки навчальних матеріалів розташовуються в кілька колонок, що дозволяє бачити більше матеріалів без прокручування. Бічна панель навігації може бути зафіксована або схована, залежно від доступного простору, що забезпечує баланс між доступністю функцій та ефективним використанням екрану (рисунок 4.21).

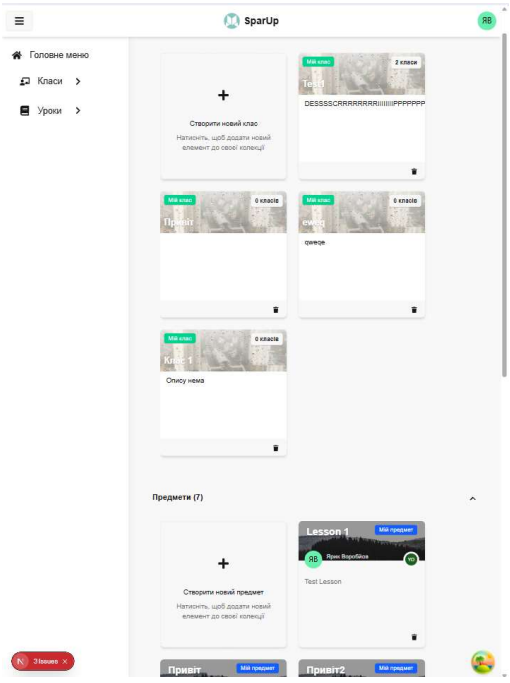


Рисунок 4.20 — Перегляд у режимі планшета

На настільних комп'ютерах система повноцінно використовує великий простір екрану, відображаючи бічну панель навігації, основний контент та додаткові елементи управління одночасно. Це забезпечує швидкий доступ до всіх функцій без необхідності перемикання між різними представленнями (рисунок 4.21)

Проведене практичне тестування клієнтської частини навчально-інформаційної системи дозволяє зробити висновок про високу якість реалізації та відповідність встановленим вимогам. Основні функціональні блоки системи, включаючи автентифікацію, управління навчальними матеріалами, комунікацію в реальному часі та відображення Markdown-контенту, працюють стабільно та забезпечують зручний користувацький досвід.

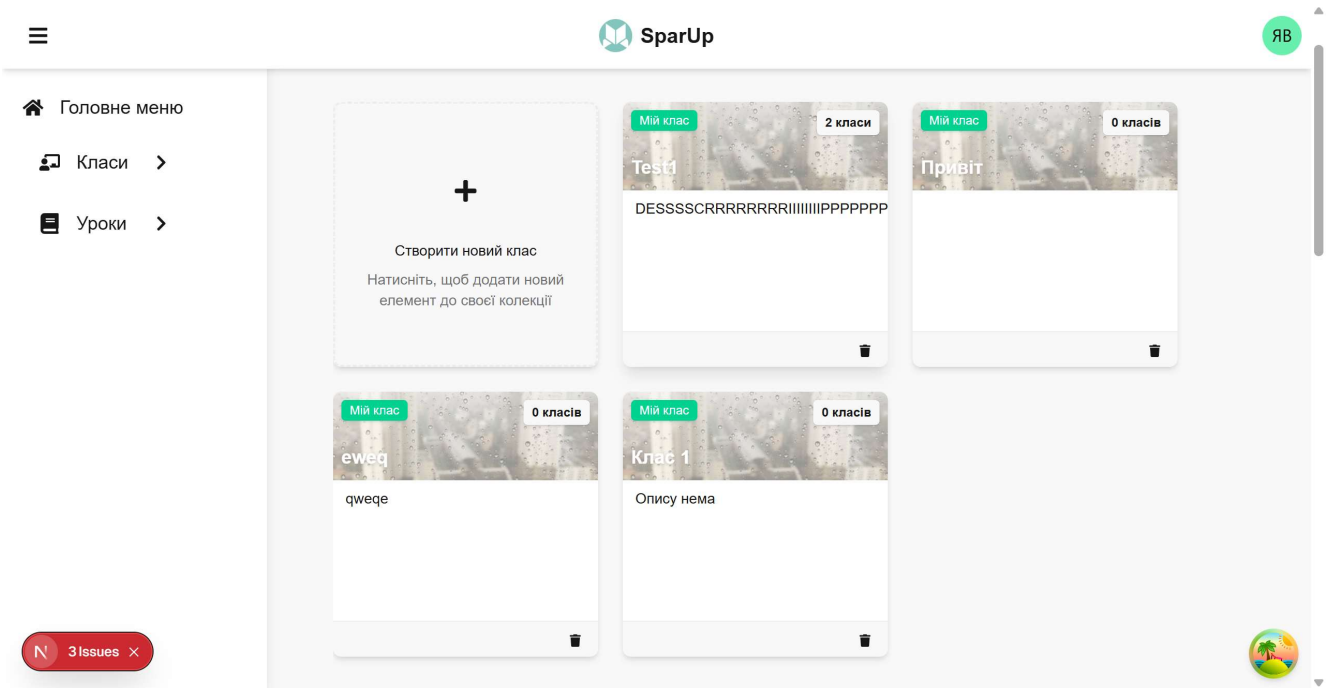


Рисунок 4.21 — Перегляд у режимі настільного комп’ютера

Адаптивний дизайн системи забезпечує комфортне використання на різних пристроях, що особливо важливо в сучасних умовах, коли доступ до навчальних ресурсів часто здійснюється з мобільних пристроїв. Інтеграція з зовнішніми сервісами, такими як Google OAuth та SignalR, реалізована надійно та забезпечує розширені можливості системи без ускладнення користувацького інтерфейсу.

У процесі тестування виявлено та усунено низку дрібних проблем, що дозволило значно підвищити якість системи. Зокрема, оптимізовано продуктивність відображення великих Markdown-документів, покращено механізм відновлення з'єднання SignalR після розривів та вдосконалено адаптивність модальних вікон на малих екранах.

Результати тестування підтверджують, що клієнтська частина навчально-інформаційної системи відповідає сучасним стандартам веб-розробки, забезпечує надійну роботу основних функцій та створює позитивний користувацький досвід для всіх категорій користувачів.

ВИСНОВКИ

У ході виконання бакалаврської кваліфікаційної роботи розроблено навчально-інформаційну систему з використанням сучасних веб-технологій, що дозволяє автоматизувати процес організації та управління навчальною діяльністю в освітніх установах через ієрархічну модель класів та предметів.

Основними результатами, отриманими під час виконання роботи, стали:

— проведення аналізу існуючих навчально-інформаційних систем з виявленням їх переваг та недоліків для визначення ключових вимог до розроблюваної системи;

— обґрунтування необхідності створення нового рішення, яке поєднує сучасні веб-технології з ефективною організацією навчального процесу;

— вибір та застосування оптимального набору технологій: ASP.NET Core для серверної частини, Next.js з React для клієнтської частини, Entity Framework Core для роботи з базою даних;

— розробка цибулевої архітектури з чітким розділенням відповідальності між шарами, що забезпечує підтримуваність коду та незалежність бізнес-логіки від зовнішніх факторів.

Спроєктовано клієнт-серверну архітектуру з використанням RESTful API для взаємодії між компонентами системи. Розроблено систему автентифікації та авторизації з підтримкою Google OAuth2 та JWT токенів, що забезпечує безпечний доступ до ресурсів системи. Реалізовано повноцінний механізм управління навчальними матеріалами через ієрархічну структуру класів та предметів.

На стороні клієнта створено інтуїтивний користувацький інтерфейс з використанням React компонентів та Tailwind CSS, що забезпечує зручну взаємодію для всіх категорій користувачів. Впроваджено React Query для ефективного управління станом додатка та кешування даних, що покращує продуктивність системи.

Розроблена система має низку переваг над аналогами:

— веб-доступність без потреби встановлення додаткового ПЗ;

— підтримка різних ролей користувачів з відповідними правами доступу;

— ієрархічна організація навчальних матеріалів через систему класів та предметів;

— ефективна система оцінювання з можливістю комунікації між викладачами та студентами;

— масштабована архітектура з можливістю подальшого розширення функціоналу;

— підтримка різних систем управління базами даних (PostgreSQL для публікації, SQLite для розробки).

Практична цінність полягає у створенні функціонального програмного продукту, який може бути впроваджений в освітніх установах різного рівня для оптимізації навчального процесу, покращення комунікації між учасниками та автоматизації рутинних процесів управління завданнями та оцінюванням.

Подальші напрями розвитку системи можуть включати:

— інтеграцію з популярними LMS системами для розширення функціональних можливостей;

— впровадження real-time комунікації через WebSocket для миттєвих сповіщень;

— розробку мобільного додатка для зручного доступу з мобільних пристроїв;

— додавання аналітичних можливостей для відстеження прогресу студентів;

— впровадження системи календарного планування занять та дедлайнів;

— розширення можливостей для групової роботи студентів над проєктами;

— інтеграцію з зовнішніми сервісами для перевірки на плагіат;

— додавання підтримки відеоконференцій та онлайн-занять.

Розроблена система успішно вирішує поставлені задачі та формує надійну основу для подальшого розвитку сучасних навчально-інформаційних платформ в освітньому середовищі.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Навчально-інформаційна система з використанням C# та Next.JS. [Електронний ресурс] / Кисюк Д. В., Воробйов Я. І. // Матеріали Всеукраїнської науково-практичної інтернет-конференції «Молодь в науці: дослідження, проблеми, перспективи (МН-2025)», Вінниця, 16 червня 2025р. Режим доступу: https://conferences.vntu.edu.ua/index.php/mn/mn2025/paper/view/ 25271.

2. Як входити в обліковий запис за допомогою паролів додатків. URL: https://support.google.com/accounts/answer/185833?hl=uk (дата звернення 15.04.2025).

3. Використання OAuth 2.0 для доступу до Google APIs. URL: https://developers.google.com/identity/protocols/oauth2 (дата звернення 15.04.25).

4. Тазетдінов А. Next.js Cookbook: повне керівництво з розробки сучасних веб-додатків. Лондон: BPB Online, 2023. 270 с. ISBN 978-93-55518-453.

5. Фрімен А. Pro ASP.NET Core MVC 2. 7-е вид. Лондон: Apress, 2017. 1024 с.

6. Чінемалу А., Банкс А. Learning React: Modern Patterns for Developing React Apps. 2-е вид. Бостон : O'Reilly Media, 2020. 310 с.

7. Tailwind CSS документація. Tailwind Labs. 2024. URL: https://tailwindcss.com/docs (дата звернення: 20.04.2025).

8. Entity Framework Core документація. Microsoft. 2024. URL: https://docs.microsoft.com/en-us/ef/core/ (дата звернення: 22.04.2025).

9. The PostgreSQL Global Development Group. PostgreSQL 9.0.22: офіційна документація. Сан-Франциско, 2015. 3450 с.

10. JWT.io — Introduction to JSON Web Tokens. Auth0. 2024. URL: https://jwt.io/introduction/ (дата звернення: 28.04.2025).

11. TypeScript офіційна документація. Microsoft. 2024. URL: https://www.typescriptlang.org/docs/ (дата звернення: 28.04.2025).

12. DaisyUI компоненти документація. DaisyUI. 2024. URL: https://daisyui.com/docs/ (дата звернення: 28.04.2025).

13. Swagger OpenAPI документація. SmartBear Software. 2024. URL: https://swagger.io/docs/ (дата звернення: 28.04.2025).

14. Docker офіційна документація. Docker Inc. 2024. URL: https://docs.docker.com/ (дата звернення: 28.04.2025).

15. Стронько К. Фальшиве відчуття безпеки або найбільший недолік Next.js. DOU. 2024. URL: https://dou.ua/forums/topic/48621/ (дата звернення: 15.03.2025).

16. Nginx офіційна документація. F5 Networks. 2024. URL: https://nginx.org/en/docs/ (дата звернення: 28.04.2025).

17. SQLite документація. SQLite Development Team. 2024. URL: https://www.sqlite.org/docs.html (дата звернення: 02.05.20).

18. Стандарт REST API Best Practices. Mozilla Developer Network. 2024. URL: https://developer.mozilla.org/en-US/docs/Web/HTTP/Methods (дата звернення: 22.03.2025).

19. GitHub Actions документація. GitHub. 2024. URL: https://docs.github.com/en/actions (дата звернення: 25.03.2025).

20. Node.js офіційна документація. OpenJS Foundation. 2024. URL: https://nodejs.org/en/docs/ (дата звернення: 28.03.2025).

21. CORS (Cross-Origin Resource Sharing) специфікація. W3C. 2024. URL: https://www.w3.org/TR/cors/ (дата звернення: 30.03.2025).

22. Принципи SOLID в програмуванні. Clean Code Blog. 2024. URL: https://blog.cleancoder.com/uncle-bob/2020/10/18/Solid-Relevance.html (дата звернення: 02.04.2025).

23. Архітектурні патерни веб-додатків. Martin Fowler. 2024. URL: https://martinfowler.com/architecture/ (дата звернення: 05.04.2025).

24. HTTP/HTTPS протоколи специфікація. IETF. 2024. URL: https://tools.ietf.org/html/rfc7540 (дата звернення: 08.04.2025).

25. Сучасні підходи до веб-розробки. MDN Web Docs. 2024. URL: https://developer.mozilla.org/en-US/docs/Learn (дата звернення: 10.04.2025).

26. Безпека веб-додатків. OWASP Foundation. 2024. URL: https://owasp.org/www-project-top-ten/ (дата звернення: 12.04.2025).

ДОДАТОК А

Технічне завдання

Міністерство освіти і науки України
Вінницький національний технічний університет
Факультет інформаційних технологій та комп'ютерної інженерії
Кафедра обчислювальної техніки

ЗАТВЕРДЖУЮ
Завідувач кафедри ОТ
д.т.н., Азаров О.Д.
21 березня 2025 р.

ТЕХНІЧНЕ ЗАВДАННЯ
на виконання бакалаврської кваліфікаційної роботи
«Навчально-інформаційна система з використанням C# та Next.JS»

08-54.БКР.023.00.000 ТЗ

Науковий керівник: ст. викл. каф. ОТ
_____Кисюк Д. В.

Студент групи 2КІ-23мс
_____Воробйов Я. І.

м. Вінниця — 2025

1 Підстава для використання бакалаврської кваліфікаційної роботи (БКР)

1.1 Актуальність розробки полягає у необхідності впровадження сучасних навчально-інформаційних систем у освітніх установах через зростання обсягів навчальної інформації та великої кількості користувачів. Розробка веб-орієнтованої навчально-інформаційної системи дасть можливість оптимізувати управління освітнім процесом, підвищити якість взаємодії між викладачами та студентами та забезпечити високий рівень доступності навчальних матеріалів.

1.2 Головним завданням розробки є створення сучасної, масштабованої та ефективної навчально-інформаційної системи з використанням технологій ASP.NET Core для бекенду та Next.js для фронтенду, системи автентифікації та авторизації, а також ієрархічної організації навчальних матеріалів.

1.3 Наказ про затвердження теми бакалаврської дипломної роботи від 21 березня 2025 р. №97.

2 Мета і призначення БКР

2.1 Мета проєкту полягає у створенні навчально-інформаційної системи з метою забезпечення ефективної організації навчального процесу через розробку клієнт-серверної архітектури з використанням ASP.NET Core та Next.js. Система повинна підтримувати сучасні механізми автентифікації з використанням JWT токенів, а також інтеграцію з Google через OAuth2, що дозволить реалізувати безпечний і зручний доступ для користувачів. У межах проєкту передбачається створення ієрархічної моделі організації навчального середовища, зокрема структури класів і предметів, що дозволить ефективно керувати навчальними даними. Крім того, розробляються інтерфейси для адміністрування навчальних завдань, виставлення оцінок та забезпечення взаємодії між викладачами й студентами.

2.2 Призначення розробки полягає у виконанні бакалаврської дипломної роботи, за результатами якої буде створено функціональний програмний продукт для автоматизації навчального процесу в освітніх установах.

3 Вихідні дані для виконання БКР

3.1 Аналіз існуючих навчально-інформаційних систем та платформ, включно з їх функціональними можливостями, архітектурними рішеннями та технологічними стеками.

3.2 Огляд і аналіз сучасних веб-технологій: фреймворків ASP.NET Core та Next.js, систем управління базами даних, технологій автентифікації та авторизації, архітектурних патернів веб-застосунків.

3.3 Визначення вимог до функціональності системи, включаючи управління користувачами, організацію класів та предметів, створення та розподіл завдань, систему оцінювання та комунікації між учасниками.

3.4 Планування архітектури системи, включаючи структуру бази даних, API кінцевих точок, компоненти користувацького інтерфейсу та додавання інтеграційних рішень.

3.5 Оцінка технологічних рішень і вибір оптимального стеку технологій для забезпечення продуктивності, масштабованості та підтримуваності системи.

3.6 Розробка плану тестування та впровадження системи з урахуванням різних типів користувачів та сценаріїв використання.

4 Вимоги до виконання БКР

Головна вимога полягає в створенні функціональної навчально-інформаційної системи, яка забезпечить:

— ієрархічну організацію навчальних матеріалів через систему класів та предметів;

— надійну систему автентифікації та авторизації з підтримкою різних ролей користувачів;

— ефективне управління завданнями та процесом їх виконання студентами;

— зручний інтерфейс для оцінювання робіт викладачами;

— масштабовану архітектуру з можливістю подальшого розвитку;

— сучасний та інтуїтивний користувацький інтерфейс.

5 Етапи БКР та очікувані результати

Етапи розробки БКР та очікувані результати наведено у Таблиці А.1

Таблиця А.1 — Етапи БКР

№ етапу	Назва етапу	Термін виконання		Очікувані результати
		Початок	Кінець	
1	Вибір, узгодження та затвердження теми БКР	21.03.25	21.03.25	Розділ 1
2	Аналіз існуючих програмних рішень та визначення вимог системи	22.03.25	23.03.25	Розділ 2
3	Проєктування архітектури системи та бази даних	24.03.25	28.03.25	Розділ 3
4	Розробка бекенду на ASP.NET Core та Entity Framework	29.03.25	13.04.25	Розділ 3
5	Реалізація системи автентифікації та авторизації з JWT токенами та OAuth2 Google	14.04.25	20.04.25	Розділ 3
6	Реалізація API та налаштування документації	21.04.25	22.04.25	Розділ 3
7	Розробка фронтенду на Next.js з використанням Tailwind CSS та DaisyUI	23.04.25	27.04.25	Розділ 4
8	Інтеграція та тестування системи	28.04.25	02.05.25	Розділ 5
9	Підготовка тезових матеріалів для публікації	03.05.25	04.05.25	Тези
10	Написання документації та пояснювальної записки	05.05.25	12.05.25	ПЗ, презентація
11	Підготовка супровідної документації, перевірка відповідності нормам та проходження тесту на плагіат	13.05.25	13.05.25	ПЗ, презентація

6 Матеріали, що подаються до захисту БКР

До захисту подаються: пояснювальна записка БКР, ілюстративні матеріали (діаграми архітектури системи, схеми бази даних, скріншоти інтерфейсів), протокол попереднього захисту БКР на кафедрі, відгук наукового керівника,

анотації до БКР українською та іноземною мовами, довідка про відповідність оформлення БКР чинним вимогам.

7 Порядок контролю виконання та захисту БКР

Виконання етапів графічної та розрахункової документації БКР контролюється науковим керівником ст. викл. каф. ОТ Кисюк Д.В. згідно зі встановленими термінами. Захист БКР відбувається на засіданні Екзаменаційної комісії, затвердженої наказом ректора ВНТУ.

8 Вимоги до оформлювання та порядок виконання БКР

8.1 При оформлюванні БКР використовуються:

— ДСТУ 3008 : 2015 «Звіти в сфері науки і техніки. Структура та правила оформлювання»;

— ДСТУ 8302 : 2015 «Бібліографічні посилання. Загальні положення та правила складання»;

— методичні вказівки до виконання бакалаврських кваліфікаційних робіт для студентів спеціальності 123 «Комп'ютерна інженерія» (освітня програма «Комп'ютерна інженерія») — кафедра обчислювальної техніки, ВНТУ 2023.

8.2 Порядок виконання БКР викладено в «Положення про кваліфікаційні роботи на першому (бакалаврському) рівні вищої освіти СУЯ ВНТУ-03.02.02-П.001.01:21»

ДОДАТОК Б

ПРОТОКОЛ ПЕРЕВІРКИ КВАЛІФІКАЦІЙНОЇ РОБОТИ НА НАЯВНІСТЬ ТЕКСТОВИХ ЗАПОЗИЧЕНЬ

Назва роботи:
Навчально-інформаційна система з використанням C# та Next.JS

Тип роботи: Бакалаврська кваліфікаційна робота
(БДР, МКР)

Підрозділ кафедра обчислювальної техніки
(кафедра, факультет)

Показники звіту подібності StrikePlagiarism

Оригінальність 98,5% Схожість 1,5%

Аналіз звіту подібності (відмітити потрібне):

✓ Запозичення, виявлення у роботі, оформлені коректно і не містять ознак плагіату.

☐ Виявлені у роботі запозичення не мають ознак плагіату, але їх надмірна кількість викликає сумніви щодо цінності роботи і відсутності самостійності її виконання автором. Роботу направити на розгляд експертної комісії кафедри.

☐ Виявлені у роботі запозичення є недобросовісними і мають ознаки плагіату та/або в ній містяться навмисні спотворення тексту, що вказують на спроби приховування недобросовісних запозичень.

Особа, відповідальна за перевірку _____ Захарченко С.М.
(підпис) (прізвище, ініціали)

Ознайомлені з повним звітом подібності, який був згенерований системою StrikePlagiarism щодо роботи.

Автор роботи _____ Воробйов М. І.
(підпис) (прізвище, ініціали)

Керівник роботи _____ Кисюк Д. В.
(підпис) (прізвище, ініціали)

ДОДАТОК В

ІЛЮСТРАТИВНА ЧАСТИНА

НАВЧАЛЬНО-ІНФОРМАЦІЙНА СИСТЕМА З ВИКОРИСТАННЯМ C# ТА NEXT.JS

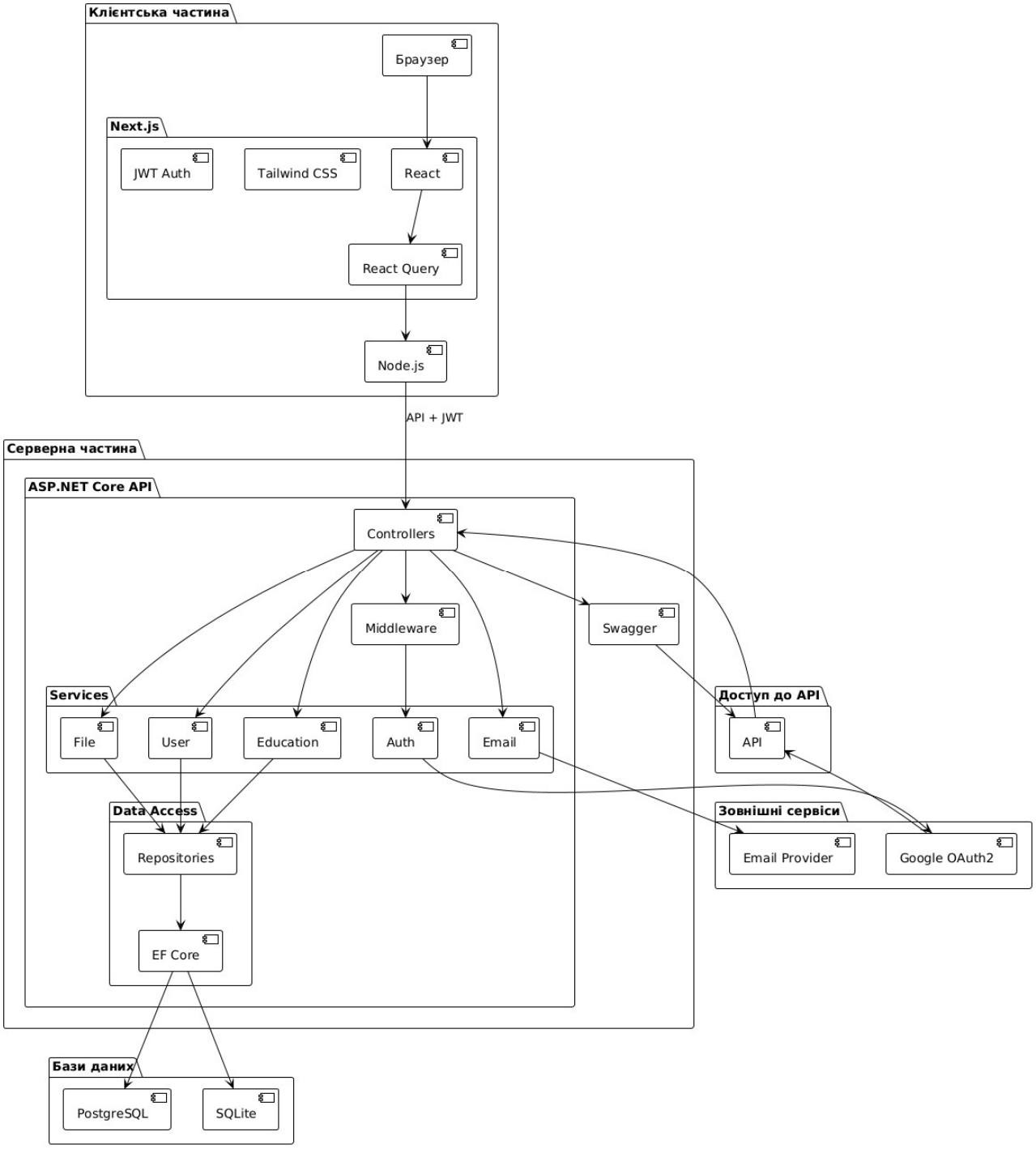


Рисунок В.1 — Архітектура проєкту

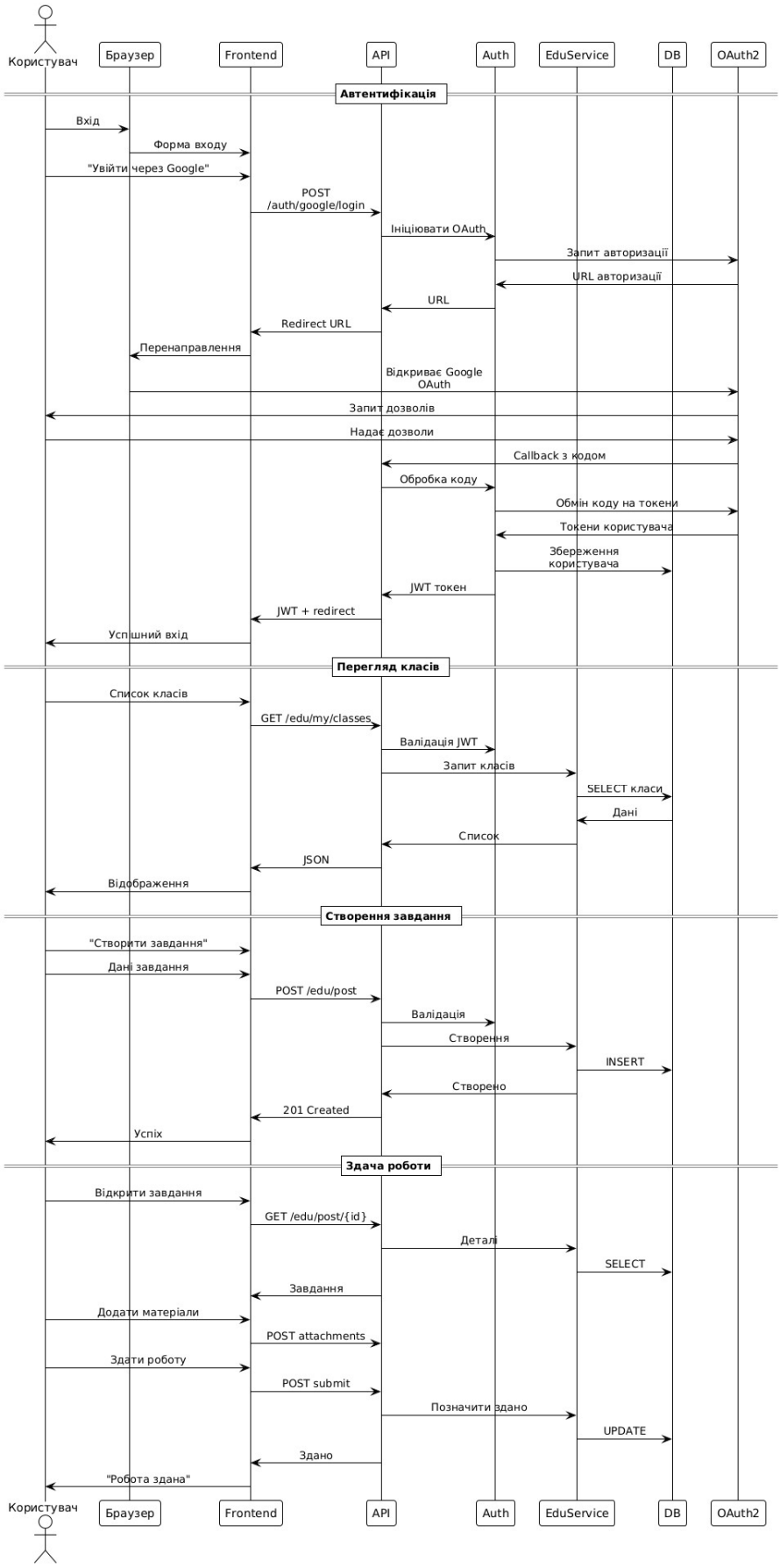


Рисунок В.2 — Діаграма взаємодії навчально-інформаційної системи

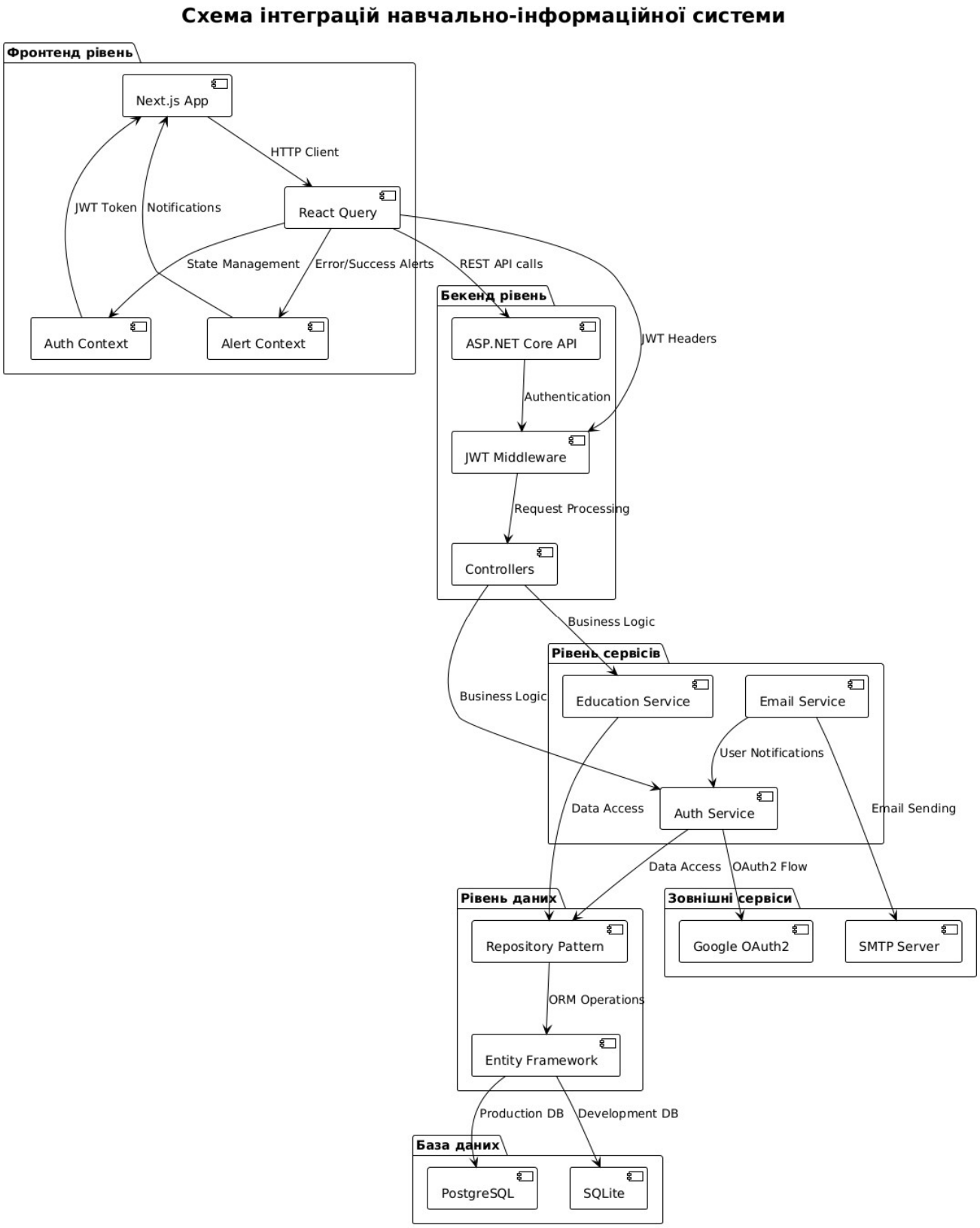


Рисунок В.3 — Схема інтеграції навчально-інформаційної системи

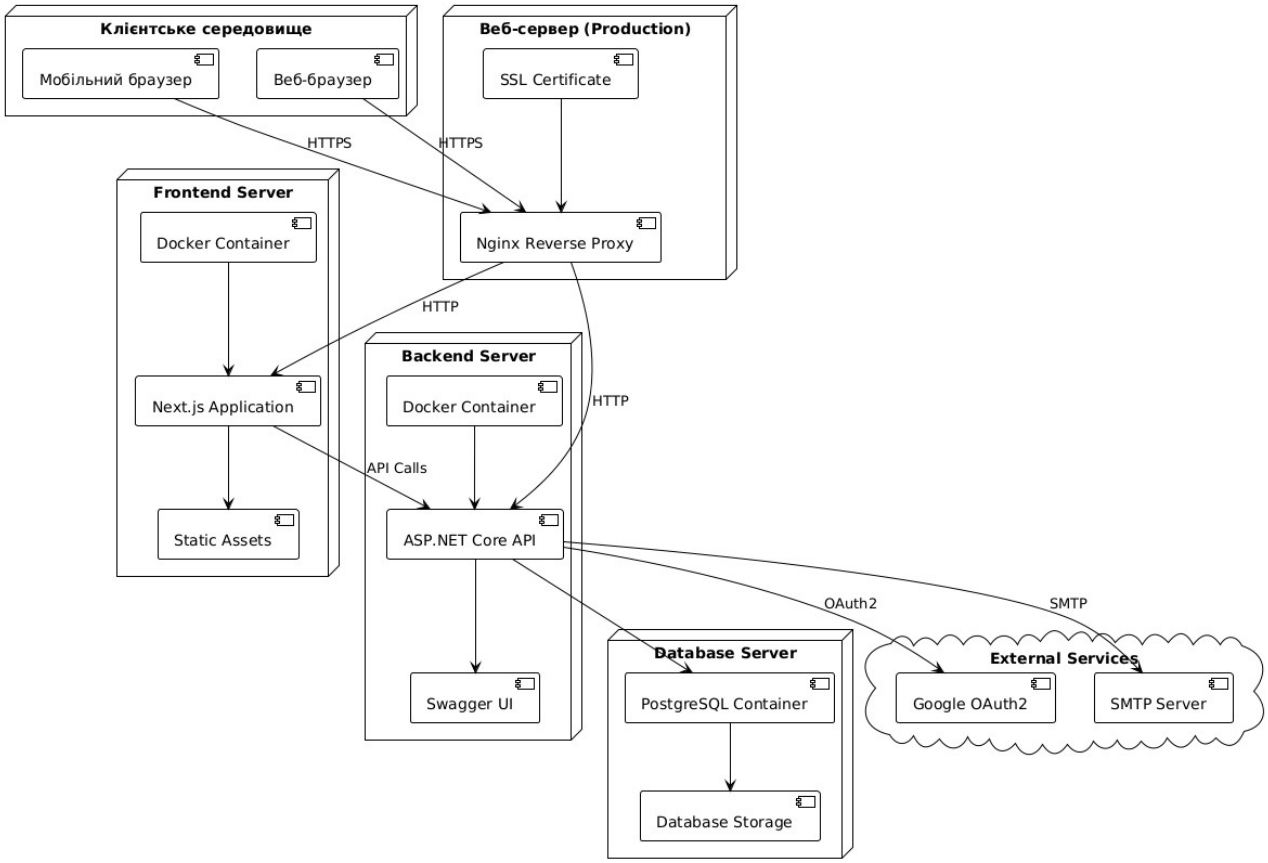


Рисунок В.4 — Діаграма розгортання навчально інформаційної системи

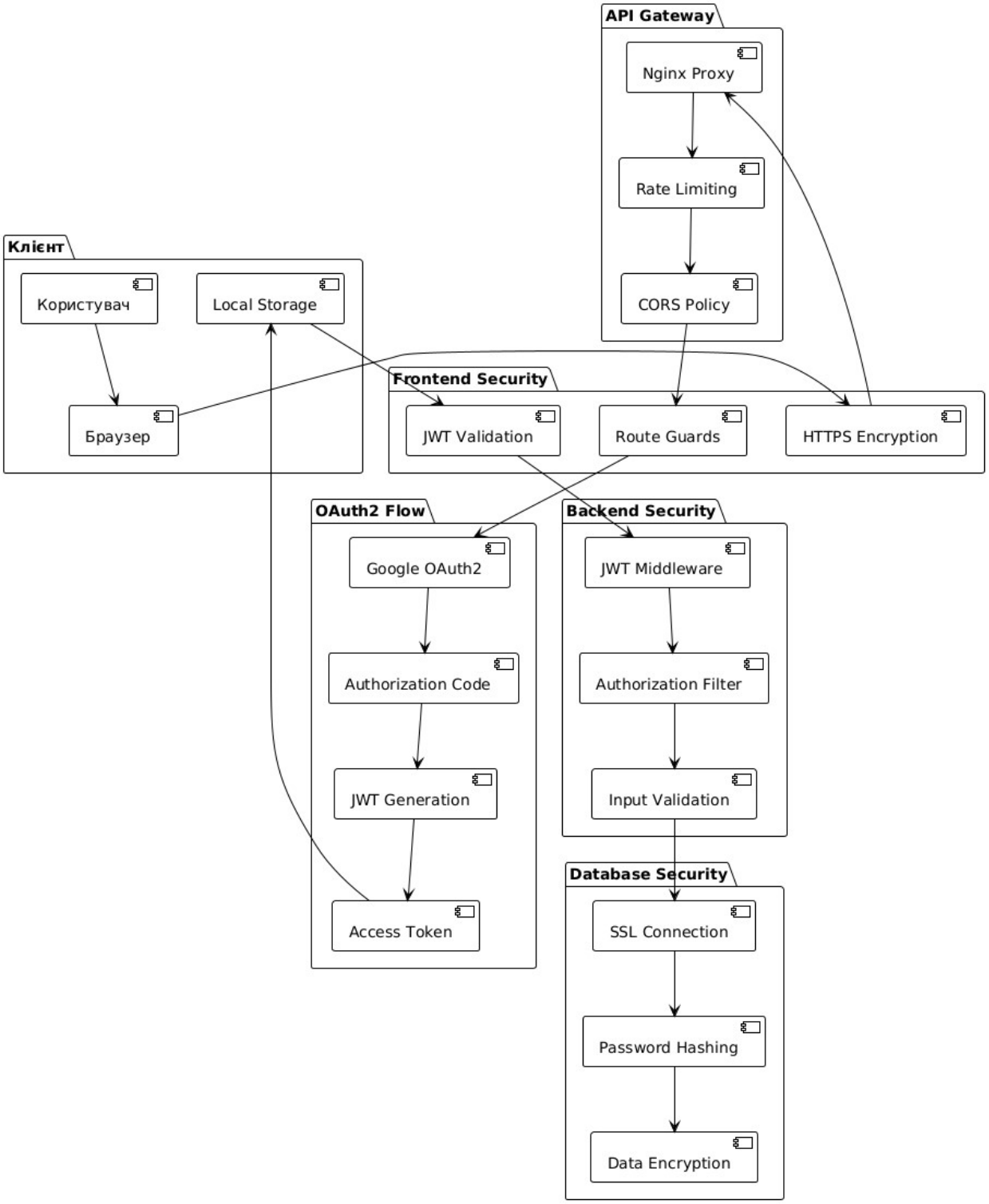


Рисунок В.5 — Схема безпеки навчально-інформаційної системи

ДОДАТОК Г

Лістинги програми

Лістинг Г.1 — Клас EduContainer

```
public abstract class EduContainer
{
    [Key]
    public Guid Id { get; set; }
    [Required]
    [MaxLength(100)]
    public string Name { get; set; }
    [MaxLength(500)]
    public string Description { get; set; }
    [Required]
    public ApplicationUser Owner { get; set; }
    [Required]
    public Guid OwnerId { get; set; }
    [Required]
    public DateTime Created { get; set; }
    [Required]
    [MaxLength(200)]
    public string Url { get; set; }
}
```

Лістинг Г.2 — Клас EduClass

```
public class EduClass : EduContainer
{
    public ICollection<EduSubject> Subjects { get; set; }
}
```

Лістинг Г.3 — Клас EduImportantType

```
public enum EduImportantType
{
    [EnumMember(Value = "Internal")]
    Internal = 0,
    [EnumMember(Value = "NonImportant")]
    NonImportant = 1,
    [EnumMember(Value = "Important")]
    Important = 2,
}
```

Лістинг Г.4 — Клас EduMaterialInfo

```
public class EduMaterialInfo
{
    [Key]
    public Guid Id { get; set; }
    [Required]
```

```
    public Guid OwnerId { get; set; }
    public ApplicationUser Owner { get; set; }
    [Required]
    [MaxLength(200)]
    public string MaterialName { get; set; }
    [Required]
    public DateTime Created { get; set; }
    public DateTime Modified { get; set; }
    [Required]
    [MaxLength(500)]
    public string Url { get; set; }
    public MaterialProtection Protection { get; set; }
}
```

Лістинг Г.5 — Клас EduNotification

```
public class EduNotification : EduPost
{
    public EduImportantType ImportantType { get; set; }
}
```

Лістинг Г.6 — Клас EduPairSubjectAndUser

```
public class EduPairSubjectAndUser
{
    [Key]
    public Guid Id { get; set; }
    [Required]
    public Guid UserId { get; set; }
    public ApplicationUser User { get; set; }
    [Required]
    public Guid EduSubjectId { get; set; }
    public EduSubject EduSubject { get; set; }
}
```

Лістинг Г.7 — Клас EduPairContainerAndPost

```
public class EduPairContainerAndPost
{
    [Key]
    public Guid Id { get; set; }
    [Required]
    public Guid ContainerId { get; set; }
    public EduContainer Container { get; set; }
    [Required]
    public Guid EduPostId { get; set; }
    public EduPost EduPost { get; set; }
}
```

Лістинг Г.8 — Клас EduPost

```
public abstract class EduPost
{
```

```
    [Key]
    public Guid Id { get; set; }
    [Required]
    [MaxLength(200)]
    public string Title { get; set; }
    [Required]
    public DateTime Created { get; set; }
    [MaxLength(2000)]
    public string Body { get; set; }
    [Required]
    public Guid CreatedById { get; set; }
    public ApplicationUser CreatedBy { get; set; }
}
```

Лістинг Г.9 — Клас EduSubject

```
public class EduSubject : EduContainer
{
    [MaxLength(20)]
    public string? RoomNumber { get; set; }
    public Guid? ClassId { get; set; }
    public EduClass? Class { get; set; }
    public ICollection<ApplicationUser> Users { get; set; }
}
```

Лістинг Г.10 — Клас EduTask

```
public class EduTask : EduPost
{
    public DateTime? DueDate { get; set; }
    public int? MaxGrade { get; set; }
}
```

Лістинг Г.11 — Клас EduTaskResult

```
public class EduTaskResult
{
    [Key]    public Guid Id { get; set; }
    [Required]    public Guid TaskId { get; set; }
    public EduTask Task { get; set; }
    [Required]    public Guid CompletedById { get; set; }
    public ApplicationUser CompletedBy { get; set; }
    public int? Grade { get; set; }
    public DateTime? Completed { get; set; }
    public DateTime? Checked { get; set; }
    public Guid? CheckedById { get; set; }
    public ApplicationUser CheckedBy { get; set; }
    public ICollection<EduTaskResultItem> Items { get; set; }
}
```

Лістинг Г.12 — Клас EduTaskResultItem

```
public class EduTaskResultItem
```

```
{
   [Key]    public Guid Id { get; set; }
   [Required]    public Guid MaterialInfoId { get; set; }
   public EduMaterialInfo MaterialInfo { get; set; }
   [Required]    public Guid TaskResultId { get; set; }
   public EduTaskResult TaskResult { get; set; }
}
```

Лістинг Г.13 — Enum MaterialProtection

```
public enum MaterialProtection
{
   [EnumMember(Value = "Private")]    Private,
   [EnumMember(Value = "Public")]    Public,
}
```

Лістинг Г.14 — Реалізація ApplicationDbContext

```
public class ApplicationDbContext(DbContextOptions<ApplicationDbContext> options)
   : IdentityDbContext<ApplicationUser, ApplicationRole, Guid>(options)
{
   public DbSet<EduClass> EduClasses { get; set; }
   public DbSet<EduSubject> EduSubjects { get; set; }
   public DbSet<EduContainer> EduContainers { get; set; }
   public DbSet<EduMaterialInfo> EduMaterialInfos {get;set;}
   public DbSet<EduPost> EduPosts { get; set; }
   public DbSet<EduNotification> EduNotifications { get; set; }
   public DbSet<EduTask> EduTasks { get; set; }
      public DbSet<EduTaskResult> EduTaskResults { get; set; }
   public DbSet<EduTaskResultItem> EduTaskResultItems { get; set; }
      public DbSet<EduPairSubjectAndUser> EduPairSubjectAndUsers { get; set; }
      public DbSet<EduPairContainerAndPost> EduPairContainerAndPosts { get; set; }
      protected override void OnModelCreating(ModelBuilder builder)
   {
      builder.Entity<EduContainer>()
         .HasDiscriminator<string>("EduType")
         .HasValue<EduClass>("Class")
         .HasValue<EduSubject>("Subject");
      builder.Entity<EduPost>()
         .HasDiscriminator<string>("PostType")
         .HasValue<EduNotification>("Notification")
         .HasValue<EduTask>("Task");
      foreach (var entityType in builder.Model.GetEntityTypes())
      {
         foreach (var property in entityType.GetProperties())
         {
            if (property.ClrType.IsEnum)
            {
               builder.Entity(entityType.Name)
                  .Property(property.Name)
                  .HasConversion<int>();
            }
         }
```

```
        }

        base.OnModelCreating(builder);
    }
}
```

Лістинг Г.15 — Реалізація JwtTokenGenerator

```
public class JwtTokenGenerator{
    public const string SecretKey = "SecretKey12341234GoodClass";
    private readonly SigningCredentials _credentials;
    public JwtTokenGenerator()    {
        var key = new SymmetricSecurityKey(Encoding.UTF8.GetBytes(SecretKey.Substring(0,
32)));
        _credentials = new SigningCredentials(key, SecurityAlgorithms.HmacSha256);
    }
    private string GenerateToken(IEnumerable<Claim> claims, TimeSpan lifetime, string
tokenType)    {
        var claimsList = new List<Claim>(claims)        {
            new Claim(MyClaimTypes.TokenType, tokenType)
        };
        var token = new JwtSecurityToken(
            issuer: "SparUp",
            audience: "SparUp Api",
            claims: claimsList,
            expires: DateTime.UtcNow.Add(lifetime),
            signingCredentials: _credentials
        );
        return new JwtSecurityTokenHandler().WriteToken(token);
    }
    public string GenerateAccessToken(ApplicationUser user)    {
        var claims = new[]        {
            new Claim(MyClaimTypes.UserId, user.Id.ToString()),
            new Claim(ClaimTypes.Role, "User"),
            new Claim(ClaimTypes.Email, user.Email)
        };
        return GenerateToken(claims, TimeSpan.FromHours(1), MyPolicy.Access);
    }
    public string GenerateRefreshToken(Guid userId)    {
        var claims = new[]        {
            new Claim(MyClaimTypes.UserId, userId.ToString())
        };
        return GenerateToken(claims, TimeSpan.FromDays(1), MyPolicy.Refresh);
    }
    public string GenerateSessionToken(Guid userId)    {
        var claims = new[]        {
            new Claim(MyClaimTypes.UserId, userId.ToString())
        };
        return GenerateToken(claims, TimeSpan.FromMinutes(5), MyPolicy.Session);
    }
}
```

Лістинг Г.16 – Клас EmailService

```
public class EmailService(IOptions<SmtpConfigurationOptions> optionProvider)
{
   private SmtpConfigurationOptions Options => optionProvider.Value;
   public async Task SendEmailAsync(EmailMessage eMessage)
   {
      var message = new MimeMessage();
      message.From.Add(new MailboxAddress(eMessage.From, Options.Email));
      message.To.Add(new MailboxAddress(eMessage.To, eMessage.EmailAddress));
      message.Subject = eMessage.Subject;
      message.Body = new TextPart("plain")
      {
         Text = eMessage.Body
      };
      using var client = new SmtpClient();
      await client.ConnectAsync(Options.SmtpServer, Options.SmtpPort,
SecureSocketOptions.Auto);
      await client.AuthenticateAsync(Options.Email, Options.Password);
      await client.SendAsync(message);
      await client.DisconnectAsync(true);
   }
}
```

Лістинг Г.17 – Клас ApiKeyEncryptionService

```
using System.Security.Cryptography;
using System.Text;
namespace SparUp.WebApi.Endpoints.Services;
public class ApiKeyEncryptionService(IConfiguration configuration) : IDisposable
{
   private readonly byte[] _masterKey =
Encoding.ASCII.GetBytes(configuration["Encryption:MasterKey"]
                                          ?? throw new InvalidOperationException());
   private readonly RandomNumberGenerator _rng = RandomNumberGenerator.Create();
   public string GenerateIv()
   {
      var iv = new byte[16];
      _rng.GetBytes(iv);
      return Convert.ToBase64String(iv);
   }
   public string EncryptApiKey(Guid apiKey, string base64Iv)
   {
      var iv = Convert.FromBase64String(base64Iv);
      var apiKeyBytes = apiKey.ToByteArray();
      using var aes = Aes.Create();
      aes.Key = _masterKey;
      aes.IV = iv;
      using var encryptor = aes.CreateEncryptor(aes.Key, aes.IV);
      using var ms = new MemoryStream();
      using var cs = new CryptoStream(ms, encryptor, CryptoStreamMode.Write);
      cs.Write(apiKeyBytes, 0, apiKeyBytes.Length);
      cs.FlushFinalBlock();
```

```
        return Convert.ToBase64String(ms.ToArray());
    }
    public byte[] DecryptApiKey(string encryptedApiKey, string base64Iv)
    {
        var iv = Convert.FromBase64String(base64Iv);
        var encryptedBytes = Convert.FromBase64String(encryptedApiKey);
        using var aes = Aes.Create();
        aes.Key = _masterKey;
        aes.IV = iv;
        var decryptor = aes.CreateDecryptor(aes.Key, aes.IV);
        using var ms = new MemoryStream(encryptedBytes);
        using var cs = new CryptoStream(ms, decryptor, CryptoStreamMode.Read);
        var result = new byte[16];
        cs.Read(result, 0, 16);
        return result;
    }
    public void Dispose()
    {
        _rng.Dispose();
    }
}
```

Лістинг Г.18 — Клас ApiUserManager

```
using Microsoft.EntityFrameworkCore;
using SparUp.WebApi.Endpoints.Data;
using SparUp.WebApi.Endpoints.Entities;
namespace SparUp.WebApi.Endpoints.Services;
public class ApiUserManager(ApiKeyEncryptionService encryptionService,
ApplicationDbContext db)
{
    public async Task<ApplicationUserApi?> FindByIdAsync(Guid userId)
    {
        return await db.FindAsync<ApplicationUserApi>(userId);
    }

    public async Task<bool> RemoveAsync(Guid userId)
    {
        var user = await FindByIdAsync(userId);
        if (user == null)
            return false;
        db.Remove(user);
        await db.SaveChangesAsync();
        return true;
    }
    public async Task<Guid> GenerateRandomKeyAsync(Guid userId)
    {
        var userApi = await FindByIdAsync(userId);
        bool isCreated = userApi == null;
        userApi ??= new ApplicationUserApi() { ApplicationUserId = userId };
        userApi.CreatedAt = DateTime.UtcNow;
        userApi.Iv = encryptionService.GenerateIv();
        var guid = Guid.NewGuid();
```

```
        userApi.Encoded = encryptionService.EncryptApiKey(guid, userApi.Iv);
        if (isCreated)
        {
            db.Add(userApi);
        }
        else
        {
            db.Update(userApi);
        }
        await db.SaveChangesAsync();
        return guid;
    }
    public async Task<bool> ValidateKeyAsync(ApplicationUserApi api, Guid guid)
    {
        var decoded = encryptionService.DecryptApiKey(api.Encoded, api.Iv);
        if (db.Entry(api).State == EntityState.Detached)
        {
            api = await FindByIdAsync(api.ApplicationUserId) ?? throw new
InvalidOperationException();
        }
        return new Guid(decoded) == guid;
    }
    public Guid Decode(ApplicationUserApi api)
    {
        return new Guid(encryptionService.DecryptApiKey(api.Encoded, api.Iv));
    }
}
```

Лістинг Г.19 — Клас SettingsApiController

```
using System.Security.Claims;
using Asp.Versioning;
using Microsoft.AspNetCore.Authentication.JwtBearer;
using Microsoft.AspNetCore.Authorization;
using Microsoft.AspNetCore.Mvc;
using SparUp.WebApi.Endpoints.Attributes;
using SparUp.WebApi.Endpoints.Constants;
using SparUp.WebApi.Endpoints.Services;
namespace SparUp.WebApi.Endpoints.Controllers.Api;
[ApiController]
[ApiVersion("1")]
[Route("api/v{version:apiVersion}/settings/api")]
public class SettingsApiController(ApiUserManager apiUserManager)
    : ControllerBase
{
    [HttpGet]
    [Authorize(AuthenticationSchemes = JwtBearerDefaults.AuthenticationScheme, Policy =
MyPolicy.Access)]
    [CheckUserId]
    public async Task<ActionResult> GetApiAsync()
    {
        var userId = HttpContext.GetUserId()!;
        var userApi = await apiUserManager.FindByIdAsync(userId);
```

```
        if (userApi == null)
        {
            return NotFound(HelperErrorDetails.With("API_NOT_AVAILABLE", "Api user not
found"));
        }

        return Ok(HelperResultDto.Success(new
        {
            user = userApi.ApplicationUserId,
            key = apiUserManager.Decode(userApi)
        }));
    }

    [HttpPost]
    [Authorize(AuthenticationSchemes = JwtBearerDefaults.AuthenticationScheme, Policy =
MyPolicy.Access)]
    [CheckUserId]
    [ApplicationUser]
    public async Task<ActionResult> CreateApiAsync()
    {
        var userId = HttpContext.GetUserId();
        var userApi = await apiUserManager.FindByIdAsync(userId);

        if (userApi == null)
        {
            var key = await apiUserManager.GenerateRandomKeyAsync(userId);
            return Ok(HelperResultDto.Success(new
            {
                user = userId,
                key = key,
            }));
        }

        return Ok(HelperResultDto.Success(new
        {
            user = userApi.ApplicationUserId,
            key = apiUserManager.Decode(userApi)
        }));
    }

    [HttpPut]
    [Authorize(AuthenticationSchemes = JwtBearerDefaults.AuthenticationScheme, Policy =
MyPolicy.Access)]
    [CheckUserId]
    public async Task<ActionResult> RefreshAsync()
    {
        var userId = HttpContext.GetUserId();
        var userApi = await apiUserManager.FindByIdAsync(userId);

        if (userApi == null)
        {
```

```
            return NotFound(HelperErrorDetails.With("API_NOT_AVAILABLE", "Api user not
found"));
        }


        var key = await apiUserManager.GenerateRandomKeyAsync(userId);
        return Ok(HelperResultDto.Success(new
        {
            user = userApi.ApplicationUserId,
            key
        }));
    }

    [HttpDelete]
    [Authorize(AuthenticationSchemes = JwtBearerDefaults.AuthenticationScheme, Policy =
MyPolicy.Access)]
    [CheckUserId]
    public async Task<ActionResult> DeleteAsync()
    {
        var userId = HttpContext.GetUserId();
        return await apiUserManager.RemoveAsync(userId)
            ? Ok(ResultDto.InstanceOfSuccess)
            : NotFound(ResultDto.InstanceOfError);
    }
}
```

Лістинг Г.20 — Клас ChatHub

```
using System.Security.Claims;
using Asp.Versioning;
using Microsoft.AspNetCore.Authentication.JwtBearer;
using Microsoft.AspNetCore.Authorization;
using Microsoft.AspNetCore.Mvc;
using SparUp.WebApi.Endpoints.Attributes;
using SparUp.WebApi.Endpoints.Constants;
using SparUp.WebApi.Endpoints.Services;
namespace SparUp.WebApi.Endpoints.Controllers.Api;
[ApiController]
[ApiVersion("1")]
[Route("api/v{version:apiVersion}/settings/api")]
public class SettingsApiController(ApiUserManager apiUserManager)
    : ControllerBase
{
    [HttpGet]
    [Authorize(AuthenticationSchemes = JwtBearerDefaults.AuthenticationScheme, Policy =
MyPolicy.Access)]
    [CheckUserId]
    public async Task<ActionResult> GetApiAsync()
    {
        var userId = HttpContext.GetUserId()!;
        var userApi = await apiUserManager.FindByIdAsync(userId);
        if (userApi == null)
        {
```

```
            return NotFound(HelperErrorDetails.With("API_NOT_AVAILABLE", "Api user not
found"));
        }

        return Ok(HelperResultDto.Success(new
        {
            user = userApi.ApplicationUserId,
            key = apiUserManager.Decode(userApi)
        }));
    }

    [HttpPost]
    [Authorize(AuthenticationSchemes = JwtBearerDefaults.AuthenticationScheme, Policy =
MyPolicy.Access)]
    [CheckUserId]
    [ApplicationUser]
    public async Task<ActionResult> CreateApiAsync()
    {
        var userId = HttpContext.GetUserId();
        var userApi = await apiUserManager.FindByIdAsync(userId);

        if (userApi == null)
        {
            var key = await apiUserManager.GenerateRandomKeyAsync(userId);
            return Ok(HelperResultDto.Success(new
            {
                user = userId,
                key = key,
            }));
        }

        return Ok(HelperResultDto.Success(new
        {
            user = userApi.ApplicationUserId,
            key = apiUserManager.Decode(userApi)
        }));
    }

    [HttpPut]
    [Authorize(AuthenticationSchemes = JwtBearerDefaults.AuthenticationScheme, Policy =
MyPolicy.Access)]
    [CheckUserId]
    public async Task<ActionResult> RefreshAsync()
    {
        var userId = HttpContext.GetUserId();
        var userApi = await apiUserManager.FindByIdAsync(userId);

        if (userApi == null)
        {
            return NotFound(HelperErrorDetails.With("API_NOT_AVAILABLE", "Api user not
found"));
        }
```

```
        var key = await apiUserManager.GenerateRandomKeyAsync(userId);
        return Ok(HelperResultDto.Success(new
        {
            user = userApi.ApplicationUserId,
            key
        }));
    }

    [HttpDelete]
    [Authorize(AuthenticationSchemes = JwtBearerDefaults.AuthenticationScheme, Policy =
MyPolicy.Access)]
    [CheckUserId]
    public async Task<ActionResult> DeleteAsync()
    {
        var userId = HttpContext.GetUserId();
        return await apiUserManager.RemoveAsync(userId)
            ? Ok(ResultDto.InstanceOfSuccess)
            : NotFound(ResultDto.InstanceOfError);
    }
}
```

Лістинг Г.21 — Клас Startuo

```
using System.Text;
using Asp.Versioning;
using Asp.Versioning.ApiExplorer;
using Microsoft.AspNetCore.Authentication.JwtBearer;
using Microsoft.AspNetCore.Identity;
using Microsoft.AspNetCore.SignalR;
using Microsoft.AspNetCore.WebSockets;
using Microsoft.EntityFrameworkCore;
using Microsoft.Extensions.Options;
using Microsoft.IdentityModel.Tokens;
using Microsoft.OpenApi.Models;
using SparUp.WebApi.Endpoints.Constants;
using SparUp.WebApi.Endpoints.Controllers.Classes;
using SparUp.WebApi.Endpoints.Convertors;
using SparUp.WebApi.Endpoints.Data;
using SparUp.WebApi.Endpoints.Services;
using SparUp.WebApi.Endpoints.Swagger.Schematics;
using Swashbuckle.AspNetCore.Filters;
using Swashbuckle.AspNetCore.SwaggerGen;
using Swashbuckle.AspNetCore.SwaggerUI;
namespace SparUp.WebApi.Endpoints;
public class Startup(IConfiguration configuration)
{
    public void ConfigureServices(IServiceCollection services)
    {
        services.AddApiVersioning(options =>
            {
                options.DefaultApiVersion = new ApiVersion(1, 0);
```

```
            options.AssumeDefaultVersionWhenUnspecified = true;
            options.ReportApiVersions = true;
        })
        .AddApiExplorer(options =>
        {
            options.GroupNameFormat = "'v'VVV";
            options.SubstituteApiVersionInUrl = true;
        });
    services.Configure<SmtpConfigurationOptions>(
        configuration.GetSection("Smtp")
    );
    services.AddDbContext<ApplicationDbContext>(options =>
        options.UseSqlite("Data Source=app.db"));
    services.AddIdentity<ApplicationUser, ApplicationRole>()
        .AddEntityFrameworkStores<ApplicationDbContext>()
        .AddDefaultTokenProviders();
    services
        .AddSingleton<JwtTokenGenerator>()
        .AddSingleton<EmailService>()
        .AddScoped<ApiUserManager>()
        .AddSingleton<ApiKeyEncryptionService>();
    services.AddSignalR(options =>
    {
        options.EnableDetailedErrors = true;
    });
    services.AddSingleton<IUserIdProvider, UserIdProvider>();
    services.AddTransient<IConfigureOptions<SwaggerGenOptions>,
ConfigureSwaggerOptions>();
    services.AddSwaggerExamplesFromAssemblyOf<Startup>();
    services.AddSwaggerGen(c =>
    {
        c.AddSecurityDefinition("Bearer", new OpenApiSecurityScheme
        {
            Name = "Authorization",
            Type = SecuritySchemeType.ApiKey,
            Scheme = "Bearer",
            BearerFormat = "JWT",
            In = ParameterLocation.Header,
            Description = "Введіть JWT токен у форматі: Bearer {токен}"
        });
        c.AddSecurityRequirement(new OpenApiSecurityRequirement
        {
            {
                new OpenApiSecurityScheme
                {
                    Reference = new OpenApiReference
                    {
                        Type = ReferenceType.SecurityScheme,
                        Id = "Bearer"
                    }
                },
                new string[] {}
```

```
            }
        });
        c.MapType<DateOnly>(() => new OpenApiSchema
        {
            Type = "string",
            Format = "date"
        });
        c.SchemaFilter<JsonPolymorphicSchemaFilter>();
        c.ExampleFilters();
        c.UseAllOfToExtendReferenceSchemas();
        c.UseAllOfForInheritance();
        c.UseOneOfForPolymorphism();
        c.SelectDiscriminatorNameUsing(_ => "type");
        c.SelectDiscriminatorValueUsing(subType => subType.Name switch
        {
            nameof(CreateNotificationDto) => "notification",
            nameof(CreateTaskDto) => "task",
            _ => subType.Name
        });
    });
    services.AddCors(options =>
    {
        options.AddDefaultPolicy(policy =>
        {
            policy.SetIsOriginAllowed(origin => true)
                .AllowAnyHeader()
                .AllowAnyMethod()
                .AllowCredentials();
        });
    });
    services.AddHttpContextAccessor();
    services.AddHttpClient();
    services.AddControllers()
        .AddJsonOptions(options =>
        {
            options.JsonSerializerOptions.Converters.Add(new DateOnlyJsonConverter());
        });
    services.AddAuthentication("Bearer")
        .AddJwtBearer("Bearer", options =>
        {
            options.TokenValidationParameters = new TokenValidationParameters
            {
                ClockSkew = TimeSpan.Zero,
                ValidateIssuer = true,
                ValidateAudience = true,
                ValidateLifetime = true,
                ValidateIssuerSigningKey = true,
                ValidIssuer = "SparUp",
                ValidAudience = "SparUp Api",
                IssuerSigningKey = new SymmetricSecurityKey(
                    Encoding.UTF8.GetBytes(configuration["Jwt:Secret"]?[..32] ?? throw new
InvalidOperationException("JWT Secret is not configured.")))
```

```
            };
            options.Events = new JwtBearerEvents
            {
               OnMessageReceived = context =>
               {
                  var accessToken = context.Request.Query["access_token"];
                  var path = context.HttpContext.Request.Path;

                  if (!string.IsNullOrEmpty(accessToken) &&
path.StartsWithSegments("/api/v1/ws/chat"))
                  {
                     context.Token = accessToken;
                  }
                  return Task.CompletedTask;
               },
               OnAuthenticationFailed = context =>
               {
                  Console.WriteLine($"JWT Auth FAILED: {context.Exception.Message}");
                  Console.WriteLine($"Exception: {context.Exception}");
                  return Task.CompletedTask;
               },
               OnTokenValidated = context =>
               {
                  var userName = context.Principal?.Identity?.Name;
                  var userId = context.Principal?.FindFirst("id")?.Value;
                  Console.WriteLine($"JWT Token VALID for user: {userName}, ID:
{userId}");
                  return Task.CompletedTask;
               },
               OnChallenge = context =>
               {
                  Console.WriteLine($"JWT Challenge triggered: {context.Error}");
                  return Task.CompletedTask;
               }
            };
         });
      services.AddAuthentication(options =>
         {
            options.DefaultAuthenticateScheme = "Cookies";
            options.DefaultChallengeScheme = "Google";
         })
         .AddCookie("Cookies")
         .AddGoogle("Google", options =>
         {
            options.ClientId = configuration["Authentication:Google:Id"];
            options.ClientSecret = configuration["Authentication:Google:Secret"];
            options.CallbackPath = "/oauth/google/callback";
            options.SaveTokens = true;
         });
      services.AddAuthorization(options =>
      {
         options.AddPolicy(MyPolicy.Refresh, policy =>
```

```
            policy.RequireClaim(MyClaimTypes.TokenType, MyPolicy.Refresh));
        options.AddPolicy(MyPolicy.Access, policy =>
            policy.RequireClaim(MyClaimTypes.TokenType, MyPolicy.Access));
    });
    services.Configure<CookiePolicyOptions>(options =>
    {
        options.CheckConsentNeeded = context => false;
        options.MinimumSameSitePolicy = SameSiteMode.None;
    });
    services.AddWebSockets(options => { });
}
public void Configure(IApplicationBuilder app, IWebHostEnvironment env,
IApiVersionDescriptionProvider apiVersionDescriptionProvider)
{
    if (env.IsDevelopment())
    {
        app.UseSwagger();
        app.UseSwaggerUI(options =>
        {
            foreach (var description in apiVersionDescriptionProvider.ApiVersionDescriptions)
            {
                options.SwaggerEndpoint(
                    $"/swagger/{description.GroupName}/swagger.json",
                    description.GroupName.ToUpperInvariant());
            }
            options.EnableDeepLinking();
            options.DisplayOperationId();
            options.DefaultModelExpandDepth(2);
            options.DefaultModelRendering(ModelRendering.Example);
            options.DefaultModelsExpandDepth(1);
            options.EnableFilter();
            options.ShowExtensions();
            options.EnableValidator();
            options.ConfigObject.AdditionalItems["syntaxHighlight"] = new Dictionary<string,
object>
            {
                ["activated"] = true,
                ["theme"] = "agate"
            };
        });
    }
    using (var scope = app.ApplicationServices.CreateScope())
    {
        var db = scope.ServiceProvider.GetRequiredService<ApplicationDbContext>();
        db.Database.EnsureCreated();
    }
    app.Use(async (context, next) =>
    {
        var origin = context.Request.Headers["Origin"].ToString();

        if (!string.IsNullOrEmpty(origin))
        {
```

```
            context.Response.Headers.Add("Access-Control-Allow-Origin", origin);
            context.Response.Headers.Add("Access-Control-Allow-Credentials", "true");
            context.Response.Headers.Add("Access-Control-Allow-Methods", "GET, POST,
PUT, DELETE, OPTIONS");

            var requestedHeaders = context.Request.Headers["Access-Control-Request-
Headers"].ToString();
            if (!string.IsNullOrEmpty(requestedHeaders))
            {
               context.Response.Headers.Add("Access-Control-Allow-Headers",
requestedHeaders);
            }
            else
            {
               context.Response.Headers.Add("Access-Control-Allow-Headers",
                  "Content-Type, Authorization, x-requested-with, X-Requested-With, Accept,
Origin, Cache-Control, X-File-Name, x-signalr-user-agent, X-SignalR-User-Agent");
            }
         }

         if (context.Request.Method == "OPTIONS")
         {
            context.Response.StatusCode = 200;
            return;
         }

         await next();
      });
      app.UseWebSockets();
      app.UseCookiePolicy();
      app.UseRouting();
      app.UseAuthentication();
      app.UseAuthorization();
      app.UseHttpsRedirection();
      app.UseEndpoints(endpoints =>
      {
         endpoints.MapHub<ChatHub>("/api/v1/ws/chat");
         endpoints.MapControllers();
      });
   }
}
```

Лістинг Г.22 — Файл api.ts

```
import axios from 'axios';
import {jwtDecode} from 'jwt-decode';
const API_URL = process.env.NEXT_PUBLIC_API_URL || '/api';
let refreshingPromise: Promise<string | null> | null = null;
let cachedDecodedToken: any | null = null;
const api = axios.create({
   baseURL: API_URL,
   headers: {
      'Content-Type': 'application/json',
```

```
    },
    timeout: 10000,
});
const triggerAuthEvent = () => {
    if (typeof window !== 'undefined') {
        window.dispatchEvent(new Event('auth:unauthorized'));
    }
};
const isTokenValid = (decoded: any): boolean => {
    const currentTime = Date.now() / 1000;
    return decoded.exp && decoded.exp > currentTime;
};
const decodeAndCacheToken = (token: string): any | null => {
    try {
        const decoded: any = jwtDecode(token);
        cachedDecodedToken = decoded;
        return decoded;
    } catch {
        return null;
    }
};
const refreshAccessToken = async (): Promise<string | null> => {
    const refreshToken = localStorage.getItem('refresh');
    if (!refreshToken) return null;
    try {
        const response = await axios.get(`${API_URL}/auth/token/access`, {
            headers: {
                'Authorization': `Bearer ${refreshToken}`,
            }
        });
        if (!response.data.success) throw new Error(response.data.error || 'Failed to refresh');
        const { access: newToken } = response.data.data;
        localStorage.setItem('access', newToken);
        localStorage.setItem('refresh', refreshToken);
        decodeAndCacheToken(newToken); // кешуємо
        return newToken;
    } catch (e) {
        localStorage.removeItem('access');
        localStorage.removeItem('refresh');
        cachedDecodedToken = null;
        triggerAuthEvent();
        return null;
    }
};
const getValidAccessToken = async (): Promise<string | null> => {
    const localToken = localStorage.getItem('access');
    if (localToken) {
        const decoded = decodeAndCacheToken(localToken);
        if (decoded && isTokenValid(decoded)) {
            return localToken;
        }
    }
```

```
    if (refreshingPromise) {
        return await refreshingPromise;
    }
    refreshingPromise = refreshAccessToken().finally(() => {
        refreshingPromise = null;
    });
    return await refreshingPromise;
};
// Request interceptor
api.interceptors.request.use(async (config) => {
    const token = await getValidAccessToken();
    console.log(token);
    if (token) {
        config.headers.Authorization = `Bearer ${token}`;
    }
    return config;
}, error => Promise.reject(error));
api.interceptors.response.use(
    (response) => response,
    async (error) => {
        if (!error.response || error.response.status !== 401) {
            return Promise.reject(error);
        }
        const originalRequest = error.config;
        if (originalRequest._retry) {
            triggerAuthEvent();
            return Promise.reject(error);
        }
        originalRequest._retry = true;
        const newToken = await getValidAccessToken();
        if (!newToken) {
            return Promise.reject(error);
        }
        originalRequest.headers.Authorization = `Bearer ${newToken}`;
        return axios(originalRequest);
    }
);
export default api;
export { API_URL };
```

Лістинг Г.23 — Контекст AuthContext

```
// src/context/AuthContext.tsx
'use client'
import { createContext, useContext, useState, useEffect, ReactNode } from 'react';
import { useRouter } from 'next/navigation';
import api from "@/lib/api/api";
import { useQueryClient } from '@tanstack/react-query';
import {jwtDecode} from 'jwt-decode';
interface IdentityData {
    id: string;
}
interface AuthContextType {
```

```
    isAuthenticated: boolean;
    identity: IdentityData | null;
    login: (access: string, refresh: string) => void;
    logout: () => void;
}
const AuthContext = createContext<AuthContextType | undefined>(undefined);
export function useAuth() {
    const context = useContext(AuthContext);
    if (!context) {
        throw new Error('useAuth must be used within an AuthProvider');
    }
    return context;
}
export function AuthProvider({ children }: { children: ReactNode }) {
    const [isAuthenticated, setIsAuthenticated] = useState(false);
    const [identity, setIdentity] = useState<IdentityData | null>(null);
    const router = useRouter();
    const queryClient = useQueryClient();
    const extractIdentityFromToken = (token: string): IdentityData | null => {
        try {
            const decoded = jwtDecode(token);
            const currentTime = Date.now() / 1000;
            if (decoded.exp && decoded.exp < currentTime) {
                return null;
            }
            console.log(decoded);
            const userId = decoded.id;
            if (!userId) {
                return null;
            }
            return {
                id: userId
            };
        } catch (error) {
            console.error('Error decoding token:', error);
            return null;
        }
    };
    const logout = () => {
        localStorage.removeItem('access');
        localStorage.removeItem('refresh');
        delete api.defaults.headers.common['Authorization'];
        setIsAuthenticated(false);
        setIdentity(null);
        queryClient.clear();
        router.push('/auth/login');
    };
    const login = (access: string, refresh: string) => {
        if (!access || !refresh || access === refresh) {
            throw new Error('token is required');
        }
        const identityData = extractIdentityFromToken(access);
```

```
        if (!identityData) {
            throw new Error('Invalid or expired token');
        }
        localStorage.setItem('access', access);
        localStorage.setItem('refresh', refresh);
        api.defaults.headers.common['Authorization'] = `Bearer ${access}`;
        setIsAuthenticated(true);
        setIdentity(identityData);
    };
    useEffect(() => {
        const token = localStorage.getItem('access');
        if (token) {
            setIsAuthenticated(true);
            api.defaults.headers.common['Authorization'] = `Bearer ${token}`;
        }
        const refresh = localStorage.getItem('refresh');
        if (refresh) {
            const identityData = extractIdentityFromToken(refresh);
            if (identityData) {
                setIdentity(identityData);
            } else {
                logout();
            }
        }
        const handleUnauthorized = () => {
            setIsAuthenticated(false);
            setIdentity(null);
            queryClient.clear();
            router.push('/auth/login');
        };
        window.addEventListener('auth:unauthorized', handleUnauthorized);
        return () => {
            window.removeEventListener('auth:unauthorized', handleUnauthorized);
        };
    }, [router, queryClient]);
    return (
        <AuthContext.Provider value={{
            isAuthenticated,
            identity,
            login,
            logout
        }}>
            {children}
        </AuthContext.Provider>
    );
}
```

Лістинг Г.24 — Контекст QueryProvider

```
'use client';
import { QueryClient, QueryClientProvider } from '@tanstack/react-query';
import { ReactQueryDevtools } from '@tanstack/react-query-devtools';
import { useState } from 'react';
```

```
export default function QueryProvider({ children }: { children: React.ReactNode }) {
   const [queryClient] = useState(() => new QueryClient({
      defaultOptions: {
         queries: {
            staleTime: 60 * 1000,
            retry: (failureCount, error: any) => {
               if (error?.response?.status === 401) {
                  return false;
               }
               return failureCount < 2;
            },
         },
      },
   }));
   return (
      <QueryClientProvider client={queryClient}>
         {children}
         <ReactQueryDevtools initialIsOpen={false} />
      </QueryClientProvider>
   );
}
```

Лістинг Г.25 — Контекст AlertContext

```
"use client";
import React, {createContext, useContext, useState, ReactNode} from "react";
import Alert from "./Alert";
interface AlertItem {
   id: number;
   content: string | ReactNode;
   type: "success" | "error" | "info";
}
interface AlertContextType {
   notify: (content: string | ReactNode, type?: AlertItem["type"], timeout?: number) => void;
}
const AlertContext = createContext<AlertContextType | undefined>(undefined);
let idCounter = 0;
type Orientation = "start" | "middle" | "end";
export const AlertProvider = ({children, horizontal = "end", vertical = "start"}: {
   children: ReactNode,
   horizontal?: Orientation,
   vertical?: Orientation
}) => {
   const [notifications, setNotifications] = useState<AlertItem[]>([]);
   const notify = (content: string | ReactNode, type: AlertItem["type"] = "info", timeout:
number = 4000) => {
      const id = idCounter++;
      setNotifications((prev) => [...prev, {id, content, type}]);
      if (timeout > 0) {
         setTimeout(() => {
            setNotifications((prev) => prev.filter((n) => n.id !== id));
         }, timeout);
      }
```

```
    };
    const remove = (id: number) => {
        setNotifications((prev) => prev.filter((n) => n.id !== id));
    };
    const getVerticalOrientation = () => {
        return ({start: "toast-top", middle: "toast-middle", end: "toast-bottom"})[vertical] ?? "";
    }
    const getHorizontalOrientation = () => {
        return ({start: "toast-start", middle: "toast-center", end: "toast-end"})[horizontal] ?? "";
    }
    return (
        <AlertContext.Provider value={{notify}}>
            {children}
            <div className={`toast ${getHorizontalOrientation()} ${getVerticalOrientation()}`}>
                {notifications.map((n) => (
                    <Alert key={n.id} id={n.id} content={n.content} type={n.type}
onClose={remove}/>
                ))}
            </div>
        </AlertContext.Provider>
    );
};
export const useNotify = (): AlertContextType => {
    const context = useContext(AlertContext);
    if (!context) {
        throw new Error("useNotify must be used within a AlertProvider");
    }
    return context;
};
```

Лістинг Г.26 — Сторінка входу

```
"use client";
import { useState, useEffect } from "react";
import { useRouter } from "next/navigation";
import Link from "next/link";
import GoogleLoginButton from "@/components/GoogleOAuthButton";
import { useMutation } from "@tanstack/react-query";
import api from "@/lib/api/api";
import { useNotify } from "@/components/notifications/AlertContext";
import { FaEye, FaEyeSlash } from "react-icons/fa";
import { useAuth } from "@/context/AuthContext";
const LoginPage = () => {
    const router = useRouter();
    const { notify } = useNotify();
    const { login, isAuthenticated } = useAuth();

    useEffect(() => {
        if (isAuthenticated) {
            router.push("/app");
        }
    }, [isAuthenticated]);
    const tryLogin = async (refreshToken) => {
```

```
        try {
            console.log("Token", refreshToken);
            api.defaults.headers.common['Authorization'] = `Bearer ${refreshToken}`;
            const result = await api.get("auth/token/access");
            if (result.status === 200) {
                const accessToken = result.data.data.access;
                login(accessToken, refreshToken);
                notify("Успішний вхід", "success");
                router.push("/app");
            }
        } catch (error) {
            notify("Помилка отримання доступу", "error");
            console.error("Помилка після логіну:", error);
        }
    };
    const {
        isPending,
        mutate,
    } = useMutation({
        mutationFn: data => api.post("/auth/login", data).then(res => res.data),
        onSuccess: async (res) => {
            await tryLogin(res.data.refresh);
        },
        onError: d => {
            if (d?.response?.data?.error && Array.isArray(d.response.data.error)) {
                d.response.data.error.forEach(e => {
                    notify(e.description, "error");
                });
            } else {
                notify(`Помилка ${d?.response?.status || 'невідома'}`, "error");
            }
        }
    });
    const [email, setEmail] = useState('');
    const [password, setPassword] = useState('');
    const [showPassword, setShowPassword] = useState(false);
    const handleSubmit = async (e) => {
        e.preventDefault();
        mutate({ email, password });
    };
    const togglePasswordVisibility = () => {
        setShowPassword(!showPassword);
    };
    return (
        <>
            <h2 className="text-2xl font-semibold text-center text-primary mb-6">
                Увійти до облікового запису
            </h2>
            <form onSubmit={handleSubmit}>
                <div className="mb-4">
                    <label htmlFor="email" className="block text-sm font-medium text-base-
content">
```

```
            Електронна пошта
          </label>
          <input
            type="email"
            id="email"
            name="email"
            required
            value={email}
            onChange={(e) => setEmail(e.target.value)}
            className="mt-2 input input-bordered w-full"
            placeholder="ваша@пошта.com"
          />
        </div>
        <div className="mb-6">
          <label htmlFor="password" className="block text-sm font-medium text-base-
content">
            Пароль
          </label>
          <div className="relative">
            <input
              type={showPassword ? "text" : "password"}
              id="password"
              name="password"
              required
              value={password}
              onChange={(e) => setPassword(e.target.value)}
              className="mt-2 input input-bordered w-full pr-10"
              placeholder="Ваш пароль"
            />
            <button
              type="button"
              className="absolute inset-y-0 right-0 mt-2 pr-3 flex items-center cursor-
pointer transition-colors hover:text-primary focus:outline-none focus:ring-2 focus:ring-primary
focus:ring-opacity-50 rounded-full p-1"
              onClick={togglePasswordVisibility}
              title={showPassword ? "Приховати пароль" : "Показати пароль"}
            >
              {showPassword ? (
                <FaEyeSlash className="h-5 w-5" />
              ) : (
                <FaEye className="h-5 w-5" />
              )}
            </button>
          </div>
        </div>
        <button
          type="submit"
          className="btn btn-primary w-full"
          disabled={isPending}
        >
          {isPending ? 'Вхід...' : 'Увійти'}
        </button>
```

```
            </form>
            <div className="mt-4 text-center">
              <Link href="/auth/forgot-password" className="link text-sm">
                Забули пароль?
              </Link>
            </div>
            <div className="mt-4 text-center">
              <span className="text-sm text-base-content">Немає облікового запису?</span>
              <Link href="/auth/register" className="ml-1 link text-sm">
                Створити
              </Link>
            </div>
            <div className="divider text-sm text-base-content">Або за допомогою</div>
            <GoogleLoginButton onSuccess={async (refresh) => {
              console.log(refresh);
              await tryLogin(refresh);
            }}/>
          </>
        );
};
export default LoginPage;
```

Лістинг Г.27 — Модальний компонент CreateClassModal

```
"use client";
import { useState, useEffect } from "react";
import { useRouter } from "next/navigation";
import Link from "next/link";
import GoogleLoginButton from "@/components/GoogleOAuthButton";
import { useMutation } from "@tanstack/react-query";
import api from "@/lib/api/api";
import { useNotify } from "@/components/notifications/AlertContext";
import { FaEye, FaEyeSlash } from "react-icons/fa";
import { useAuth } from "@/context/AuthContext";
const LoginPage = () => {
    const router = useRouter();
    const { notify } = useNotify();
    const { login, isAuthenticated } = useAuth();

    useEffect(() => {
        if (isAuthenticated) {
            router.push("/app");
        }
    }, [isAuthenticated]);
    const tryLogin = async (refreshToken) => {
        try {
            console.log("Token", refreshToken);
            api.defaults.headers.common['Authorization'] = `Bearer ${refreshToken}`;
            const result = await api.get("auth/token/access");
            if (result.status === 200) {
                const accessToken = result.data.data.access;
                login(accessToken, refreshToken);
                notify("Успішний вхід", "success");
```

```
            router.push("/app");
          }
        } catch (error) {
          notify("Помилка отримання доступу", "error");
          console.error("Помилка після логіну:", error);
        }
      };
      const {
        isPending,
        mutate,
      } = useMutation({
        mutationFn: data => api.post("/auth/login", data).then(res => res.data),
        onSuccess: async (res) => {
          await tryLogin(res.data.refresh);
        },
        onError: d => {
          if (d?.response?.data?.error && Array.isArray(d.response.data.error)) {
            d.response.data.error.forEach(e => {
              notify(e.description, "error");
            });
          } else {
            notify(`Помилка ${d?.response?.status || 'невідома'}`, "error");
          }
        }
      });
      const [email, setEmail] = useState('');
      const [password, setPassword] = useState('');
      const [showPassword, setShowPassword] = useState(false);
      const handleSubmit = async (e) => {
        e.preventDefault();
        mutate({ email, password });
      };
      const togglePasswordVisibility = () => {
        setShowPassword(!showPassword);
      };
      return (
        <>
          <h2 className="text-2xl font-semibold text-center text-primary mb-6">
            Увійти до облікового запису
          </h2>
          <form onSubmit={handleSubmit}>
            <div className="mb-4">
              <label htmlFor="email" className="block text-sm font-medium text-base-
content">
                Електронна пошта
              </label>
              <input
                type="email"
                id="email"
                name="email"
                required
                value={email}
```

```
                onChange={(e) => setEmail(e.target.value)}
                className="mt-2 input input-bordered w-full"
                placeholder="ваша@пошта.com"
              />
            </div>
            <div className="mb-6">
              <label htmlFor="password" className="block text-sm font-medium text-base-
content">
                Пароль
              </label>
              <div className="relative">
                <input
                  type={showPassword ? "text" : "password"}
                  id="password"
                  name="password"
                  required
                  value={password}
                  onChange={(e) => setPassword(e.target.value)}
                  className="mt-2 input input-bordered w-full pr-10"
                  placeholder="Ваш пароль"
                />
                <button
                  type="button"
                  className="absolute inset-y-0 right-0 mt-2 pr-3 flex items-center cursor-
pointer transition-colors hover:text-primary focus:outline-none focus:ring-2 focus:ring-primary
focus:ring-opacity-50 rounded-full p-1"
                  onClick={togglePasswordVisibility}
                  title={showPassword ? "Приховати пароль" : "Показати пароль"}
                >
                  {showPassword ? (
                    <FaEyeSlash className="h-5 w-5" />
                  ) : (
                    <FaEye className="h-5 w-5" />
                  )}
                </button>
              </div>
            </div>
            <button
              type="submit"
              className="btn btn-primary w-full"
              disabled={isPending}
            >
              {isPending ? 'Вхід...' : 'Увійти'}
            </button>
          </form>
          <div className="mt-4 text-center">
            <Link href="/auth/forgot-password" className="link text-sm">
              Забули пароль?
            </Link>
          </div>
          <div className="mt-4 text-center">
            <span className="text-sm text-base-content">Немає облікового запису?</span>
```

```
            <Link href="/auth/register" className="ml-1 link text-sm">
              Створити
            </Link>
          </div>
          <div className="divider text-sm text-base-content">Або за допомогою</div>
          <GoogleLoginButton onSuccess={async (refresh) => {
            console.log(refresh);
            await tryLogin(refresh);
          }}/>
        </>
    );
};
export default LoginPage;
```

Лістинг Г.28 — Хук для роботи з SignalR useTaskChat

```
import {useState, useEffect, useCallback} from 'react';
import {useQuery, useQueryClient, useMutation} from '@tanstack/react-query';
import {HttpTransportType, HubConnectionBuilder, LogLevel} from '@microsoft/signalr';
import api, {API_URL} from "@/lib/api/api";
export const useTaskChat = (
    postId: string,
    studentId: string | null = null,
    enabled: boolean | (() => boolean) = true,
    options: {
        staleTime: number;
        logLevel: LogLevel;
    } = {}): object => {
    const [connection, setConnection] = useState(null);
    const [isConnected, setIsConnected] = useState(false);
    const [connectionState, setConnectionState] = useState('Disconnected');
    const queryClient = useQueryClient();
    const isEnabled = typeof enabled === 'function' ? enabled() : !!enabled;
    const isHookEnabled = isEnabled && !!postId;
    const {
        staleTime = 1000 * 60 * 5,
        logLevel = LogLevel.Error
    } = options;
    const MESSAGES_KEY = studentId
        ? ['chat', 'task', postId, 'student', studentId]
        : ['chat', 'task', postId];
    useEffect(() => {
        if (!isHookEnabled) return;
        const token = localStorage.getItem('access');
        if (!token) return;
        const params = new URLSearchParams({
            access_token: token,
            taskId: postId
        });
        if (studentId) {
            params.append('studentId', studentId);
        }
        const url = `${API_URL}/ws/chat?${params.toString()}`;
```

```
const newConnection = new HubConnectionBuilder()
    .withUrl(url, {
        transport: HttpTransportType.WebSockets,
    })
    .withAutomaticReconnect([0, 2000, 10000, 30000])
    .configureLogging(logLevel)
    .build();
newConnection.onclose(() => {
    setIsConnected(false);
    setConnectionState('Disconnected');
    setConnection(null);
});
newConnection.onreconnecting(() => {
    setIsConnected(false);
    setConnectionState('Reconnecting');
});
newConnection.onreconnected(() => {
    setIsConnected(true);
    setConnectionState('Connected');
});
newConnection.on('Connected', () => {
    setConnectionState('Connected');
});
newConnection.on('Disconnected', () => {
    setConnectionState('Disconnected');
});
setConnectionState('Connecting');
newConnection.start()
    .then(() => {
        setIsConnected(true);
        setConnectionState('Connected');
        setConnection(newConnection);
        newConnection.on('Receive', (messageData) => {
            if (!messageData) return;
            queryClient.setQueryData(MESSAGES_KEY, (oldMessages = []) => {
                const newMessage = {
                    id: messageData.id || `temp-${Date.now()}`,
                    message: messageData.message,
                    fullname: messageData.fullname,
                    photoUrl: messageData.photoUrl,
                    created: messageData.created || new Date().toISOString(),
                };
                return [...oldMessages, newMessage];
            });
        });
    })
    .catch(() => {
        setIsConnected(false);
        setConnectionState('Failed');
    });
return () => {
    if (newConnection) {
```

```
            newConnection.stop();
        }
        setConnection(null);
        setIsConnected(false);
        setConnectionState('Disconnected');
    };
}, [postId, studentId, queryClient, isHookEnabled, logLevel]);
const {
    data: messages = [],
    isLoading: messagesLoading,
    error: messagesError,
    refetch: refetchMessages
} = useQuery({
    queryKey: MESSAGES_KEY,
    queryFn: async () => {
        const url = studentId
            ? `/edu/post/${postId}/student/${studentId}/chat`
            : `/edu/post/${postId}/chat`;
        try {
            const response = await api.get(url);
            return response.data.data || [];
        } catch (error) {
            throw error;
        }
    },
    staleTime: staleTime,
    enabled: isHookEnabled,
    retry: (failureCount, error) => {
        if (failureCount > 2) return false;
        if (error?.response?.status === 401 ||
            error?.response?.status === 403 ||
            error?.response?.status === 404) {
            return false;
        }
        return true;
    }
});
const sendMessageMutation = useMutation({
    mutationFn: async (message) => {
        if (!isHookEnabled) {
            throw new Error('Хук деактивовано');
        }
        if (!connection) {
            throw new Error('Немає підключення до SignalR');
        }
        if (!isConnected) {
            throw new Error('SignalR не підключено');
        }
        if (connection.state !== 'Connected') {
            throw new Error(`SignalR у стані: ${connection.state}`);
        }
        try {
```

```
            await connection.invoke('Send', message);
        } catch (error) {
            throw error;
        }
    }
});
const sendMessage = useCallback((message) => {
    if (!isHookEnabled) {
        return Promise.reject(new Error('Хук деактивовано'));
    }
    if (!message || !message.trim()) {
        return Promise.reject(new Error('Повідомлення не може бути порожнім'));
    }
    return sendMessageMutation.mutateAsync(message.trim());
}, [sendMessageMutation, isHookEnabled]);
const addLocalMessage = useCallback((messageText, author = 'Система') => {
    if (!isHookEnabled) return;
    const tempMessage = {
        id: `local-${Date.now()}`,
        message: messageText,
        fullname: author,
        photoUrl: null,
        created: new Date().toISOString(),
    };
    queryClient.setQueryData(MESSAGES_KEY, (oldMessages = []) => {
        return [...oldMessages, tempMessage];
    });
    return tempMessage;
}, [queryClient, MESSAGES_KEY, isHookEnabled]);
return {
    messages,
    messagesLoading,
    messagesError,
    isSendingMessage: sendMessageMutation.isPending,
    sendError: sendMessageMutation.error,
    sendMessage,
    refetchMessages,
    addLocalMessage,
    isConnected,
    connectionState,
    isEnabled: isHookEnabled,
    connectionId: connection?.connectionId
};
};
```

Лістинг Г.29 — Компонент SafeMarkdownView

```
import React from 'react';
import ReactMarkdown from 'react-markdown';
import DOMPurify from 'dompurify';
import remarkGfm from 'remark-gfm';
import rehypeRaw from 'rehype-raw';
import { Prism as SyntaxHighlighter } from 'react-syntax-highlighter';
```

```
import {
    atomDark,
    vs,
    materialLight
} from 'react-syntax-highlighter/dist/esm/styles/prism';
import './markdown-styles.css'; // Імпортуємо окремий CSS файл для стилів Markdown
interface SafeMarkdownViewProps {
    content?: string | null | undefined;
    isDarkMode?: boolean;
}
const SUPPORTED_LANGUAGES = {
    'js': 'javascript',
    'jsx': 'jsx',
    'ts': 'typescript',
    'tsx': 'tsx',
    'python': 'python',
    'py': 'python',
    'css': 'css',
    'scss': 'scss',
    'html': 'html',
    'xml': 'xml',
    'json': 'json',
    'java': 'java',
    'c': 'c',
    'cpp': 'cpp',
    'cs': 'csharp',
    'csharp': 'csharp',
    'php': 'php',
    'ruby': 'ruby',
    'go': 'go',
    'rust': 'rust',
    'swift': 'swift',
    'kotlin': 'kotlin',
    'sql': 'sql',
    'sh': 'bash',
    'bash': 'bash',
    'shell': 'bash',
    'yaml': 'yaml',
    'yml': 'yaml',
    'dockerfile': 'dockerfile',
    'md': 'markdown',
    'markdown': 'markdown'
};
const getLanguage = (className?: string, metastring?: string): string => {
    if (className) {
        const match = /language-(\w+)/.exec(className || '');
        if (match && match[1]) {
            const lang = match[1].toLowerCase();
            return SUPPORTED_LANGUAGES[lang] || lang;
        }
    }
    // Додаткова перевірка для метаданих, якщо вони є
```

```
    if (metastring) {
      const langMatch = metastring.match(/lang=(\w+)/);
      if (langMatch && langMatch[1]) {
        const lang = langMatch[1].toLowerCase();
        return SUPPORTED_LANGUAGES[lang] || lang;
      }
    }
    return 'text'; // Повертаємо 'text' як мову за замовчуванням
};
const SafeMarkdownView: React.FC<SafeMarkdownViewProps> = ({ content, isDarkMode =
true }) => {
    if (!content) {
      return null;
    }
    // Вибір теми підсвічування в залежності від режиму
    const syntaxTheme = isDarkMode ? atomDark : materialLight;
    // Очищаємо HTML, щоб запобігти інжекціям
    const sanitizedContent = DOMPurify.sanitize(content);
    return (
      <div className="markdown-content">
        <ReactMarkdown
          children={sanitizedContent}
          remarkPlugins={[remarkGfm]} // Підтримка GFM (таблиці, списки, закреслення
тощо)
          rehypePlugins={[rehypeRaw]} // Дозволяє обмежену кількість HTML-тегів
          components={{
            // Кастомізація компонентів для різних елементів Markdown
            h1: ({node, ...props}) => <h1 className="md-h1" {...props} />,
            h2: ({node, ...props}) => <h2 className="md-h2" {...props} />,
            h3: ({node, ...props}) => <h3 className="md-h3" {...props} />,
            h4: ({node, ...props}) => <h4 className="md-h4" {...props} />,
            h5: ({node, ...props}) => <h5 className="md-h5" {...props} />,
            h6: ({node, ...props}) => <h6 className="md-h6" {...props} />,
            p: ({node, ...props}) => <p className="md-p" {...props} />,
            ul: ({node, ...props}) => <ul className="md-ul" {...props} />,
            ol: ({node, ...props}) => <ol className="md-ol" {...props} />,
            li: ({node, ...props}) => <li className="md-li" {...props} />,
            blockquote: ({node, ...props}) => <blockquote className="md-blockquote"
{...props} />,
            a: ({node, ...props}) => <a className="md-a" {...props} target="_blank"
rel="noopener noreferrer" />,
            pre: ({node, ...props}) => <pre className="md-pre" {...props} />,
            // Використовуємо SyntaxHighlighter для блоків коду
            code: ({node, inline, className, children, ...props}) => {
              // Якщо це інлайн-код, повертаємо простий тег <code>
              if (inline) {
                return <code className="md-inline-code" {...props}>{children}</code>;
              }
              // Визначаємо мову програмування
              const language = getLanguage(className, props.metastring);
              // Отримуємо текст коду і видаляємо зайві пробіли на початку/кінці
              const match = /language-(\w+)/.exec(className || '');
```

```
                const code = String(children).replace(/\n$/, '');
                // Відображення з підсвічуванням синтаксису
                return (
                  <div className="md-code-block">
                    {match && (
                      <div className="md-code-language">
                        {language.toUpperCase()}
                      </div>
                    )}
                    <SyntaxHighlighter
                      style={syntaxTheme}
                      language={language}
                      PreTag="div"
                      showLineNumbers={code.split('\n').length > 2} // Показуємо номери
рядків, якщо рядків більше 2
                      wrapLongLines={true} // Перенос довгих рядків
                      customStyle={{
                        margin: 0,
                        borderRadius: '0 0 0.375rem 0.375rem',
                        fontSize: '0.875rem',
                      }}
                    >
                      {code}
                    </SyntaxHighlighter>
                  </div>
                );
              },
              img: ({node, ...props}) => <img className="md-img" {...props} alt={props.alt ||
''} />,
              table: ({node, ...props}) => <table className="md-table" {...props} />,
              thead: ({node, ...props}) => <thead className="md-thead" {...props} />,
              tbody: ({node, ...props}) => <tbody className="md-tbody" {...props} />,
              tr: ({node, ...props}) => <tr className="md-tr" {...props} />,
              th: ({node, ...props}) => <th className="md-th" {...props} />,
              td: ({node, ...props}) => <td className="md-td" {...props} />
            }}
          />
        </div>
    );
};
export default SafeMarkdownView;
```