МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ЗАПОРІЗЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

МАТЕМАТИЧНИЙ ФАКУЛЬТЕТ
Кафедра комп'ютерних наук

КВАЛІФІКАЦІЙНА РОБОТА БАКАЛАВРА

на тему: **«ПРОЄКТУВАННЯ ТА РОЗРОБКА ВЕБПЛАТФОРМИ ДЛЯ СТВОРЕННЯ ТА ПОШИРЕННЯ МУЛЬТИМЕДІЙНИХ НАВЧАЛЬНИХ МАТЕРІАЛІВ»**

Виконав: студент 4 курсу, групи 6.1222
спеціальності 122 Комп'ютерні науки
(шифр і назва спеціальності)
освітньої програми комп'ютерні науки
(назва освітньої програми)
М.А. Головко
(ініціали та прізвище)
Керівник доцент кафедри комп'ютерних наук, доцент, к.т.н. Решевська К.С.
(посада, вчене звання, науковий ступінь, прізвище та ініціали)
Рецензент завідувач кафедри фундаментальної та прикладної математики, професор, д.т.н. Гребенюк С.М.
(посада, вчене звання, науковий ступінь, прізвище та ініціали)

Запоріжжя – 2026

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

ЗАПОРІЗЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

Факультет математичний

Кафедра комп'ютерних наук

Рівень вищої освіти бакалавр

Спеціальність 122 Комп'ютерні науки
(шифр і назва)

Освітня програма комп'ютерні науки

ЗАТВЕРДЖУЮ
В.о. завідувача кафедри
комп'ютерних наук, к.пед.н.,
доцент
_____ Пшенична О.С
(підпис)

" ____ " _____ 2025 р.

З А В Д А Н Н Я

НА КВАЛІФІКАЦІЙНУ РОБОТУ СТУДЕНТОВІ

Головку Микиті Антоновичу
(прізвище, ім'я та по-батькові)

1. Тема роботи Проєктування та розробка вебплатформи для створення та поширення мультимедійних навчальних матеріалів

керівник роботи Решевська Катерина Сергіївна, к.т.н., доцент
(прізвище, ім'я та по-батькові, науковий ступінь, вчене звання)

затверджені наказом ЗНУ від « 02 » грудня 2025 року № 1741-с

2. Строк подання студентом роботи 01.06.2026 р.

3. Вихідні дані до роботи 1. Постановка задачі.
2. Перелік літератури.

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити)
1. Технічне завдання.
2. Проєктування.
3. Програмна реалізація та тестування.

5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень)
презентація за темою доповіді

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		Завдання видав	Завдання прийняв

7. Дата видачі завдання 16.12.2025 р.

К А Л Е Н Д А Р Н И Й П Л А Н

№	Назва етапів кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1.	Розробка плану роботи.	12.01.2026	
2.	Збір вихідних даних.	26.01.2026	
3.	Обробка методичних та теоретичних	16.02.2026	
	джерел.		
4.	Розробка першого та другого розділу.	30.03.2026	
5.	Розробка третього розділу.	18.05.2026	
6.	Оформлення та нормоконтроль	25.05.2026	
	кваліфікаційної роботи бакалавра.		
7.	Захист кваліфікаційної роботи.	19.06.2026	

Студент _____ М.А. Головко
(підпис) (ініціали та прізвище)

Керівник роботи _____ К.С. Решевська
(підпис) (ініціали та прізвище)

Нормоконтроль пройдено

Нормоконтролер _____ О.Г. Спиця
(підпис) (ініціали та прізвище)

РЕФЕРАТ

Кваліфікаційна робота бакалавра «Проєктування та розробка вебплатформи для створення та поширення мультимедійних навчальних матеріалів»: 101 с., 40 рис., 7 табл., 25 джерел, 5 додатків.

ANGULAR, EXPRESS, JWT, MULTIMEDIA, MYSQL, NODE.JS, REST API, SEQUELIZE ORM, TAILWIND CSS, UML.

Об’єкт дослідження – вебплатформи для створення та поширення цифрових навчальних матеріалів.

Мета роботи – розробити вебплатформу для створення, редагування, публікації та поширення мультимедійних навчальних матеріалів.

Методи дослідження – аналітичний, моделювання, проєктування, програмний.

У роботі розглянуто проєктування та реалізацію вебплатформи для створення та поширення мультимедійних навчальних матеріалів. Клієнтська частина виконана засобами Angular, серверна – на Node.js та Express.js з REST API. Для збереження даних використано MySQL з ORM Sequelize.

Реалізовано автентифікацію, розмежування доступу за ролями, повний цикл роботи з навчальними матеріалами, підтримку мультимедійного контенту, пошук, фільтрацію, статистику та публічний перегляд без авторизації. Побудовано ER-модель і основні UML-діаграми, проведено unit-тестування компонента створення матеріалу.

Розроблена платформа забезпечує зручний інтерфейс для структурування освітнього контенту, додавання мультимедійних файлів і поширення матеріалів за публічним посиланням.

SUMMARY

Bachelor’s qualifying paper «Design and Development of a Web Platform for Creating and Distributing Multimedia Educational Materials»: 101 pages, 40 figures, 7 tables, 25 references, 5 supplements.

ANGULAR, EXPRESS, JWT, MULTIMEDIA, MYSQL, NODE.JS, REST API, SEQUELIZE ORM, TAILWIND CSS, UML.

The object of the study is web platforms for creating and sharing digital educational materials.

The aim of the study is to develop a web platform for creating, editing, publishing, and sharing multimedia educational materials.

The methods of research are analytical, modeling, design, and programming.

The paper considers the design and implementation of a web platform for creating and sharing multimedia educational materials. The client side is implemented using Angular. The server side is implemented using Node.js and Express.js and provides a REST API for working with users, categories, educational materials, multimedia files, public access, and statistics. MySQL is used for data storage, while interaction with the database is performed through Sequelize ORM.

The system implements authentication, role-based access control, a full lifecycle of educational material management, multimedia content support, search, filtering, statistics, and public access without authorization. An ER model of the database and key UML diagrams were developed, and unit testing of the material creation component was conducted.

Thus, the developed web platform provides a convenient interface for structuring educational content, adding multimedia files, and sharing materials via a public link.

ЗМІСТ

Завдання на кваліфікаційну роботу 2
Реферат 4
Summary 5
Вступ 8
1 Технічне завдання 10
1.1 Перелік умовних позначень, скорочень, термінів 10
1.2 Функціональні вимоги 11
1.2.1 Призначення і мета створення системи 11
1.2.2 Функціональні можливості системи 12
1.3 Нефункціональні вимоги 13
1.4 Опис предметної області 14
1.5 Опис системи 15
1.6 Огляд та порівняння аналогів 17
2 Проєктування 20
2.1 ER-модель бази даних та опис сутностей системи 20
2.2 Діаграма варіантів використання 23
2.2.1 Опис варіантів використання 25
2.3 Діаграма діяльності 35
2.4 Діаграма класів 37
2.5 Діаграма послідовності 39
2.6 Діаграма розгортання 40
2.7 Висновки до розділу 42
3 Програмна реалізація та тестування 43
3.1 Опис інструментів розробки 43
3.2 Основні складові системи 44
3.3 Встановлення та налаштування оточення 48
3.4 Опис API 54

3.5 Тестування проєкту 60
3.6 Керівництво користувача 62
3.6.1 Вхід та реєстрація 62
3.6.2 Перегляд дашборду 63
3.6.3 Перегляд списку навчальних матеріалів 63
3.6.4 Створення навчального матеріалу 64
3.6.5 Перегляд навчального матеріалу 66
3.6.6 Редагування та видалення матеріалу 67
3.6.7 Публікація навчального матеріалу 68
3.6.8 Публічний перегляд матеріалу 68
3.6.9 Керування категоріями 69
Висновки 71
Перелік посилань 72
Додаток А Angular MaterialFormComponent 74
Додаток Б Angular MaterialService 80
Додаток В Express material.controller.js 83
Додаток Г Express Material.js 92
Додаток Д Unit-тест MaterialFormComponent 94

ВСТУП

У сучасних умовах цифровізації освіти зростає потреба у простих і доступних інструментах для створення, збереження та поширення навчальних матеріалів. Викладачі, вчителі та інші автори освітнього контенту все частіше використовують не лише текстові матеріали, а й зображення, PDF-файли, відеопосилання та інші мультимедійні елементи [3, 9]. Такий формат подання інформації дозволяє зробити навчальний матеріал більш наочним, зручним для самостійного опрацювання та доступним для різних категорій користувачів.

На ринку існує багато освітніх платформ і сервісів для роботи з навчальним контентом, зокрема системи дистанційного навчання, хмарні сервіси для організації занять і конструктори інтерактивних матеріалів. Однак частина таких рішень має надмірний функціонал, потребує складного налаштування або орієнтована на повноцінне керування навчальним процесом, а не на швидке створення й поширення окремих мультимедійних навчальних матеріалів.

Актуальність роботи полягає у створенні вебплатформи, яка забезпечує повний цикл роботи з мультимедійним навчальним матеріалом: від реєстрації користувача та створення матеріалу до додавання текстового змісту, відеопосилань, зображень, PDF-файлів, публікації й поширення матеріалу за публічним посиланням [25]. Така система має бути зручною, зрозумілою для користувача та побудованою на сучасних вебтехнологіях.

З огляду на це, сформульовано мету та задачі дослідження.

Мета: розробити вебплатформу для створення, редагування, публікації та поширення мультимедійних навчальних матеріалів.

Задачі:

- сформулювати функціональні та нефункціональні вимоги до системи;
- проаналізувати предметну область створення та поширення мультимедійних навчальних матеріалів;
- розглянути та порівняти аналогічні програмні рішення;

– спроєктувати структуру бази даних і побудувати ER-модель;
– побудувати UML-діаграми для основних процесів системи;
– реалізувати вебплатформу для створення та поширення мультимедійних навчальних матеріалів;
– виконати тестування системи;
– підготувати керівництво користувача.

Об’єкт дослідження: вебплатформи для створення та поширення цифрових навчальних матеріалів.

Предмет дослідження: засоби та методи розробки клієнт-серверної вебплатформи для створення, редагування, публікації та поширення мультимедійних навчальних матеріалів.

Методи дослідження: аналітичний, моделювання, проєктування, програмний.

У першому розділі описано функціональні та нефункціональні вимоги до системи, розглянуто предметну область створення та поширення мультимедійних навчальних матеріалів, подано опис системи та огляд аналогів із порівняльною таблицею.

У другому розділі виконано проєктування вебплатформи: побудовано ER-модель бази даних, діаграму варіантів використання, діаграму діяльності, діаграми класів клієнтської та серверної частин, діаграму послідовності та діаграму розгортання.

У третьому розділі описано інструменти розробки, основні класи й модулі системи для прецеденту створення навчального матеріалу, налаштування оточення, API, тестування основного компонента та керівництво користувача.

1 ТЕХНІЧНЕ ЗАВДАННЯ

1.1 Перелік умовних позначень, скорочень, термінів

Система – вебплатформа для створення та поширення мультимедійних навчальних матеріалів.

Користувач – авторизована особа, яка взаємодіє із системою. У межах розроблюваної системи користувач може мати роль автора або адміністратора.

Гість – неавторизована особа, яка може переглядати опубліковані навчальні матеріали за публічним посиланням.

Автор – користувач, який має можливість створювати, редагувати, видаляти та публікувати власні навчальні матеріали.

Адміністратор – користувач із розширеними правами доступу, який може керувати категоріями та переглядати загальну інформацію про матеріали системи.

Навчальний матеріал – основна сутність системи, яка містить назву, опис, текстовий зміст, відеопосилання, категорію, статус публікації та прикріплені мультимедійні файли.

Мультимедійний навчальний матеріал – навчальний ресурс, який може поєднувати текст, зображення, PDF-файли, відеопосилання та інші допоміжні елементи для подання освітнього змісту.

Категорія – тематичний розділ, за допомогою якого навчальні матеріали групуються за предметом, напрямом або типом освітнього контенту.

Публікація матеріалу – зміна статусу навчального матеріалу, після якої він стає доступним для перегляду за публічним посиланням.

Публічне посилання – URL-адреса, за якою опублікований навчальний матеріал може бути переглянутий без авторизації в системі.

Дашборд – інформаційна панель системи, яка відображає узагальнені статистичні дані: кількість матеріалів, кількість опублікованих матеріалів, кількість чернеток, завантажених файлів і переглядів.

Angular – вебфреймворк для побудови сучасних застосунків і масштабованих вебінтерфейсів [1].

Node.js – середовище виконання JavaScript на сервері [14].

Express.js – легкий і гнучкий фреймворк для Node.js, який використовується для REST API та підключення middleware [4].

MySQL – система керування базами даних, яка використовується для збереження даних системи [13].

Sequelize – ORM для Node.js, що забезпечує роботу з реляційними базами даних [22].

REST API – підхід до організації взаємодії між клієнтською та серверною частинами системи через HTTP-запити [10].

JWT – JSON Web Token, формат токена, який використовується для передавання даних авторизації між клієнтом і сервером [19].

1.2 Функціональні вимоги

Функціональні вимоги визначають перелік основних можливостей, які має надавати вебплатформа користувачам [8, 23]. Для системи створення та поширення мультимедійних навчальних матеріалів ці вимоги пов’язані з реєстрацією користувачів, авторизацією, керуванням навчальними матеріалами, завантаженням мультимедійних файлів, публікацією матеріалів, переглядом статистики та керуванням категоріями.

1.2.1 Призначення і мета створення системи

Функціональне призначення системи – надати користувачу зручний інструмент для створення, редагування, збереження, публікації та поширення мультимедійних навчальних матеріалів.

Експлуатаційне призначення – система використовується авторами навчального контенту для підготовки цифрових навчальних матеріалів і надання до них доступу через публічні посилання.

Мета створення системи – розробити вебплатформу, яка дозволить користувачу швидко й зрозуміло створювати навчальні матеріали без потреби використовувати складні системи дистанційного навчання.

1.2.2 Функціональні можливості системи

У системі передбачено три ролі: гість, автор і адміністратор.

Гість має доступ лише до публічних сторінок опублікованих навчальних матеріалів. Автор отримує доступ до основних функцій створення й керування власними матеріалами. Адміністратор має розширені права та може керувати категоріями системи.

Система має надавати гостям наступні можливості:

– переглянути опублікований навчальний матеріал за публічним посиланням;
– копіювати посилання на опублікований навчальний матеріал;
– створити обліковий запис;
– увійти до облікового запису.

Авторизованим користувачам система має надавати такі можливості:

– вийти з облікового запису;
– переглянути статистику;
– створити/редагувати/видалити навчальний матеріал;
– переглянути список навчальних матеріалів;
– виконувати пошук;
– фільтрувати матеріали;
– публікувати навчальний матеріал/знімати матеріал з публікації;
– додавати до матеріалу зображення та PDF-файли;
– видаляти прикріплені файли.

Адміністратору система має надавати додаткові можливості:

– переглянути список категорій навчальних матеріалів;
– створити/редагувати/видалити категорію навчальних матеріалів.

1.3 Нефункціональні вимоги

Нефункціональні вимоги визначають загальні характеристики системи, які не описують окремі дії користувача, але впливають на якість, зручність, безпеку, продуктивність і можливість подальшої підтримки вебплатформи [8, 23].

Вимоги до інтерфейсу користувача. Інтерфейс системи має бути простим, зрозумілим і зручним для користувачів, що не мають спеціальної технічної підготовки.

Вимоги до адаптивності. Вебплатформа має коректно відображатися на різних типах пристроїв.

Вимоги до доступності. Форми введення даних мають містити зрозумілі підписи. Повідомлення про помилки повинні чітко пояснювати причину проблеми. Кнопки мають мати чіткі назви відповідно до виконуваної дії.

Вимоги до продуктивності. Система повинна швидко реагувати на дії користувача. Отримання списку навчальних матеріалів має виконуватися з використанням пагінації, пошуку та фільтрації, щоб уникнути надмірного завантаження клієнтської та серверної частин.

Вимоги до безпеки. Доступ до основних модулів системи має бути обмежений лише для авторизованих користувачів. Авторизація реалізується з використанням JWT-токена. Паролі користувачів повинні зберігатися на серверній стороні у вигляді хешу. Сервер повинен перевіряти як авторизацію, так і права користувача на виконання дій із конкретним матеріалом [15, 18, 19].

Вимоги до роботи з файлами. Система повинна дозволяти завантаження лише визначених типів файлів (зображення/PDF), обмежувати їх максимальний розмір та кількість за один запит. У разі видалення файлу запис про нього має видалятися з бази даних, а фізичний файл – із серверного сховища [20].

Вимоги до цілісності даних. Навчальний матеріал має бути пов’язаний з автором, категорією та прикріпленими файлами. У разі видалення матеріалу пов’язані з ним файли мають бути коректно оброблені системою.

Вимоги до надійності. Система повинна коректно обробляти типові помилки: відсутність доступу, невалідні дані, помилки авторизації, відсутність матеріалу, помилки завантаження файлів. Користувач має отримувати зрозуміле повідомлення про причину помилки.

Вимоги до сумісності. Вебдодаток має коректно працювати в актуальних версіях популярних браузерів: Google Chrome, Firefox, Edge, Opera та Safari.

Вимоги до масштабованості. Архітектура системи має передбачати можливість подальшого розширення.

Вимоги до збереження конфіденційних налаштувань. Конфігураційні дані, зокрема параметри підключення до бази даних і ключ JWT, мають зберігатися у файлі .env і не повинні потрапляти до системи контролю версій.

1.4 Опис предметної області

Предметною областю розроблюваної системи є створення та поширення цифрових мультимедійних навчальних матеріалів. До цієї області належать процеси підготовки навчального контенту, його структурування за категоріями, додавання мультимедійних елементів, публікація матеріалів і надання доступу до них через інтернет [3, 25].

У сучасному освітньому процесі все частіше використовуються цифрові матеріали, що поєднують різні способи подання інформації. Окрім лише текстового пояснення викладач може використовувати зображення, PDF-файли, відео, посилання на зовнішні ресурси та інші допоміжні матеріали. Такий підхід дає змогу зробити навчальний матеріал більш наочним, доступним і зручним для самостійного опрацювання [9].

Основною сутністю предметної області є навчальний матеріал. У межах системи він містить назву, короткий опис, основний текстовий зміст,

відеопосилання, категорію, статус публікації та набір прикріплених файлів. Навчальний матеріал може перебувати у двох станах: чернетка або опублікований матеріал. Чернетка доступна лише автору або адміністратору, а опублікований матеріал може бути відкритий за публічним посиланням.

Важливою складовою системи є категоризація навчальних матеріалів. Категорії дають змогу структурувати матеріали за предметами або тематичними напрямами. Завдяки цьому користувач може швидше знаходити потрібні матеріали та організовувати власний навчальний контент.

Окремим елементом предметної області є мультимедійні файли. До навчального матеріалу можуть додаватися зображення та PDF-документи. Зображення можуть використовуватися для ілюстрації понять, схем, прикладів або результатів роботи. PDF-файли можуть містити додаткові інструкції, завдання, конспекти або навчальні матеріали для завантаження [9].

Важливою складовою є публічний доступ. Після публікації матеріалу система формує публічне посилання, за яким матеріал може переглянути користувач без авторизації. Це дозволяє швидко поширювати матеріали серед студентів або інших зацікавлених осіб [25].

Основними проблемами предметної області, які має вирішити система, є розрізнене зберігання навчальних матеріалів, складність швидкого поширення контенту, відсутність єдиного простого інструменту для створення мультимедійних освітніх сторінок і потреба в доступному публічному перегляді матеріалів [3, 25].

У межах розроблюваної системи робота з мультимедійним навчальним матеріалом розглядається як послідовність етапів: створення, редагування, публікація, поширення та перегляд.

1.5 Опис системи

Розроблювана система є клієнт-серверною вебплатформою для створення та поширення мультимедійних навчальних матеріалів. Вона призначена для

авторів навчального контенту, які потребують простого та зручного у використанні інструменту для підготовки цифрових матеріалів і надання доступу до них через публічне посилання.

Клієнтська частина системи реалізується з використанням Angular [1]. Вона відповідає за відображення інтерфейсу користувача, роботу з формами, маршрутизацію, надсилання HTTP-запитів до серверної частини та відображення отриманих даних.

Серверна частина створюється на основі Node.js та Express.js [4, 14]. Вона забезпечує роботу REST API, обробку запитів клієнтської частини, авторизацію користувачів, перевірку прав доступу, роботу з навчальними матеріалами, категоріями, файлами та статистикою.

Для збереження даних використовується MySQL. Взаємодія серверної частини з базою даних виконується через Sequelize ORM [13, 22]. У базі даних зберігаються користувачі, категорії, навчальні матеріали та посилання на мультимедійні файли.

У системі передбачено три основні рольові моделі: гість, автор і адміністратор. Гість може переглядати лише опубліковані матеріали за публічними посиланнями. Автор може створювати, редагувати, видаляти та публікувати власні навчальні матеріали. Адміністратор може керувати категоріями та переглядати загальні дані системи.

Основними модулями системи є:

– модуль авторизації – забезпечує реєстрацію, вхід, вихід із системи та захист приватних сторінок;
– модуль навчальних матеріалів – дозволяє створювати, переглядати, редагувати, видаляти та публікувати матеріали;
– модуль категорій – забезпечує створення, редагування, видалення та перегляд категорій;
– модуль мультимедійних файлів – відповідає за завантаження, збереження та видалення зображень і PDF-файлів;
– модуль публічного перегляду – дозволяє переглядати опублікований матеріал без авторизації;

– модуль статистики – відображає кількість матеріалів, чернеток, опублікованих матеріалів, завантажених файлів і переглядів.

Основний бізнес-процес системи полягає в тому, що автор створює навчальний матеріал, додає до нього опис, текст, відеопосилання та мультимедійні файли. Після цього матеріал може залишатися чернеткою або бути опублікованим. У разі публікації система створює публічне посилання, яке можна скопіювати та передати іншим користувачам.

Таким чином, вебплатформа забезпечує повний цикл роботи з мультимедійним навчальним матеріалом: створення, редагування, зберігання, публікацію, поширення та перегляд.

1.6 Огляд та порівняння аналогів

На ринку цифрових освітніх сервісів існує багато інструментів, які частково вирішують задачу створення, зберігання або поширення навчальних матеріалів. Найбільш наближеними до розроблюваної системи є Google Classroom, Moodle, Canva for Education і Genially [2, 6, 7, 11].

Google Classroom – сервіс для організації навчального процесу, створення класів, керування завданнями, оцінками та навчальними матеріалами. Дозволяє створювати й керувати класами, завданнями та оцінками, додавати до завдань матеріали, зокрема YouTube-відео, Google Forms та файли з Google Drive. Перевагою є безкоштовне базове використання для шкіл, але розширені функції вже вимагають підписки [7].

Moodle – система керування навчанням, що використовується для створення онлайн-курсів і навчальних середовищ. Виділяється відкритістю коду, що полегшує її розширення, але при цьому є складною для користувача. Також, на відміну від розглянутих хмарних рішень і нашого застосунка, потребує власних серверних потужностей [11].

Canva for Education – освітня версія сервісу Canva, яка дозволяє створювати навчальні плани, інфографіку, постери, відео та інші візуальні

матеріали. Як у випадку з Google Classroom є безкоштовним для шкіл (K-12), але для університетів вже необхідно купувати підписку [2].

Genially – сервіс для створення інтерактивного та анімованого контенту. Він використовується для створення презентацій, інфографіки, вікторин, escape rooms та інших інтерактивних матеріалів без потреби програмування. Надає широкий безкоштовний функціонал, але для додаткових можливостей необхідно оформити підписку [6].

У таблиці 1.1 наведено порівняння аналогів із розроблюваною системою.

Таблиця 1.1 – Порівняння аналогів

Критерій	**Google Classroom**	**Moodle**	**Canva for Education**	**Genially**	**Розроблювана система**
Створення навчальних матеріалів	Так	Так	Так	Так	Так
Додавання тексту	Так	Так	Так	Так	Так
Додавання зображень	Так	Так	Так	Так	Так
Додавання PDF-файлів	Так	Так	Обмежено	Обмежено	Так
Додавання відеопосилань	Так	Так	Так	Так	Так
Публічний перегляд без авторизації	Обмежено	Залежить від налаштувань	Так	Так	Так
Керування категоріями	Частково	Так	Ні	Ні	Так
Дашборд зі статистикою матеріалів	Частково	Так	Обмежено	Обмежено	Так
Простота розгортання власної системи	Не є власною системою	Потребує встановлення та адміністрування	Не є власною системою	Не є власною системою	Так
Власна база даних і контроль структури	Ні	Так	Ні	Ні	Так
Призначення	Організація навчального процесу	Повноцінна система керування	Створення дизайнів	Інтерактивний контент	Створення та поширення мультимедійних матеріалів

Аналіз аналогів показує, що наявні рішення мають широкий функціонал, але часто орієнтовані або на повне керування навчальним процесом, або на створення окремих візуальних чи інтерактивних матеріалів. Google Classroom і Moodle більше зосереджені на організації навчання, класів, курсів і завдань. Canva for Education і Genially добре підходять для створення візуального та інтерактивного контенту, але не є спеціалізованими вебплатформами з власною базою даних для керування навчальними матеріалами в межах розробленої системи. Крім того, більшість цих сервісів використовує модель монетизації, де розширені можливості доступні лише за платною підпискою.

Головною перевагою розроблюваної системи є поєднання простоти й достатнього функціоналу: користувач може створити матеріал, додати текст, відеопосилання, зображення та PDF-файли, опублікувати матеріал і поширити його за публічним посиланням. При цьому система має власну клієнт-серверну архітектуру, базу даних, авторизацію, розмежування ролей, REST API та дашборд зі статистикою. На відміну від комерційних аналогів, повний функціонал системи, включно зі статистикою та керуванням базою даних, залишається доступним без додаткових фінансових витрат з боку користувача.

Отже, розроблювана вебплатформа не є повним аналогом систем дистанційного навчання, але вирішує більш спеціалізовану задачу – створення та поширення мультимедійних навчальних матеріалів без обмежень, характерних для платних версій комерційних платформ.

2 ПРОЄКТУВАННЯ

2.1 ER-модель бази даних та опис сутностей системи

База даних вебплатформи призначена для збереження інформації про користувачів, категорії навчальних матеріалів, навчальні матеріали та прикріплені до них мультимедійні файли. Використання реляційної бази даних дозволяє структуровано зберігати інформацію про основні сутності системи та зв’язки між ними [13, 22]. Основна логіка системи побудована навколо навчального матеріалу, який створюється автором, може належати до певної категорії, містити текстовий зміст, відеопосилання, прикріплені файли та мати статус чернетки або опублікованого матеріалу.

Для опису логічної структури даних використано ER-модель, яка дає змогу визначити основні сутності системи, їхні атрибути та зв’язки між ними на етапі проєктування [23].

ER-модель системи наведено на рисунку 2.1.

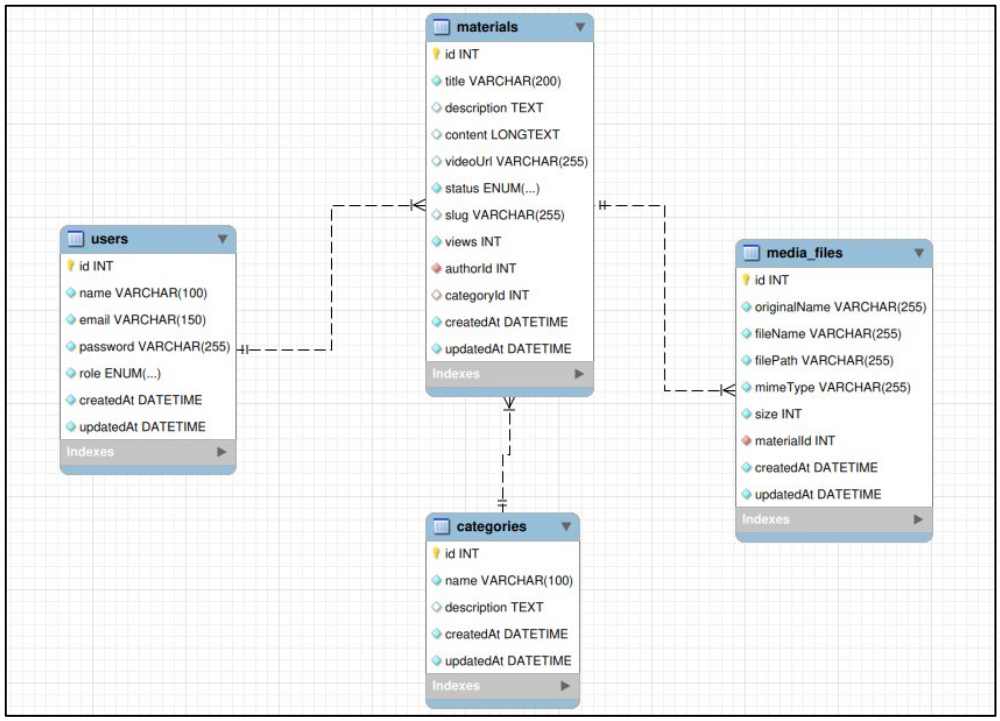


Рисунок 2.1 – ER-діаграма системи

У базі даних виділено чотири основні сутності: User, Category, Material та MediaFile.

User – сутність, що зберігає дані користувача системи. Вона містить ім’я, email, хеш пароля, роль користувача та службові дати створення й оновлення запису. Роль користувача визначає його права в системі. У розроблюваній вебплатформі передбачено ролі author та admin. Автор може створювати, редагувати, видаляти та публікувати власні навчальні матеріали, а адміністратор має розширені права, зокрема можливість керування категоріями.

Category – сутність, що використовується для групування навчальних матеріалів за предметами, напрямами або тематичними розділами. Для кожної категорії зберігаються назва, опис і службові дати створення та оновлення. Категорії дозволяють структурувати навчальний контент і спрощують фільтрацію матеріалів у системі.

Material – основна сутність системи, яка описує мультимедійний навчальний матеріал. Для кожного матеріалу зберігаються назва, короткий опис, основний текстовий зміст, відеопосилання, статус, унікальний slug для публічного посилання, кількість переглядів, ідентифікатор автора та ідентифікатор категорії. Матеріал може мати статус draft або published. Статус draft означає, що матеріал є чернеткою і доступний лише автору або адміністратору. Статус published означає, що матеріал опубліковано і він може бути переглянутий за публічним посиланням.

MediaFile – сутність, що зберігає інформацію про мультимедійні файли, прикріплені до навчального матеріалу. Для кожного файлу зберігаються оригінальна назва, системна назва файлу, шлях до файлу, MIME-тип, розмір, ідентифікатор навчального матеріалу та службові дати створення й оновлення. У межах реалізованої системи до матеріалу можуть додаватися зображення та PDF-файли.

Зв’язки між сутностями надалі реалізуються на серверній частині за допомогою Sequelize ORM, що дозволяє описувати асоціації між моделями та виконувати запити до пов’язаних даних [22].

Між сутностями встановлено такі зв’язки:

– один користувач може створювати багато навчальних матеріалів;
– один навчальний матеріал належить одному автору;
– одна категорія може містити багато навчальних матеріалів;
– один навчальний матеріал може належати до однієї категорії або не мати категорії;
– один навчальний матеріал може мати багато прикріплених мультимедійних файлів;
– кожен мультимедійний файл належить одному навчальному матеріалу.

Особливістю моделі є те, що основна сутність Material пов’язує між собою користувача, категорію та мультимедійні файли. Саме навколо цієї сутності побудовано ключові функції системи: створення навчального матеріалу, редагування, публікація, зняття з публікації, публічний перегляд, підрахунок переглядів і завантаження файлів.

Поле authorId у сутності Material використовується для зв’язку матеріалу з користувачем, який його створив. Завдяки цьому серверна частина може перевіряти, чи має поточний користувач право редагувати або видаляти конкретний матеріал. Якщо користувач має роль адміністратора, він може переглядати ширший набір даних системи, тоді як автор працює переважно зі своїми матеріалами.

Поле categoryId у сутності Material використовується для зв’язку матеріалу з категорією. Цей зв’язок є необов’язковим, оскільки навчальний матеріал може бути створений без вибору категорії. У разі видалення категорії матеріали не повинні видалятися автоматично, оскільки вони залишаються самостійними одиницями навчального контенту.

Поле slug використовується для формування публічного посилання на опублікований матеріал. Це дозволяє відкривати матеріал не за числовим ідентифікатором, а за зрозумілою URL-адресою. Поле views використовується для підрахунку кількості переглядів опублікованого матеріалу на публічній сторінці.

Сутність MediaFile пов’язана з Material через поле materialId. Такий зв’язок дозволяє зберігати декілька файлів для одного навчального матеріалу. Фізично файли зберігаються на сервері, а в базі даних зберігається інформація, необхідна для їх відображення, відкриття та видалення.

Запропонована ER-модель забезпечує цілісне збереження даних вебплатформи, розмежування матеріалів між авторами, групування навчального контенту за категоріями та можливість прикріплення мультимедійних файлів. Вона є основою для реалізації серверної логіки, REST API, публічного перегляду матеріалів, дашборду зі статистикою та функцій керування навчальним контентом [4, 22, 23].

2.2 Діаграма варіантів використання

Діаграма варіантів використання відображає основні функціональні можливості системи з погляду користувачів та показує, які дії можуть виконувати різні актори в межах вебплатформи [5, 12]. Для розроблюваної системи така діаграма є важливою, оскільки вона дозволяє наочно подати розмежування доступу між гостем, автором і адміністратором.

У системі виділено три актори: гість, автор та адміністратор. Гість є неавторизованим користувачем, який може переглядати опубліковані навчальні матеріали за публічним посиланням, копіювати посилання на матеріал, а також перейти до сторінок реєстрації або входу.

Автор є авторизованим користувачем системи, який отримує доступ до основних функцій вебплатформи: створення, редагування, видалення, публікації та перегляду власних навчальних матеріалів.

Адміністратор має розширені права доступу та може додатково керувати категоріями навчальних матеріалів.

Діаграму варіантів використання системи наведено на рисунку 2.2.

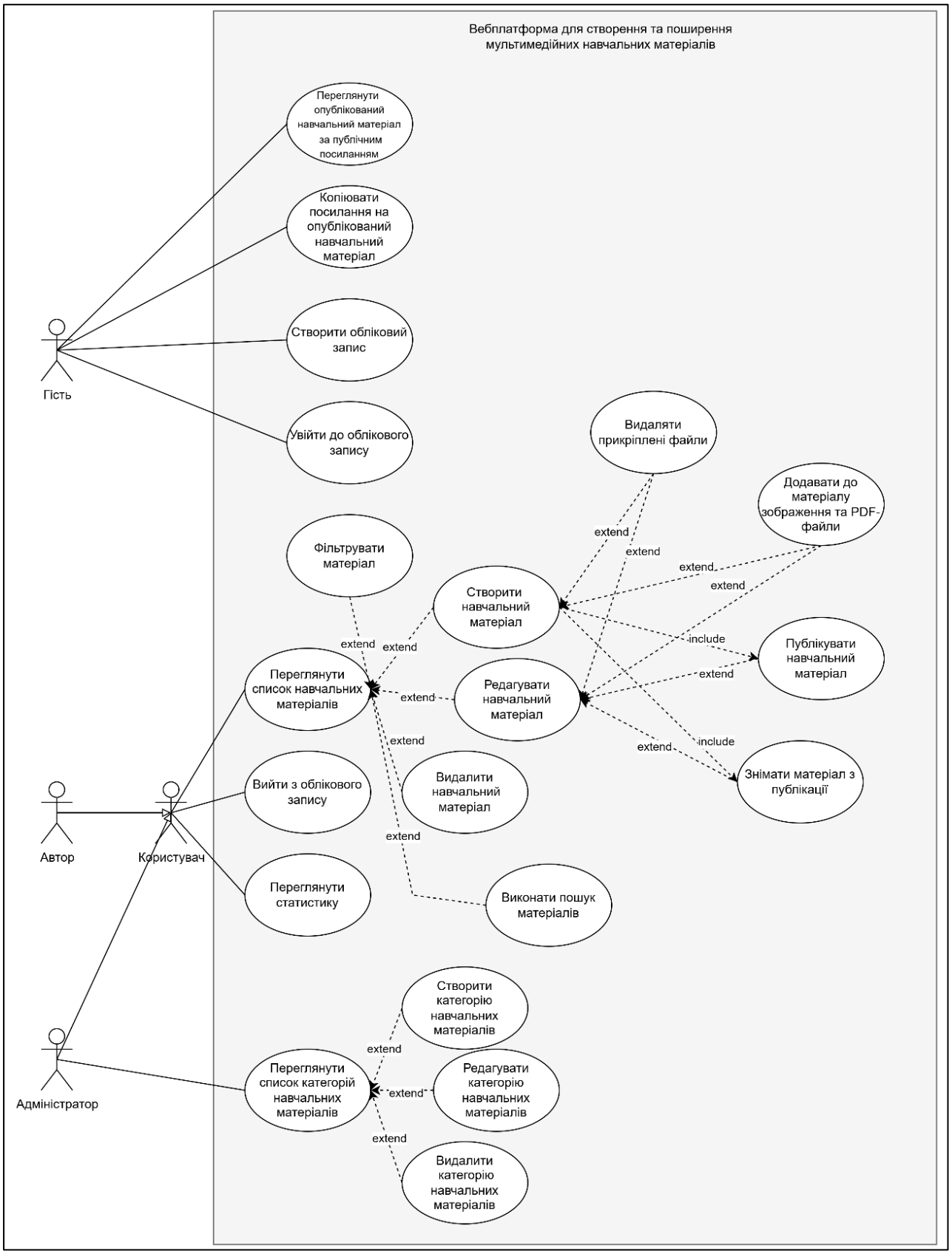


Рисунок 2.2 – Діаграма варіантів використання

Основним прецедентом системи є «Створити навчальний матеріал», оскільки саме він реалізує ключове призначення вебплатформи – підготовку

мультимедійного навчального контенту. У межах цього сценарію автор вводить назву матеріалу, короткий опис, основний текстовий зміст, за потреби додає відеопосилання, обирає категорію та завантажує мультимедійні файли. Після створення матеріал може залишатися чернеткою або бути опублікованим.

2.2.1 Опис варіантів використання

Прецедент «Створити обліковий запис».

Призначення: даний варіант використання надає можливість гостю створити обліковий запис у системі.

Передумови: користувач знаходиться на сторінці реєстрації.

Основний потік подій: даний варіант використання починає виконуватися, коли користувач вводить ім’я, email, пароль і натискає кнопку реєстрації. Система перевіряє коректність введених даних, перевіряє, чи не використовується вказаний email іншим користувачем, хешує пароль і створює новий обліковий запис із роллю автора. Після успішної реєстрації користувач отримує доступ до захищених сторінок системи.

Виняткова ситуація 1: невалідні дані – система виводить повідомлення про помилку та не створює обліковий запис.

Виняткова ситуація 2: email уже використовується – система повідомляє користувача, що обліковий запис із таким email уже існує.

Прецедент «Увійти до облікового запису».

Призначення: даний варіант використання надає можливість користувачу увійти до системи.

Передумови: користувач має створений обліковий запис і знаходиться на сторінці входу.

Основний потік подій: даний варіант використання починає виконуватися, коли користувач вводить email, пароль і натискає кнопку входу. Система перевіряє наявність користувача з таким email, порівнює введений пароль із

хешем пароля в базі даних, формує JWT-токен і передає його клієнтській частині. Після цього користувач переходить до основних модулів вебплатформи.

Виняткова ситуація 1: email або пароль введено неправильно – система виводить повідомлення про помилку.

Виняткова ситуація 2: обліковий запис не знайдено – система не надає доступ до захищених сторінок.

Прецедент «Вийти з облікового запису».

Призначення: даний варіант використання надає можливість авторизованому користувачу завершити роботу із системою.

Передумови: користувач авторизований у системі.

Основний потік подій: даний варіант використання починає виконуватися, коли користувач натискає кнопку виходу. Клієнтська частина видаляє збережений JWT-токен і дані користувача з локального сховища, після чого перенаправляє користувача на сторінку входу.

Виняткова ситуація 1: локальні дані користувача вже відсутні – система все одно перенаправляє користувача на сторінку входу.

Прецедент «Переглянути опублікований навчальний матеріал за публічним посиланням».

Призначення: даний варіант використання надає можливість гостю або авторизованому користувачу переглянути опублікований навчальний матеріал без входу до системи.

Передумови: навчальний матеріал має статус опублікованого, а користувач має публічне посилання на нього.

Основний потік подій: даний варіант використання починає виконуватися, коли користувач відкриває публічне посилання. Система знаходить матеріал за унікальним slug, перевіряє, чи має він статус опублікованого, збільшує кількість переглядів і відображає назву, опис, основний текст, відеопосилання, автора, категорію та прикріплені файли.

Виняткова ситуація 1: матеріал не знайдено – система виводить повідомлення, що матеріал недоступний.

Виняткова ситуація 2: матеріал має статус чернетки – система не відкриває матеріал для публічного перегляду.

Прецедент «Копіювати посилання на опублікований навчальний матеріал».

Призначення: даний варіант використання надає можливість користувачу скопіювати посилання на опублікований навчальний матеріал.

Передумови: матеріал має статус опублікованого та має сформований slug.

Основний потік подій: даний варіант використання починає виконуватися, коли користувач натискає кнопку копіювання посилання. Система формує URL-адресу публічної сторінки матеріалу та копіює її до буфера обміну.

Виняткова ситуація 1: матеріал не має публічного посилання – система повідомляє користувача, що посилання ще не створено.

Виняткова ситуація 2: браузер не дозволяє доступ до буфера обміну – система не виконує копіювання.

Прецедент «Переглянути статистику».

Призначення: даний варіант використання надає можливість авторизованому користувачу переглянути узагальнену інформацію про навчальні матеріали.

Передумови: користувач авторизований і знаходиться на сторінці дашборду.

Основний потік подій: даний варіант використання починає виконуватися, коли користувач відкриває сторінку дашборду. Система надсилає запит до сервера, отримує статистичні дані та відображає кількість матеріалів, кількість чернеток, кількість опублікованих матеріалів, кількість завантажених файлів, загальну кількість переглядів, останні створені матеріали та популярні матеріали.

Виняткова ситуація 1: статистичні дані відсутні – система відображає нульові значення або порожні списки.

Виняткова ситуація 2: помилка завантаження статистики – система виводить повідомлення про технічну проблему.

Прецедент «Переглянути список навчальних матеріалів».

Призначення: даний варіант використання надає можливість авторизованому користувачу переглянути список навчальних матеріалів.

Передумови: користувач авторизований і знаходиться на сторінці навчальних матеріалів.

Основний потік подій: даний варіант використання починає виконуватися, коли користувач відкриває сторінку матеріалів. Система надсилає запит до сервера, отримує список матеріалів і відображає їх у вигляді карток із назвою, описом, категорією, статусом, автором, датою створення та кількістю переглядів. Для автора система відображає його власні матеріали, а для адміністратора – ширший перелік матеріалів системи.

Виняткова ситуація 1: матеріали відсутні – система відображає порожній стан і пропонує створити перший навчальний матеріал.

Виняткова ситуація 2: помилка завантаження даних – система виводить повідомлення про технічну проблему.

Прецедент «Виконати пошук матеріалів».

Призначення: даний варіант використання надає можливість авторизованому користувачу виконати пошук навчальних матеріалів за назвою або описом.

Передумови: користувач авторизований і знаходиться на сторінці списку навчальних матеріалів.

Основний потік подій: даний варіант використання починає виконуватися, коли користувач вводить пошуковий запит і застосовує фільтр. Система надсилає запит до сервера з параметром пошуку та відображає лише ті матеріали, назва або опис яких відповідають введеному запиту.

Виняткова ситуація 1: за пошуковим запитом нічого не знайдено – система відображає порожній стан.

Виняткова ситуація 2: виникла помилка під час пошуку – система виводить повідомлення про технічну проблему.

Прецедент «Фільтрувати матеріали».

Призначення: даний варіант використання надає можливість авторизованому користувачу фільтрувати навчальні матеріали за статусом і категорією.

Передумови: користувач авторизований і знаходиться на сторінці списку навчальних матеріалів.

Основний потік подій: даний варіант використання починає виконуватися, коли користувач обирає статус матеріалу або категорію. Система надсилає запит до сервера з відповідними параметрами фільтрації та відображає матеріали, що відповідають вибраним умовам.

Виняткова ситуація 1: матеріали за заданими параметрами не знайдено – система відображає повідомлення про відсутність результатів.

Виняткова ситуація 2: параметри фільтрації некоректні – система не виконує фільтрацію та повідомляє про помилку.

Прецедент «Створити навчальний матеріал».

Призначення: даний варіант використання надає можливість автору створити новий мультимедійний навчальний матеріал.

Передумови: користувач авторизований і знаходиться на сторінці створення матеріалу.

Основний потік подій: даний варіант використання починає виконуватися, коли автор вводить назву матеріалу, короткий опис, основний текстовий зміст, за потреби додає відеопосилання, обирає категорію, визначає статус матеріалу та натискає кнопку збереження. Система перевіряє введені дані, створює навчальний матеріал у базі даних, формує унікальний slug і прив’язує матеріал до поточного користувача.

Виняткова ситуація 1: назва матеріалу відсутня або занадто коротка – система виводить повідомлення про помилку.

Виняткова ситуація 2: обраної категорії не існує – система не створює матеріал і повідомляє про помилку.

Виняткова ситуація 3: відеопосилання має некоректний формат – система не зберігає матеріал.

Прецедент «Редагувати навчальний матеріал».

Призначення: даний варіант використання надає можливість автору змінити дані раніше створеного навчального матеріалу.

Передумови: користувач авторизований, має створений матеріал і знаходиться на сторінці його редагування.

Основний потік подій: даний варіант використання починає виконуватися, коли користувач відкриває форму редагування матеріалу. Система завантажує поточні дані матеріалу та підставляє їх у форму. Користувач змінює назву, опис, текст, відеопосилання, категорію або статус і натискає кнопку збереження. Система перевіряє дані та оновлює матеріал у базі даних.

Виняткова ситуація 1: матеріал не знайдено – система повідомляє користувача про помилку.

Виняткова ситуація 2: користувач намагається редагувати матеріал, який йому не належить – система блокує дію.

Виняткова ситуація 3: введені дані не проходять валідацію – система не зберігає зміни.

Прецедент «Видалити навчальний матеріал».

Призначення: даний варіант використання надає можливість автору видалити навчальний матеріал.

Передумови: користувач авторизований і має право керувати відповідним матеріалом.

Основний потік подій: даний варіант використання починає виконуватися, коли користувач натискає кнопку видалення матеріалу та підтверджує дію. Система перевіряє право доступу до матеріалу, видаляє запис із бази даних і прибирає матеріал зі списку навчальних матеріалів.

Виняткова ситуація 1: матеріал не знайдено – система виводить повідомлення про помилку.

Виняткова ситуація 2: користувач не має права видаляти цей матеріал – система блокує дію.

Виняткова ситуація 3: виникла помилка під час видалення – система повідомляє про технічну проблему.

Прецедент «Додати мультимедійні файли».

Призначення: даний варіант використання надає можливість автору прикріпити до навчального матеріалу зображення або PDF-файли.

Передумови: користувач авторизований, має право керувати матеріалом і знаходиться на сторінці створення або редагування матеріалу.

Основний потік подій: даний варіант використання починає виконуватися, коли користувач обирає один або декілька файлів і зберігає матеріал. Система перевіряє тип, розмір і кількість файлів, зберігає фізичні файли на сервері та створює записи про них у базі даних із прив’язкою до відповідного навчального матеріалу.

Виняткова ситуація 1: файл має недозволений тип – система відхиляє завантаження.

Виняткова ситуація 2: файл перевищує допустимий розмір – система повідомляє користувача про помилку.

Виняткова ситуація 3: користувач намагається додати файл до чужого матеріалу – система блокує дію.

Прецедент «Опублікувати навчальний матеріал».

Призначення: даний варіант використання надає можливість автору зробити навчальний матеріал доступним для публічного перегляду.

Передумови: користувач авторизований, матеріал існує в системі та користувач має право керувати ним.

Основний потік подій: даний варіант використання починає виконуватися, коли користувач натискає кнопку публікації матеріалу. Система перевіряє право доступу, за потреби формує унікальний slug, змінює статус матеріалу на published і повертає публічне посилання на матеріал.

Виняткова ситуація 1: матеріал не знайдено – система повідомляє про помилку.

Виняткова ситуація 2: користувач не має права публікувати цей матеріал – система блокує дію.

Виняткова ситуація 3: виникла помилка під час оновлення статусу – система повідомляє про технічну проблему.

Прецедент «Видалити прикріплений файл».

Призначення: даний варіант використання надає можливість автору видалити файл, прикріплений до навчального матеріалу.

Передумови: користувач авторизований, файл існує в системі, а користувач має право керувати матеріалом, до якого цей файл належить.

Основний потік подій: даний варіант використання починає виконуватися, коли користувач натискає кнопку видалення біля прикріпленого файлу та підтверджує дію. Система перевіряє право доступу, видаляє фізичний файл із серверного сховища та видаляє запис про файл із бази даних.

Виняткова ситуація 1: файл не знайдено – система повідомляє про помилку.

Виняткова ситуація 2: користувач не має права видаляти цей файл – система блокує дію.

Виняткова ситуація 3: фізичний файл уже відсутній на сервері – система видаляє запис із бази даних або повідомляє про технічну проблему залежно від ситуації.

Прецедент «Зняти матеріал з публікації».

Призначення: даний варіант використання надає можливість автору зробити раніше опублікований матеріал недоступним для публічного перегляду.

Передумови: користувач авторизований, матеріал існує в системі, має статус опублікованого та користувач має право керувати ним.

Основний потік подій: даний варіант використання починає виконуватися, коли користувач натискає кнопку зняття матеріалу з публікації. Система перевіряє право доступу, змінює статус матеріалу на draft і оновлює дані матеріалу.

Виняткова ситуація 1: матеріал не знайдено – система повідомляє про помилку.

Виняткова ситуація 2: користувач не має права змінювати статус матеріалу – система блокує дію.

Виняткова ситуація 3: виникла помилка під час оновлення статусу – система повідомляє про технічну проблему.

Прецедент «Переглянути список категорій навчальних матеріалів».

Призначення: даний варіант використання надає можливість адміністратору переглянути список категорій навчальних матеріалів.

Передумови: користувач авторизований і має роль адміністратора.

Основний потік подій: даний варіант використання починає виконуватися, коли адміністратор відкриває сторінку категорій. Система надсилає запит до сервера, отримує список категорій і відображає їх у таблиці з назвою, описом, датою створення та доступними діями.

Виняткова ситуація 1: категорії відсутні – система відображає порожній стан.

Виняткова ситуація 2: користувач не має ролі адміністратора – система перенаправляє його на дашборд або блокує доступ.

Прецедент «Створити категорію навчальних матеріалів».

Призначення: даний варіант використання надає можливість адміністратору створити нову категорію навчальних матеріалів.

Передумови: користувач авторизований, має роль адміністратора і знаходиться на сторінці категорій.

Основний потік подій: даний варіант використання починає виконуватися, коли адміністратор вводить назву категорії, за потреби додає опис і натискає кнопку створення. Система перевіряє введені дані, перевіряє унікальність назви категорії та створює новий запис у базі даних.

Виняткова ситуація 1: назва категорії не вказана – система виводить повідомлення про помилку.

Виняткова ситуація 2: категорія з такою назвою вже існує – система не створює дубль і повідомляє про помилку.

Виняткова ситуація 3: користувач не має ролі адміністратора – система блокує дію.

Прецедент «Редагувати категорію навчальних матеріалів».

Призначення: даний варіант використання надає можливість адміністратору змінити дані існуючої категорії.

Передумови: користувач авторизований, має роль адміністратора, а категорія існує в системі.

Основний потік подій: даний варіант використання починає виконуватися, коли адміністратор обирає категорію для редагування, змінює її назву або опис і натискає кнопку збереження. Система перевіряє введені дані, перевіряє відсутність дубліката за назвою та оновлює категорію в базі даних.

Виняткова ситуація 1: категорію не знайдено – система повідомляє про помилку.

Виняткова ситуація 2: нова назва категорії дублює вже існуючу – система не зберігає зміни.

Виняткова ситуація 3: користувач не має ролі адміністратора – система блокує дію.

Прецедент «Видалити категорію навчальних матеріалів».

Призначення: даний варіант використання надає можливість адміністратору видалити категорію навчальних матеріалів.

Передумови: користувач авторизований, має роль адміністратора, а категорія існує в системі.

Основний потік подій: даний варіант використання починає виконуватися, коли адміністратор натискає кнопку видалення біля категорії та підтверджує дію. Система перевіряє право доступу, видаляє категорію з бази даних і оновлює список категорій. Матеріали, які були пов’язані з цією категорією, не видаляються, а залишаються в системі без категорії.

Виняткова ситуація 1: категорію не знайдено – система повідомляє про помилку.

Виняткова ситуація 2: користувач не має ролі адміністратора – система блокує дію.

Виняткова ситуація 3: виникла помилка під час видалення – система повідомляє про технічну проблему.

2.3 Діаграма діяльності

Діаграма діяльності використовується для відображення послідовності дій під час виконання бізнес-процесів системи [5, 12]. Для розроблюваної вебплатформи основним процесом є створення навчального матеріалу, оскільки саме він реалізує головне призначення системи – підготовку мультимедійного навчального контенту з можливістю подальшої публікації та поширення.

Діаграму діяльності для прецеденту «Створити навчальний матеріал» наведено на рисунках 2.3 – 2.4.

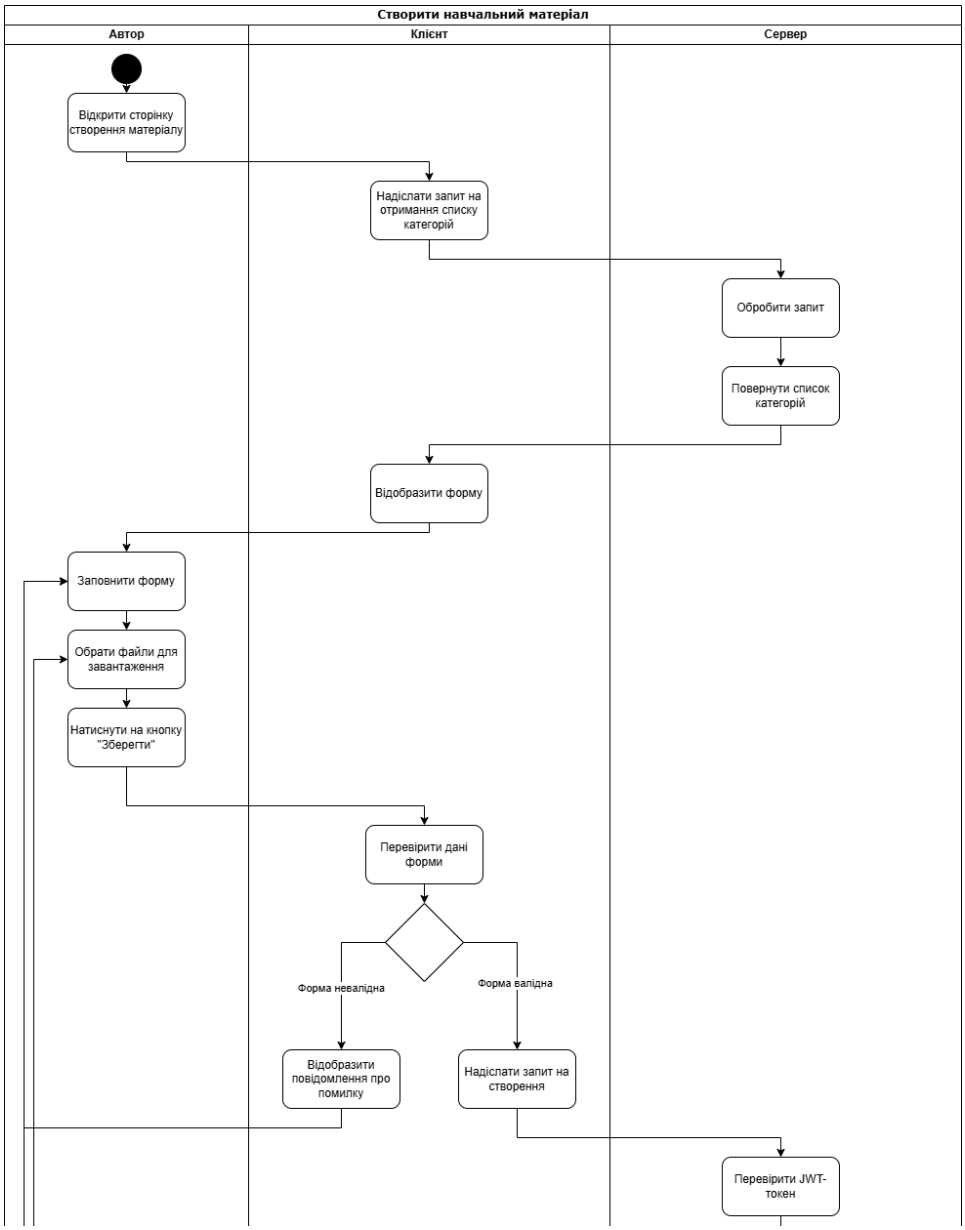


Рисунок 2.3 – Діаграма діяльності процесу створення навчального матеріалу

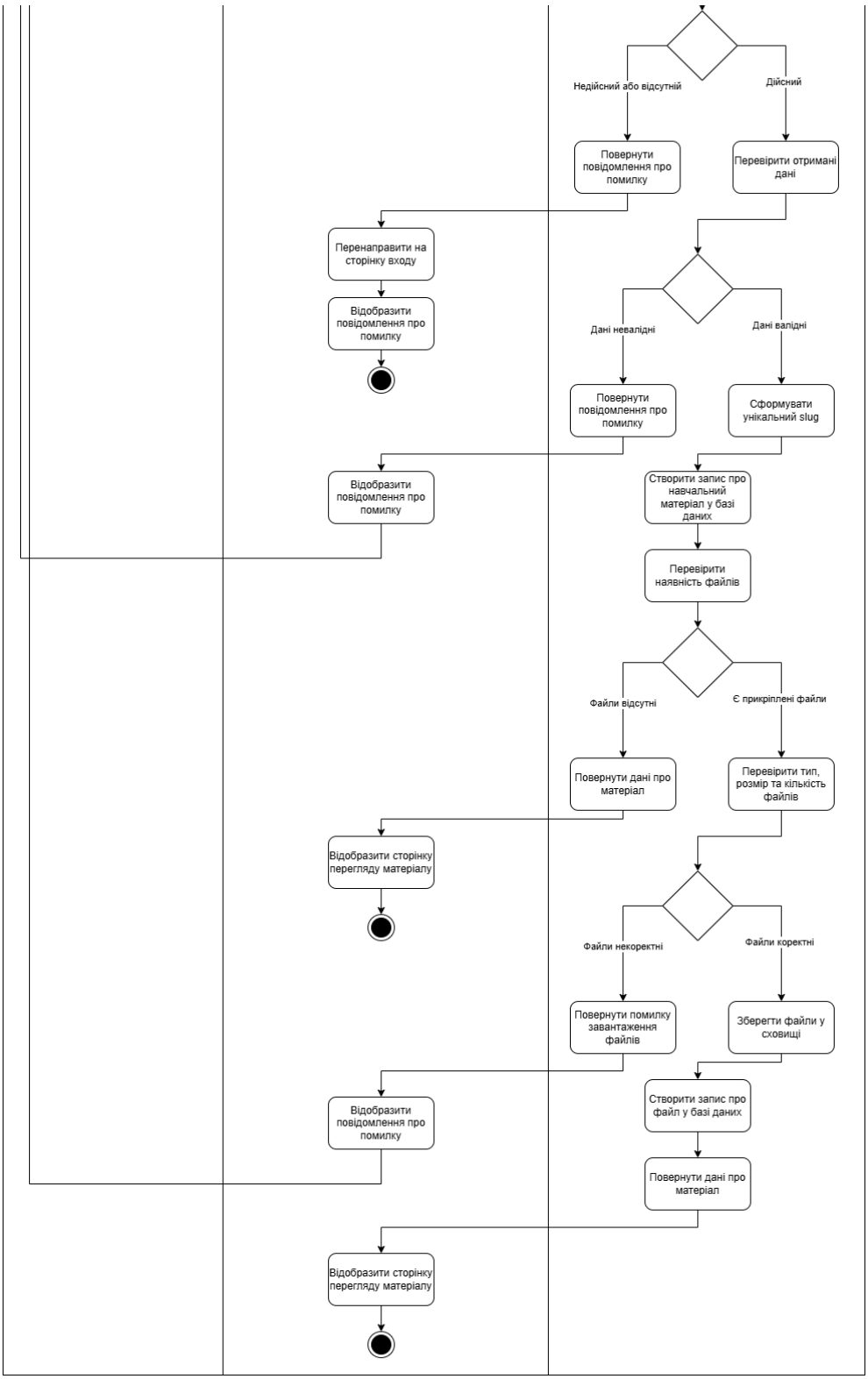


Рисунок 2.4 – Діаграма діяльності процесу створення навчального матеріалу (продовження)

Дана діаграма демонструє послідовність дій автора та системи під час створення навчального матеріалу. Вона відображає основні етапи процесу: відкриття форми, заповнення даних, перевірку введеної інформації, створення

запису в базі даних, завантаження мультимедійних файлів і перехід до сторінки перегляду матеріалу. Окремо враховано альтернативні сценарії, пов’язані з невалідними даними або помилками під час завантаження файлів.

2.4 Діаграма класів

Діаграма класів використовується для відображення основних структурних елементів системи, їхніх атрибутів, методів і зв’язків між ними [5, 12]. На відміну від діаграми варіантів використання, яка показує систему з погляду користувача, діаграма класів дозволяє описати внутрішню організацію програмної реалізації.

Оскільки розроблювана вебплатформа має клієнт-серверну архітектуру, доцільно розділити діаграму класів на дві частини: клієнтську та серверну. Клієнтська частина реалізована на Angular і містить компоненти, сервіси, моделі даних, guard-и та HTTP-interceptor [1]. Серверна частина реалізована на Node.js та Express.js і містить маршрути, контролери, middleware, Sequelize-моделі та конфігураційні модулі [4, 22].

Клієнтська діаграма класів відображає взаємодію Angular-компонентів із сервісами, які відповідають за HTTP-запити до REST API. Основними класами клієнтської частини є компоненти авторизації, дашборду, списку матеріалів, форми створення/редагування матеріалу, перегляду матеріалу, публічного перегляду та керування категоріями. Дані між клієнтом і сервером передаються через типізовані моделі User, Category, Material, MediaFile та Statistics. Діаграму класів зі сторони клієнта наведено на рисунку 2.5.

Серверна діаграма класів відображає структуру Express-додатка. Вона містить контролери, маршрути, middleware та Sequelize-моделі. Основними моделями серверної частини є User, Category, Material та MediaFile. Саме ці моделі відповідають сутностям бази даних і використовуються контролерами для виконання CRUD-операцій. Діаграму класів зі сторони сервера наведено на рисунку 2.6.

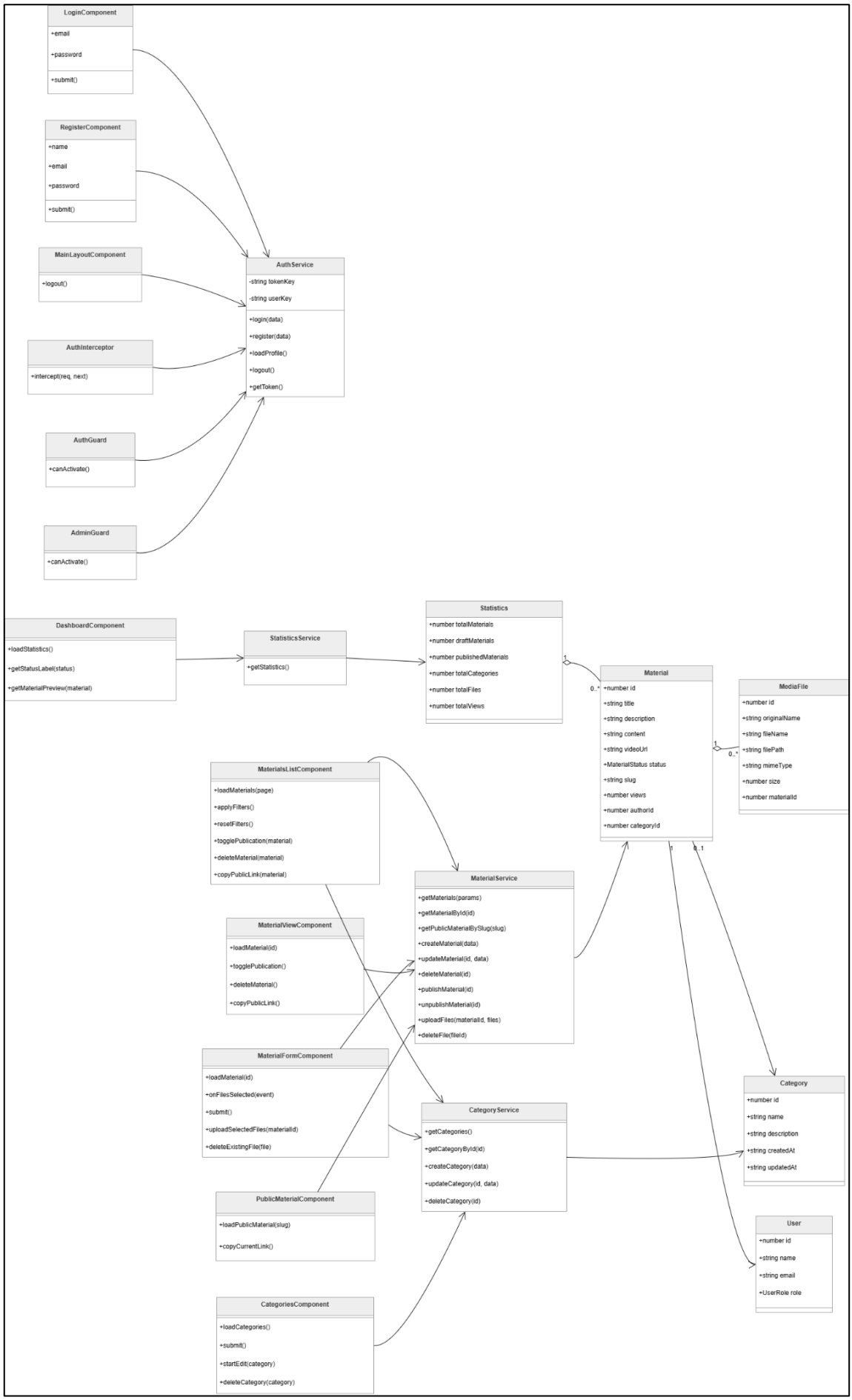


Рисунок 2.5 – Діаграма класів зі сторни клієнта

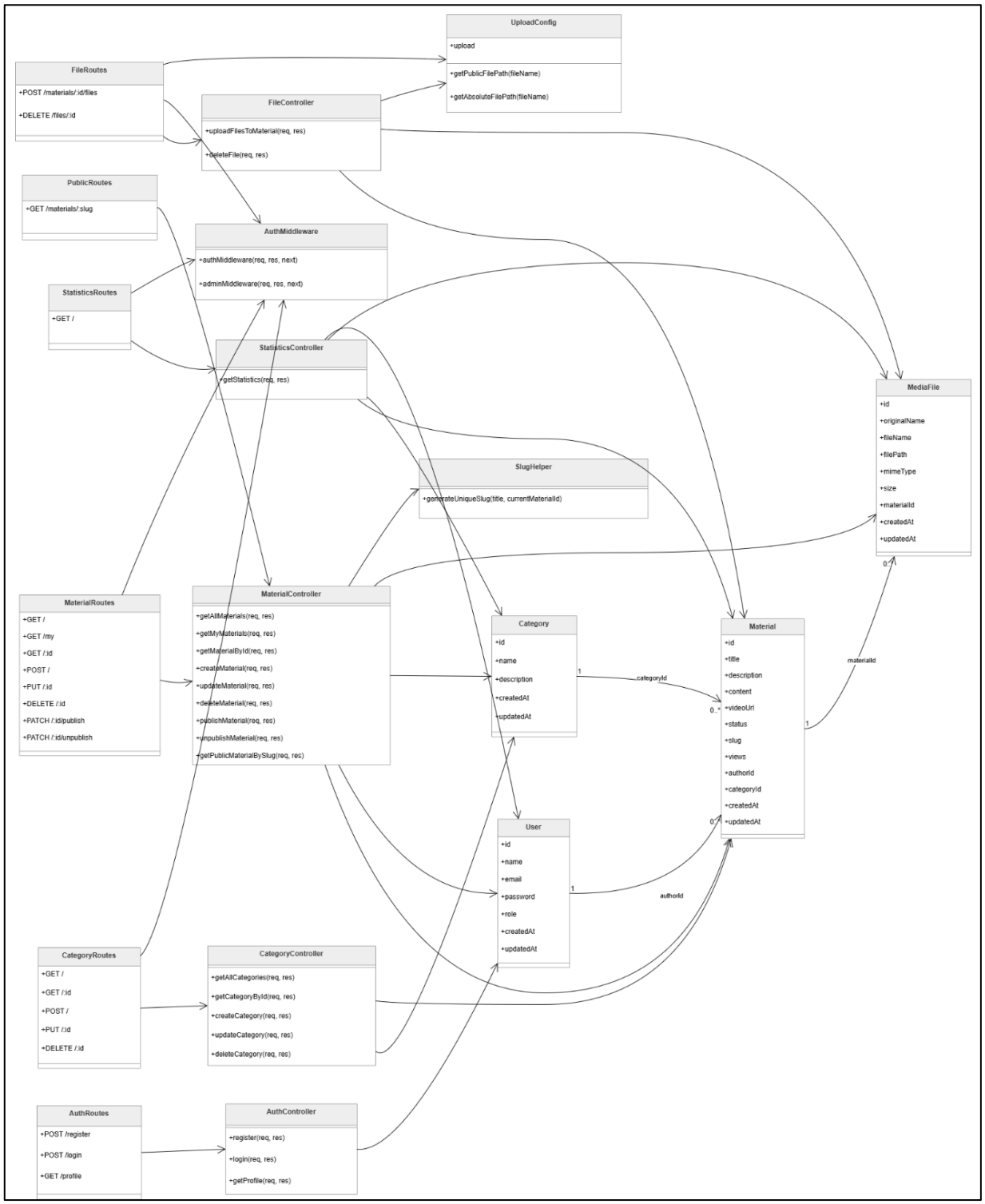


Рисунок 2.6 – Діаграма класів зі сторони сервера

2.5 Діаграма послідовності

Діаграма послідовності використовується для відображення взаємодії між користувачем, клієнтською частиною, серверною частиною та базою даних у

межах конкретного сценарію роботи системи [5, 12]. Вона дозволяє показати не лише перелік учасників процесу, а й порядок обміну повідомленнями між ними.

Діаграму послідовності для прецеденту «Створити навчальний матеріал» наведено на рисунку 2.7.

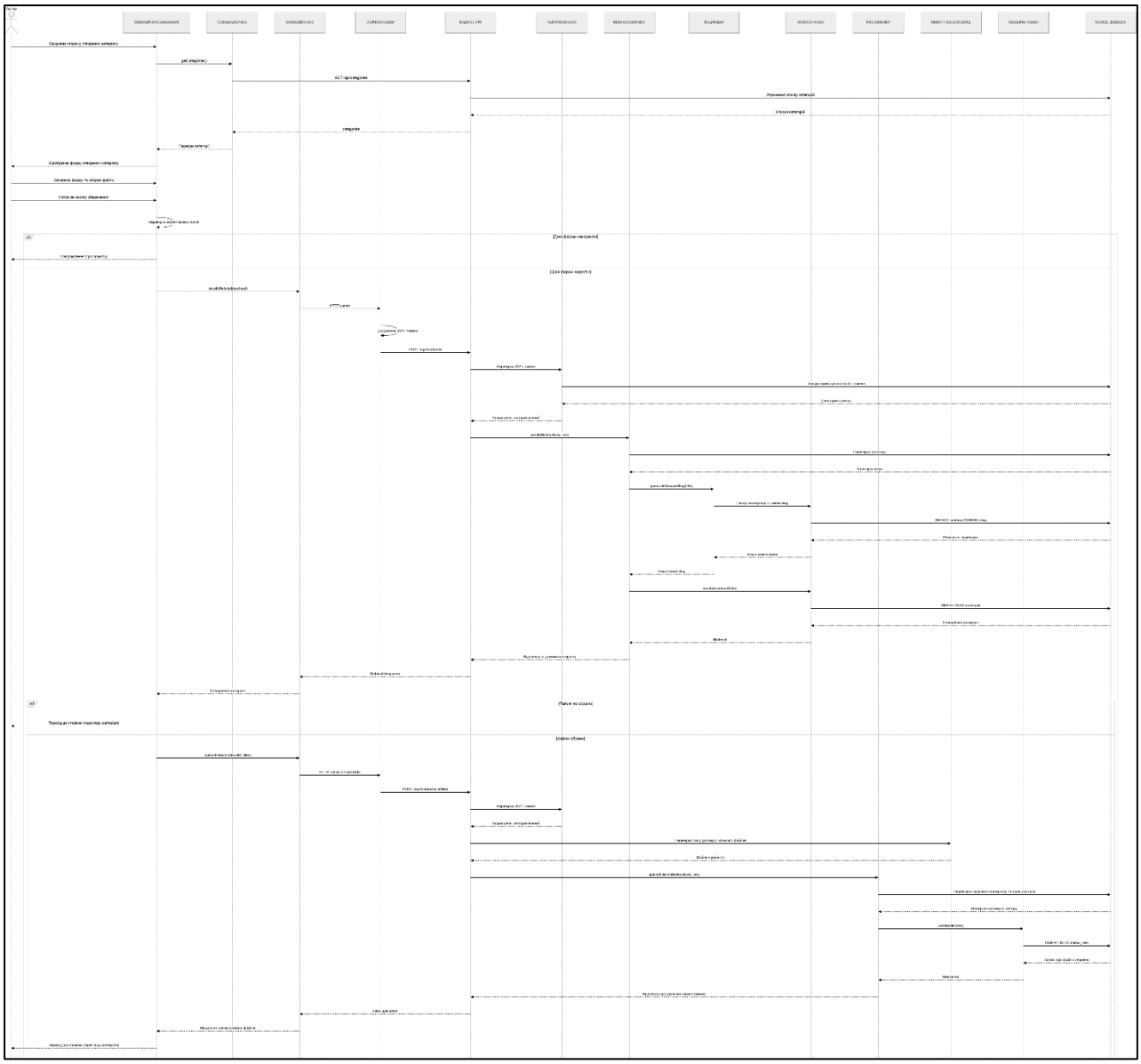


Рисунок 2.7 – Діаграма послідовності процесу створення навчального матеріалу

2.6 Діаграма розгортання

Діаграма розгортання відображає фізичну структуру системи та показує, на яких вузлах розміщуються її основні програмні компоненти [5, 12]. Для

вебплатформи створення та поширення мультимедійних навчальних матеріалів така діаграма є важливою, оскільки система має клієнт-серверну архітектуру та складається з клієнтського застосунку, серверного API, бази даних і файлового сховища.

Діаграму розгортання системи наведено на рисунку 2.8.

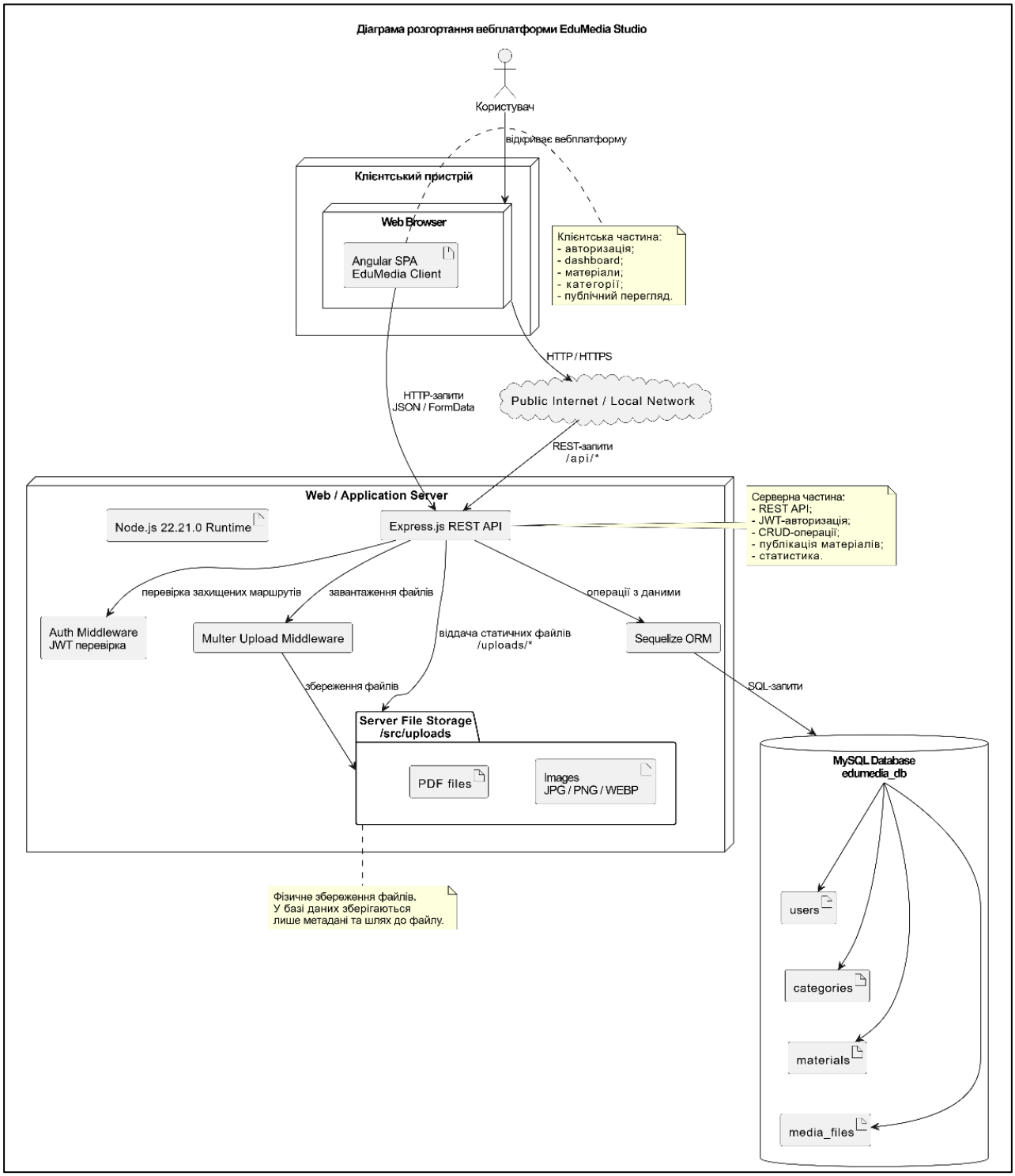


Рисунок 2.8 – Діаграма розгортання

2.7 Висновки до розділу

У другому розділі було виконано проєктування вебплатформи для створення та поширення мультимедійних навчальних матеріалів. На цьому етапі визначено логічну структуру бази даних, основні сутності системи та зв’язки між ними. Побудована ER-модель показує, що центральною сутністю системи є навчальний матеріал, який пов’язується з автором, категорією та прикріпленими мультимедійними файлами.

Також було розроблено UML-діаграми, які описують систему з різних поглядів. Діаграма варіантів використання відображає функціональні можливості гостя, автора й адміністратора. Діаграма діяльності демонструє послідовність створення навчального матеріалу. Діаграми класів зі сторони клієнта та сервера показують структуру основних компонентів Angular-додатка й Express API. Діаграма послідовності деталізує взаємодію між користувачем, клієнтською частиною, сервером і базою даних під час створення матеріалу. Діаграма розгортання відображає фізичну організацію системи та взаємодію між браузером, сервером, базою даних і файловим сховищем.

Отже, результати проєктування дали змогу сформувати цілісне уявлення про структуру, поведінку та технічну організацію вебплатформи. Побудовані моделі є основою для подальшої програмної реалізації клієнтської й серверної частин системи.

3 ПРОГРАМНА РЕАЛІЗАЦІЯ ТА ТЕСТУВАННЯ

3.1 Опис інструментів розробки

Для реалізації вебплатформи було використано набір сучасних вебтехнологій, які забезпечують розділення клієнтської та серверної частин, збереження даних у реляційній базі даних, авторизацію користувачів і роботу з мультимедійними файлами.

Клієнтську частину системи реалізовано на основі фреймворка Angular, який забезпечує побудову компонентів, маршрутизацію, роботу з формами та HTTP-запитами [1]. За його допомогою реалізовано сторінки авторизації, дашборду, списку навчальних матеріалів, форми створення та редагування матеріалу, сторінки перегляду, публічного доступу за посиланням і керування категоріями. Для стилізації інтерфейсу застосовано Tailwind CSS – utility-first фреймворк, що дозволяє будувати адаптивний інтерфейс за допомогою службових класів без написання великих окремих CSS-правил [24].

Серверну частину побудовано на платформі Node.js версії 22.21.0 – кросплатформному середовищі виконання JavaScript. Для реалізації API обрано фреймворк Express.js завдяки його гнучкості: він відповідає за опис маршрутів, підключення middleware, обробку HTTP-запитів, авторизацію, перевірку прав доступу та завантаження файлів [4, 14].

Для збереження даних використано реляційну систему керування базами даних MySQL. У базі зберігаються користувачі, категорії, навчальні матеріали та метадані мультимедійних файлів. Взаємодію з базою даних забезпечує Sequelize ORM, яка дозволяє описувати таблиці у вигляді моделей і виконувати операції створення, читання, оновлення та видалення записів [13, 22].

Авторизацію реалізовано на основі стандарту JSON Web Token [19]. Після входу користувач отримує токен, який зберігається на клієнті та передається із запитами до захищених маршрутів. Сервер перевіряє токен, ідентифікує

користувача та надає доступ відповідно до його ролі. Паролі зберігаються у хешованому вигляді за допомогою бібліотеки bcrypt [15].

Для завантаження мультимедійних файлів використано Multer – middleware для обробки запитів у форматі multipart/form-data [20]. Система перевіряє тип, розмір і кількість файлів, після чого зберігає їх у серверному каталозі, а в базі даних фіксує метадані: назву, шлях, MIME-тип, розмір і зв’язок із навчальним матеріалом.

Підключення до бази даних, порт сервера, секретний ключ JWT та адреса клієнта винесено у файл .env за допомогою dotenv [17]. Додатково підключено cors для коректної взаємодії між клієнтом і сервером на різних портах та helmet для налаштування HTTP-заголовків безпеки [16, 18].

3.2 Основні складові системи

Основним бізнес-процесом розроблюваної вебплатформи є створення мультимедійного навчального матеріалу Отже, основними класами, інтерфейсами та модулями є ті, що забезпечують повний цикл роботи з навчальним матеріалом: створення, редагування, публікацію, завантаження файлів і перегляд матеріалу.

На клієнтській частині основними елементами є Angular-компоненти та сервіси, які забезпечують взаємодію з API [1].

MaterialFormComponent – Angular-компонент, що відповідає за сторінку створення та редагування навчального матеріалу. Він забезпечує відображення форми, завантаження списку категорій, заповнення полів матеріалу, вибір статусу, вибір файлів для завантаження та первинну перевірку введених даних перед надсиланням запиту на сервер.

MaterialService – Angular-сервіс, що відповідає за взаємодію з серверною частиною під час роботи з навчальними матеріалами. Через нього виконуються запити на отримання списку матеріалів, створення, редагування, видалення,

публікацію, зняття з публікації, завантаження файлів і отримання публічного матеріалу за slug.

CategoryService – Angular-сервіс, що використовується для отримання списку категорій навчальних матеріалів. Дані категорій потрібні під час створення або редагування матеріалу, оскільки автор може прив’язати матеріал до певного тематичного розділу.

AuthService – Angular-сервіс, що відповідає за авторизацію користувача, реєстрацію, вихід із системи, збереження JWT-токена та даних поточного користувача. Він забезпечує доступ користувача до захищених сторінок системи.

AuthInterceptor – функціональний HTTP-interceptor, що додає JWT-токен до запитів на захищені API-маршрути. Завдяки цьому серверна частина може перевірити, який користувач виконує запит, і дозволити або заборонити виконання певної дії.

MaterialModel – модель даних навчального матеріалу на клієнтській частині. Вона описує назву, опис, основний текстовий зміст, відеопосилання, статус, slug, кількість переглядів, автора, категорію та прикріплені файли.

CategoryModel – модель категорії навчального матеріалу. Вона містить назву, опис і службові дати створення та оновлення запису.

MediaFileModel – модель мультимедійного файлу. Вона описує оригінальну назву файлу, системну назву, шлях до файлу, MIME-тип, розмір і зв’язок із навчальним матеріалом.

UserModel – модель користувача системи. Вона містить ідентифікатор, ім’я, email і роль користувача. Роль використовується для розмежування доступу між автором і адміністратором.

Зі сторони серверу основними елементами є маршрути, контролери, middleware та моделі [4, 22].

Material.routes.js – файл маршрутів, що визначає API-шляхи для роботи з навчальними матеріалами. Для створення матеріалу використовується маршрут POST /api/materials. Також у цьому файлі описано маршрути для отримання, редагування, видалення, публікації та зняття матеріалу з публікації.

Auth.middleware.js – middleware, що перевіряє JWT-токен користувача та додає дані поточного користувача до запиту. Це потрібно для того, щоб система створювала навчальні матеріали лише для авторизованого користувача та могла перевіряти права доступу до конкретного матеріалу.

Material.controller.js – контролер, що містить основну логіку роботи з навчальними матеріалами. Він відповідає за створення, отримання, редагування, видалення, публікацію, зняття з публікації та публічний перегляд матеріалу за slug.

File.controller.js – контролер, що відповідає за завантаження та видалення мультимедійних файлів. Він перевіряє, чи має користувач право додавати або видаляти файли конкретного навчального матеріалу, після чого створює або видаляє відповідні записи в базі даних.

Upload.config.js – конфігураційний модуль, що налаштовує роботу Multer. Він визначає директорію для збереження файлів, допустимі MIME-типи, максимальний розмір файлу та правила формування унікальної назви файлу.

Slug.helper.js – допоміжний модуль, що відповідає за створення унікального slug для навчального матеріалу. Slug використовується для формування публічного посилання на опублікований матеріал.

Sequelize – ORM, що забезпечує взаємодію серверної частини з базою даних MySQL. Він використовується для опису моделей, налаштування зв’язків між ними та виконання CRUD-операцій.

Material – модель навчального матеріалу в базі даних. Вона зберігає назву, опис, текстовий зміст, відеопосилання, статус, slug, кількість переглядів, автора та категорію.

Category – модель категорії навчальних матеріалів. Вона використовується для групування матеріалів за тематичними напрямами.

MediaFile – модель мультимедійного файлу. Вона зберігає метадані файлу: оригінальну назву, системну назву, шлях до файлу, MIME-тип, розмір і зв’язок із навчальним матеріалом.

User – модель користувача системи. Вона зберігає ім’я, email, хеш пароля

та роль користувача. Через зв’язок із моделлю Material система визначає автора кожного навчального матеріалу.

Нижче наведено фрагмент реалізації створення навчального матеріалу на серверній частині, який демонструє основну бізнес-логіку: перевірку даних, перевірку категорії, формування slug, створення матеріалу та прив’язку його до поточного користувача (рис. 3.1).

```
export const createMaterial = async (req, res) :Promise<...> => {  Show usages
    try {
        const {
            title,
            description,
            content,
            videoUrl,
            categoryId,
            status :string  = 'draft'
        } = req.body;

        if (!title || title.trim() === '') {
            return res.status(400).json({
                message: 'Material title is required'
            });
        }

        if (status && !['draft', 'published'].includes(status)) {
            return res.status(400).json({
                message: 'Invalid material status'
            });
        }

        if (categoryId) {
            const category :Model<any, TModelAttributes>  = await Category.findByPk(categoryId);

            if (!category) {
                return res.status(404).json({
                    message: 'Category not found'
                });
            }
        }
```

Рисунок 3.1 – Фрагмент реалізації створення навчального матеріалу на серверній частині

Повні реалізації ключових компонентів і сервісів фронтенду наведено в додатках А та Б, а серверні обробники й моделі – у додатках В і Г.

3.3 Встановлення та налаштування оточення

Для реалізації вебплатформи були створені окремі клієнтські та серверні частини. Цей підхід відповідає клієнт-серверній архітектурі, оскільки інтерфейс користувача, бізнес-логіка та збереження даних розміщуються в окремих частинах системи.

Загальну структуру проєкту зображено на рисунку 3.2.



Рисунок 3.2 – Структура проєкту

Папка client містить Angular-застосунок, який відповідає за інтерфейс користувача. Папка server містить серверну частину на Node.js та Express.js, моделі Sequelize, контролери, маршрути, middleware, конфігураційні файли та каталог uploads для збереження завантажених файлів.

Серверну частину було створено як окремий Node.js-проєкт. Для цього у папці server виконано ініціалізацію проєкту за допомогою команди npm init -y.

Після цього, за допомогою команди npm install express sequelize mysql2 dotenv cors helmet bcrypt jsonwebtoken multer slugify, було встановлено основні залежності, необхідні для реалізації API, роботи з базою даних, авторизації, завантаження файлів і налаштування серверного застосунку [4, 15–20, 22].

Для полегшення процесу розробки було встановлено dev-залежність nodemon для зручного запуску сервера [21].

Для збереження конфігураційних параметрів у корені папки server було створено файл .env (рис. 3.3) [17].

```
PORT=5000

DB_HOST=localhost
DB_PORT=3306
DB_NAME=edumedia_db
DB_USER=user
DB_PASSWORD=password

JWT_SECRET=my_jwt
JWT_EXPIRES_IN=7d

CLIENT_URL=http://localhost:4200
```

Рисунок 3.3 – Файл .env

У цьому файлі зберігаються порт сервера, параметри підключення до бази даних, секретний ключ для JWT-токенів і адреса клієнтської частини. Такий підхід дозволяє не зберігати конфіденційні налаштування безпосередньо в програмному коді.

Для збереження даних системи використано MySQL. Перед запуском серверної частини було створено базу даних edumedia_db (рис. 3.4) [13].



Рисунок 3.4 – Свторення бази даних edumedia_db

Кодування utf8mb4 використовується для коректного збереження українського тексту, спеціальних символів і різних типів текстових даних, які можуть міститися в навчальних матеріалах [13].

Підключення до бази даних виконується у файлі database.js за допомогою Sequelize (рис. 3.5). Після запуску сервера Sequelize встановлює з’єднання з MySQL і синхронізує моделі з таблицями бази даних [4, 22]. У проєкті використовуються моделі User, Category, Material та MediaFile.

```
import { Sequelize } from 'sequelize';
import dotenv from 'dotenv';

dotenv.config();

export const sequelize :Sequelize = new Sequelize( Show usages
    process.env.DB_NAME,
    process.env.DB_USER,
    process.env.DB_PASSWORD,
    options: {
        host: process.env.DB_HOST,
        port: process.env.DB_PORT,
        dialect: 'mysql',
        logging: false
    }
);
```

Рисунок 3.5 – Файл database.js

Для демонстрації роботи системи було створено seed-файл, який автоматично додає до бази даних адміністратора, тестового автора, базові категорії та кілька навчальних матеріалів (рис. 3.6).

```
const seedDatabase = async () :Promise<void> => { Show usages
    try {
        await sequelize.authenticate();

        console.log('Database connection established.');

        await sequelize.sync( options: { alter: true });

        const adminPassword = await bcrypt.hash( data: 'admin123', salt: 10);
        const authorPassword = await bcrypt.hash( data: 'author123', salt: 10);

        const [admin :Model<any, TModelAttributes> ] = await User.findOrCreate( options: {
            where: {
                email: 'admin@edumedia.com'
            },
            defaults: {
                name: 'System Administrator',
                email: 'admin@edumedia.com',
                password: adminPassword,
                role: 'admin'
            }
        });
```

Рисунок 3.6 – Seed-файл

У файлі package.json було додано команди для запуску серверної частини (рис. 3.7) [14].

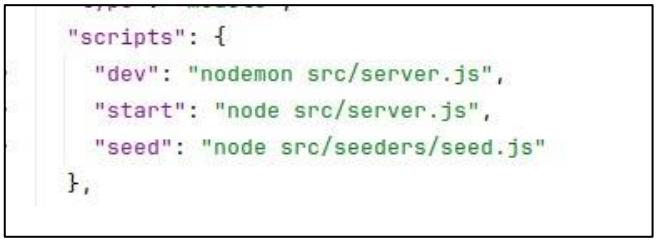


Рисунок 3.7 – Команди запуску в package.json

Команда npm run dev використовується під час розробки, оскільки nodemon автоматично перезапускає сервер після зміни файлів. Команда npm start призначена для звичайного запуску серверної частини. Команда npm run seed використовується для початкового наповнення бази даних тестовими даними [14].

Клієнтську частину було створено за допомогою інструментів Angular CLI. У кореневій папці проєкту виконано команду ng new client --routing --style css [1].

Параметр --routing одразу додає до Angular-застосунку підтримку маршрутизації, а параметр --style css визначає використання CSS для стилізації [1].

Так як для стилізації проєкту використовується Tailwind CSS, то наступною була виконана команда npm install tailwindcss @tailwindcss/postcss postcss [24].

Для підключення Tailwind CSS у корені клієнтської частини було створено файл .postcssrc.json (рис. 3.8) [24].



Рисунок 3.8 – Файл .postcssrc.json

Також у файл src/styles.css було додано імпорт Tailwind CSS та занадно базові стилі сторінки (рис. 3.9) [24].

```
@import "tailwindcss";

body {
  margin: 0;
  font-family: Arial, Helvetica, sans-serif;
  background: #f8fafc;
  color: #0f172a;
}

button,
input,
textarea,
select {
  font: inherit;
}
```

Рисунок 3.9 – Файл styles.css

Для збереження параметрів зв’язку з серверною частиною було створено файл environment.ts (рис. 3.10) [1].

```
export const environment = {  Show usages
  production: false,
  apiUrl: 'http://localhost:5000/api',
  serverUrl: 'http://localhost:5000'
};
```

Рисунок 3.10 – Файл environment.ts

Параметр apiUrl використовується для HTTP-запитів до REST API, а serverUrl – для формування повних посилань на завантажені зображення та PDF-файли.

Для запуску серверної частини потрібно перейти до папки server та виконати команду npm run dev. Після успішного запуску сервер працює за адресою http://localhost:5000 (рис. 3.11) [14].

```
svarog@svarog:~/projects/ed-platform/server$ npm run dev

> edumedia-server@1.0.0 dev
> nodemon src/server.js

[nodemon] 3.1.14
[nodemon] to restart at any time, enter `rs`
[nodemon] watching path(s): *.*
[nodemon] watching extensions: js,mjs,cjs,json
[nodemon] starting `node src/server.js`
◇ injected env (9) from .env // tip: ◈ secrets for agents [www.dotenvx.com]
◇ injected env (0) from .env // tip: ⌘ override existing { override: true }
Database connection has been established successfully.
Server is running on port 5000
```

Рисунок 3.11 – Запуск серверної частини

Клієнтська частина запускається окремо з папки client за допомогою команди ng serve. Після запуску команди Angular-застосунок стає доступний у браузері за адресою http://localhost:4200 (рис. 3.12) [1].

```
svarog@svarog:~/projects/ed-platform/client$ ng serve
Component HMR has been enabled, see https://angular.dev/hmr for more info.
Initial chunk files | Names         |  Raw size
main.js             | main          | 287.56 kB |
polyfills.js        | polyfills     |  89.77 kB |
styles.css          | styles        |  26.54 kB |

                    | Initial total | 403.87 kB

Application bundle generation complete. [1.518 seconds]

Watch mode enabled. Watching for file changes...
NOTE: Raw file sizes do not reflect development server per-request transformations.
  ➜  Local:   http://localhost:4200/
  ➜  press h + enter to show help
```

Рисунок 3.12 – Запуск клієнтської частини

Таким чином, налаштування оточення включає підготовку серверної частини, створення бази даних MySQL, підключення Sequelize, налаштування змінних середовища, створення Angular-застосунку, підключення Tailwind CSS і запуск обох частин системи. Після виконання цих дій вебплатформа готова до роботи в локальному середовищі розробки.

3.4 Опис API

Взаємодія між клієнтською та серверною частинами вебплатформи реалізована за допомогою REST API [4, 10] та поділена на логічні групи: авторизація, категорії, навчальні матеріали, мультимедійні файли, публічний перегляд і статистика.

Для захищених маршрутів використовується JWT-авторизація. Після входу до системи користувач отримує токен, який клієнтська частина додає до HTTP-запитів у заголовку Authorization. Серверна частина перевіряє токен за допомогою middleware та визначає поточного користувача. Якщо токен відсутній або є недійсним, доступ до захищених маршрутів блокується [4, 19].

Маршрути авторизації відповідають за реєстрацію користувача, вхід до системи та отримання даних поточного користувача (табл. 3.1).

Таблиця 3.1 – Маршрути авторизації

Метод	**Маршрут**	**Захищений**	**Призначення**
POST	/api/auth/register	Ні	Реєстрація нового користувача
POST	/api/auth/login	Ні	Вхід користувача до системи
GET	/api/auth/profile	Так	Отримання профілю поточного користувача

Маршрут POST /api/auth/register приймає ім’я, email і пароль користувача. Сервер перевіряє коректність даних, перевіряє унікальність email, хешує пароль і створює нового користувача з роллю автора.

Маршрут POST /api/auth/login використовується для входу до системи. Після успішної перевірки email і пароля сервер формує JWT-токен і повертає його разом із даними користувача.

Маршрут GET /api/auth/profile є захищеним і використовується для отримання інформації про поточного авторизованого користувача.

Приклад запиту для входу до системи наведено на рисунку 3.13. Приклад відповіді сервера – на рисунку 3.14.

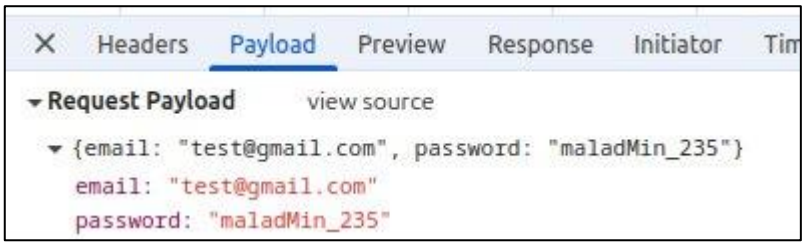


Рисунок 3.13 – Приклад запиту для входу до системи



Рисунок 3.14 – Приклад відповіді сервера для входу до системи

Маршрути категорій використовуються для перегляду, створення, редагування та видалення тематичних розділів навчальних матеріалів. У таблиці 3.2 наведено перелік та призначення цих маршрутів.

Таблиця 3.2 – Маршрути категорій

Метод	**Маршрут**	**Захищений**	**Призначення**
GET	/api/categories	Так	Отримання списку категорій
GET	/api/categories/:id	Так	Отримання однієї категорії
POST	/api/categories	Так	Створення категорії
PUT	/api/categories/:id	Так	Редагування категорії
DELETE	/api/categories/:id	Так	Видалення категорії

Отримання списку категорій доступне авторизованим користувачам і використовується під час створення або редагування навчального матеріалу. Створення, редагування та видалення категорій доступні лише адміністратору. Для цього сервер перевіряє не тільки JWT-токен, а й роль користувача [4].

Маршрути навчальних матеріалів є основною частиною API, оскільки саме вони забезпечують створення, редагування, видалення, публікацію та перегляд матеріалів (табл. 3.3).

Таблиця 3.3 – Маршрути навчальних матеріалів

Метод	**Маршрут**	**Захищений**	**Призначення**
GET	/api/materials	Так	Отримання списку навчальних матеріалів
GET	/api/materials/my	Так	Отримання власних матеріалів користувача
GET	/api/materials/:id	Так	Отримання одного матеріалу за id
POST	/api/materials	Так	Створення навчального матеріалу
PUT	/api/materials/:id	Так	Редагування навчального матеріалу
DELETE	/api/materials/:id	Так	Видалення навчального матеріалу
PATCH	/api/materials/:id/publish	Так	Публікація матеріалу
PATCH	/api/materials/:id/unpublish	Так	Зняття матеріалу з публікації

Маршрут GET /api/materials підтримує пошук, фільтрацію та пагінацію. Це дозволяє клієнтській частині отримувати не всі матеріали одразу, а лише потрібну частину даних. Для цього можуть використовуватися query-параметри search, status, categoryId, page і limit.

Маршрут POST /api/materials використовується для створення навчального матеріалу. Під час створення сервер перевіряє коректність назви, статусу та категорії, після чого формує унікальний slug і створює запис у базі даних.

Приклад тіла запиту для створення матеріалу зображено на рисунку 3.15. Приклад відповіді сервера – на рисунку 3.16.



Рисунок 3.15 – Приклад тіла запиту для створення матеріалу

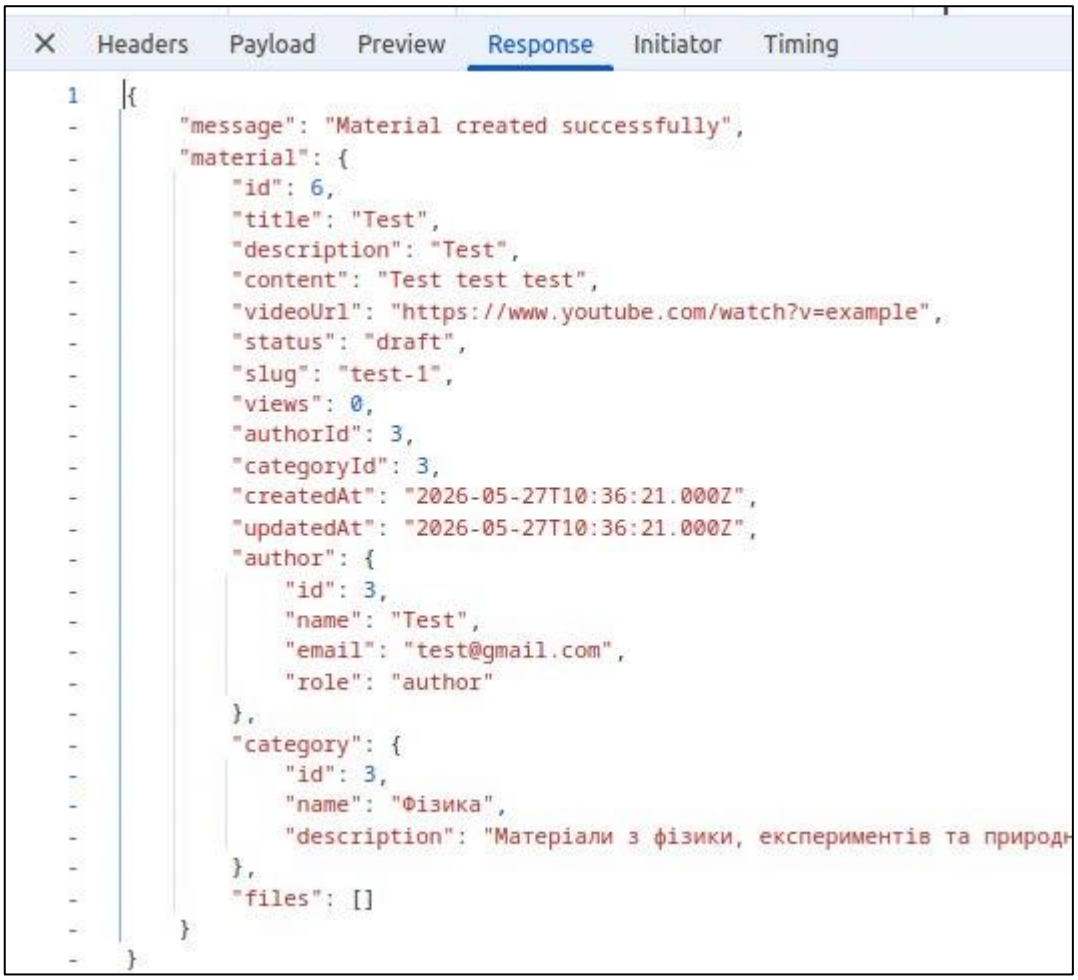


Рисунок 3.16 – Приклад відповіді сервера для створення матеріалу

Маршрути PATCH /api/materials/:id/publish і PATCH /api/materials/:id/unpublish використовуються для зміни статусу матеріалу. Після публікації матеріал стає доступним за публічним посиланням.

Маршрути файлів відповідають за завантаження та видалення зображень і PDF-документів, які прикріплюються до навчальних матеріалів (табл. 3.4).

Таблиця 3.4 – Маршрути файлів

Метод	**Маршрут**	**Захищений**	**Призначення**
POST	/api/materials/:id/files	Так	Завантаження файлів до матеріалу
DELETE	/api/files/:id	Так	Видалення прикріпленого файлу

Завантаження файлів виконується у форматі multipart/form-data [20]. Сервер перевіряє, чи існує навчальний матеріал, чи має користувач право

додавати до нього файли, а також перевіряє тип, розмір і кількість файлів. У системі дозволено завантажувати зображення та PDF-файли.

Після успішного завантаження фізичний файл зберігається у серверному каталозі uploads, а в базі даних створюється запис із метаданими файлу: оригінальною назвою, системною назвою, шляхом, MIME-типом, розміром і зв'язком із навчальним матеріалом [14, 22].

Приклад відповіді сервера після завантаження файлу наведено на рисунку 3.17.


```
            "name": "STEM",
            "description": "Інтегровані навчальні матеріали з науки, тех
        },
        "files": [
            {
                "id": 1,
                "originalName": "finance-report-2026-04-30-2026-05-31 (1
                "fileName": "ab7d3270-4620-47e2-8d96-45c2a9a974e0.pdf",
                "filePath": "/uploads/ab7d3270-4620-47e2-8d96-45c2a9a974
                "mimeType": "application/pdf",
                "size": 67699,
                "materialId": 5,
                "createdAt": "2026-05-25T12:09:46.000Z",
                "updatedAt": "2026-05-25T12:09:46.000Z"
            }
        ]
    }
```


Рисунок 3.17 – Приклад відповіді сервера після завантаження файлу

Публічний API використовується для перегляду опублікованого навчального матеріалу без авторизації (табл. 3.5) [4, 19].

Таблиця 3.5 – Публічний API

Метод	**Маршрут**	**Захищений**	**Призначення**
GET	/api/public/materials/:slug	Ні	Публічний перегляд матеріалу за slug

Маршрут GET /api/public/materials/:slug шукає матеріал за унікальним slug і перевіряє, чи має він статус published. Якщо матеріал опублікований, сервер повертає його дані разом з автором, категорією та прикріпленими файлами. Також під час відкриття публічного матеріалу збільшується кількість його переглядів.

Маршрут статистики використовується для відображення даних на дашборді (табл. 3.6).

Таблиця 3.6 – Маршрут статистики

Метод	**Маршрут**	**Захищений**	**Призначення**
GET	/api/statistics	Так	Отримання статистики системи

Для автора сервер повертає статистику лише за його матеріалами. Для адміністратора може повертатися загальна статистика системи. У відповідь входять кількість матеріалів, кількість чернеток, кількість опублікованих матеріалів, кількість категорій, кількість завантажених файлів, загальна кількість переглядів, останні матеріали та найпопулярніші матеріали.

Приклад відповіді сервера зображено на рисунку 3.18.


```
{
    "statistics": {
        "totalMaterials": 2,
        "draftMaterials": 1,
        "publishedMaterials": 1,
        "totalCategories": 2,
        "totalFiles": 1,
        "totalViews": 1,
        "popularMaterials": [
            {
                "id": 5,
                "title": "Test",
                "description": "test test test",
                "status": "published",
                "slug": "test",
                "views": 1,
                "createdAt": "2026-05-25T12:09:46.000Z",
                "author": {
                    "id": 3,
                    "name": "Test",
                    "email": "test@gmail.com",
                    "role": "author"
                },
                "category": {
                    "id": 4,
                    "name": "STEM"
                }
            },
            {
```


Рисунок 3.18 – Приклад відповіді сервера

Реалізоване API забезпечує всі основні функції вебплатформи: авторизацію користувачів, керування категоріями, створення та редагування

навчальних матеріалів, публікацію матеріалів, завантаження мультимедійних файлів, публічний перегляд і отримання статистики.

3.5 Тестування проєкту

Для перевірки коректності бізнес-логіки, роботи форм та взаємодії клієнтської частини з API було проведено unit-тестування з використанням вбудованих засобів Angular CLI – бібліотек Jasmine та Karma [1].

Тестування зосереджено на компоненті MaterialFormComponent, який відповідає за створення та редагування навчальних матеріалів. Перевірялися такі сценарії: заборона збереження матеріалу без назви, коректне формування запиту, завантаження категорій, підстановка даних при редагуванні та завантаження файлів після створення запису.

На рисунку 3.19 наведено фрагмент unit-тесту, який перевіряє бізнес-правило щодо обов’язкової назви навчального матеріалу.

```
it('should not submit form when title is empty', () => {
  createComponent();

  component.form.patchValue({
    title: '',
    description: 'Опис матеріалу',
    content: 'Текст матеріалу',
    videoUrl: '',
    categoryId: '1',
    status: 'draft'
  });

  component.submit();

  expect(component.form.invalid).toBeTrue();
  expect(component.errorMessage()).toBe(
    'Перевірте правильність заповнення форми'
  );
  expect(materialService.createMaterial).not.toHaveBeenCalled();
});
```

Рисунок 3.19 – Фрагмент unit-тесту компонента MaterialFormComponent

Самі тести запускаються за допомогою команди ng test [1]. Після запуску Karma відображає результати у консолі (рис. 3.20) та браузері (рис. 3.21) [1].

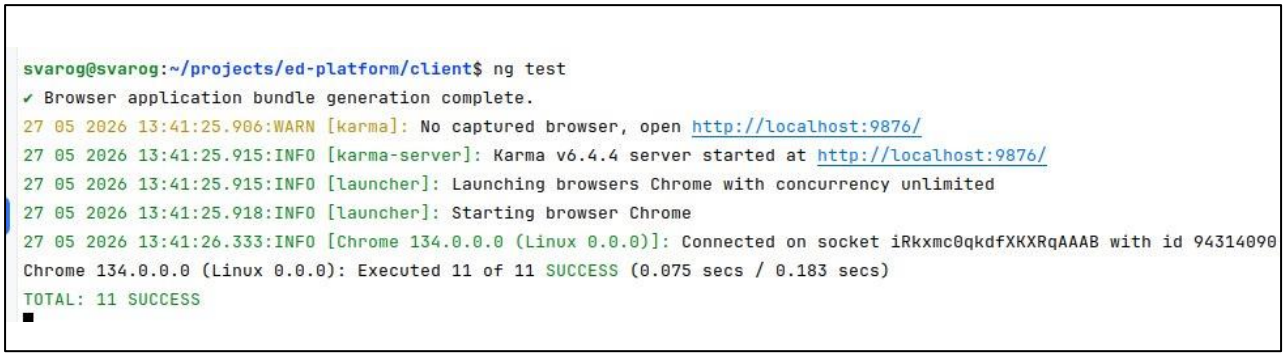


Рисунок 3.20 – Запуск та результат у консолі

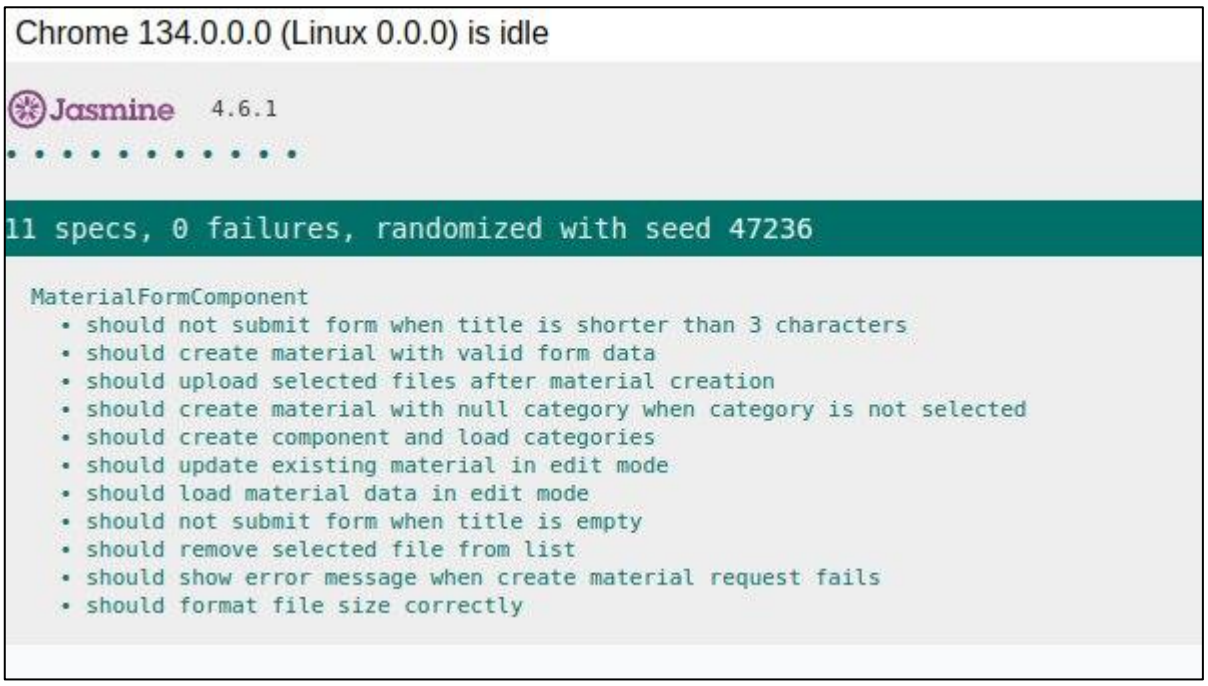


Рисунок 3.21 – Результат у браузері

Проведене тестування дозволяє перевірити коректність важливих бізнес-правил і зменшити ризик появи помилок під час створення мультимедійних навчальних матеріалів. Зокрема, unit-тести підтверджують, що система не дозволяє створити матеріал без назви, коректно формує запит на створення матеріалу, завантажує категорії, підставляє дані під час редагування та виконує завантаження файлів після створення матеріалу.

Детально ознайомитися із кодом тесту можна в додатку Д.

3.6 Керівництво користувача

3.6.1 Вхід та реєстрація

Для початку роботи користувач відкриває сторінку входу. На цій сторінці потрібно ввести email і пароль, після чого натиснути кнопку «Увійти». Якщо введені дані є правильними, система авторизує користувача та відкриває головну сторінку – дашборд. На рисунку 3.22 наведено сторінку входу до системи.

Якщо користувач ще не має облікового запису, він може перейти на сторінку реєстрації. Для створення облікового запису потрібно вказати ім’я, email і пароль. Після успішної реєстрації користувач отримує роль автора та може створювати навчальні матеріали. На рисунку 3.23 наведено сторінку реєстрації нового користувача.



Рисунок 3.22 – Сторінка входу до системи

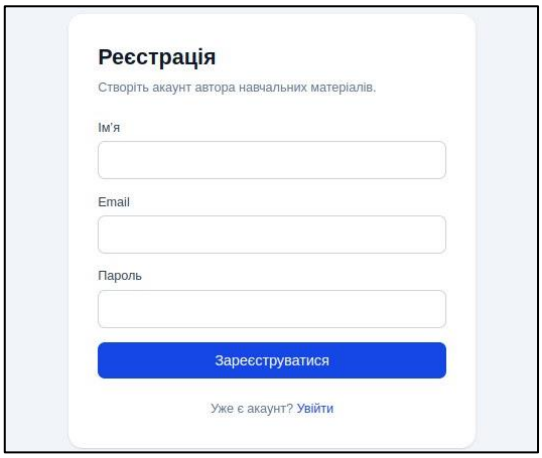


Рисунок 3.23 – Сторінка реєстрації

3.6.2 Перегляд дашборду

Після входу до системи користувач переходить на сторінку дашборду. На цій сторінці відображається узагальнена статистика роботи з навчальними матеріалами: загальна кількість матеріалів, кількість опублікованих матеріалів, кількість чернеток, кількість категорій, завантажених файлів і переглядів. Також на дашборді відображаються останні створені матеріали та найпопулярніші матеріали за кількістю переглядів. Це дозволяє користувачу швидко оцінити стан власного навчального контенту. На рисунку 3.24 наведено сторінку дашборду.

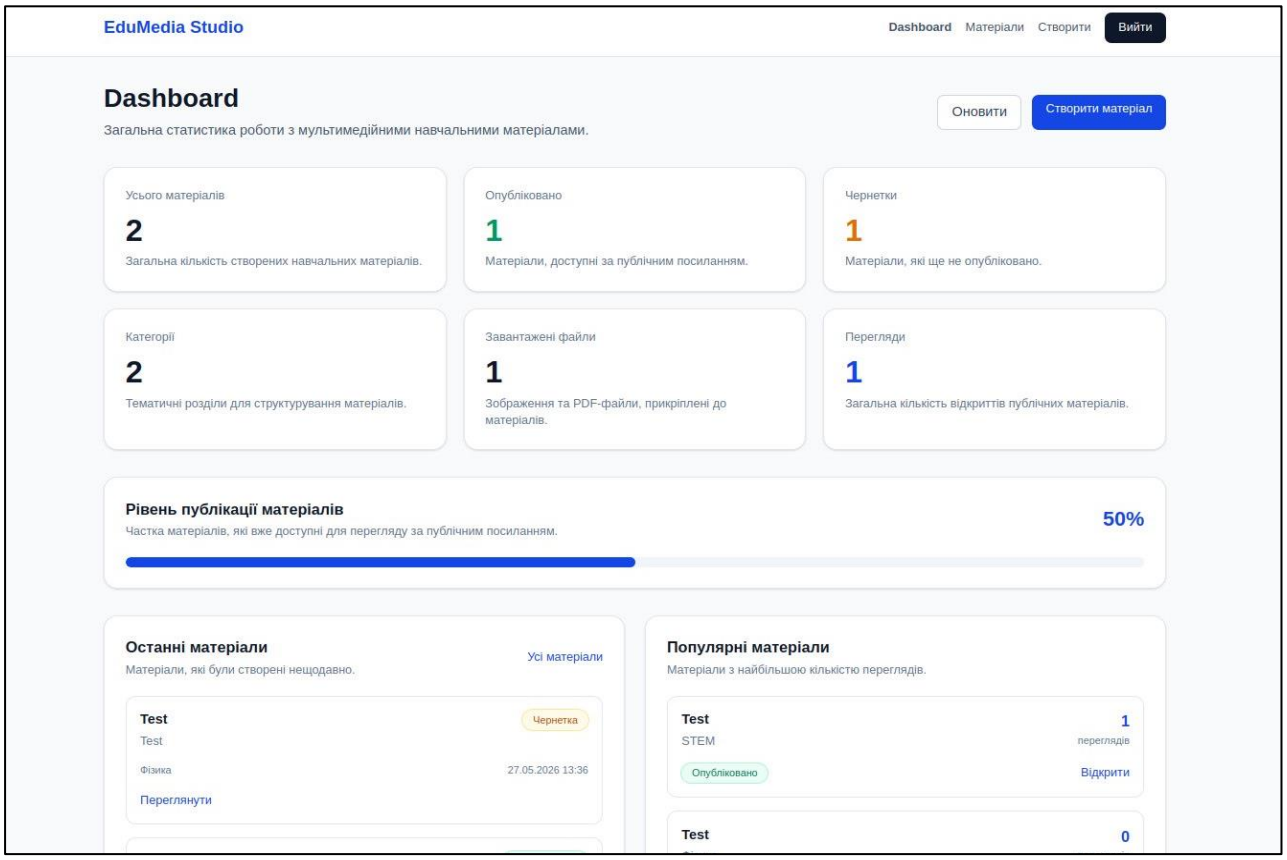


Рисунок 3.24 – Сторінка дашборду

3.6.3 Перегляд списку навчальних матеріалів

Для переходу до списку навчальних матеріалів користувач обирає пункт меню «Матеріали». На цій сторінці відображаються створені матеріали у вигляді

карток. Для кожного матеріалу показано назву, короткий опис, категорію, автора, дату створення, статус і кількість переглядів.

Користувач може виконувати пошук за назвою або описом матеріалу, фільтрувати матеріали за статусом і категорією, а також переходити до перегляду, редагування або публікації матеріалу.

На рисунку 3.25 наведено сторінку списку навчальних матеріалів.

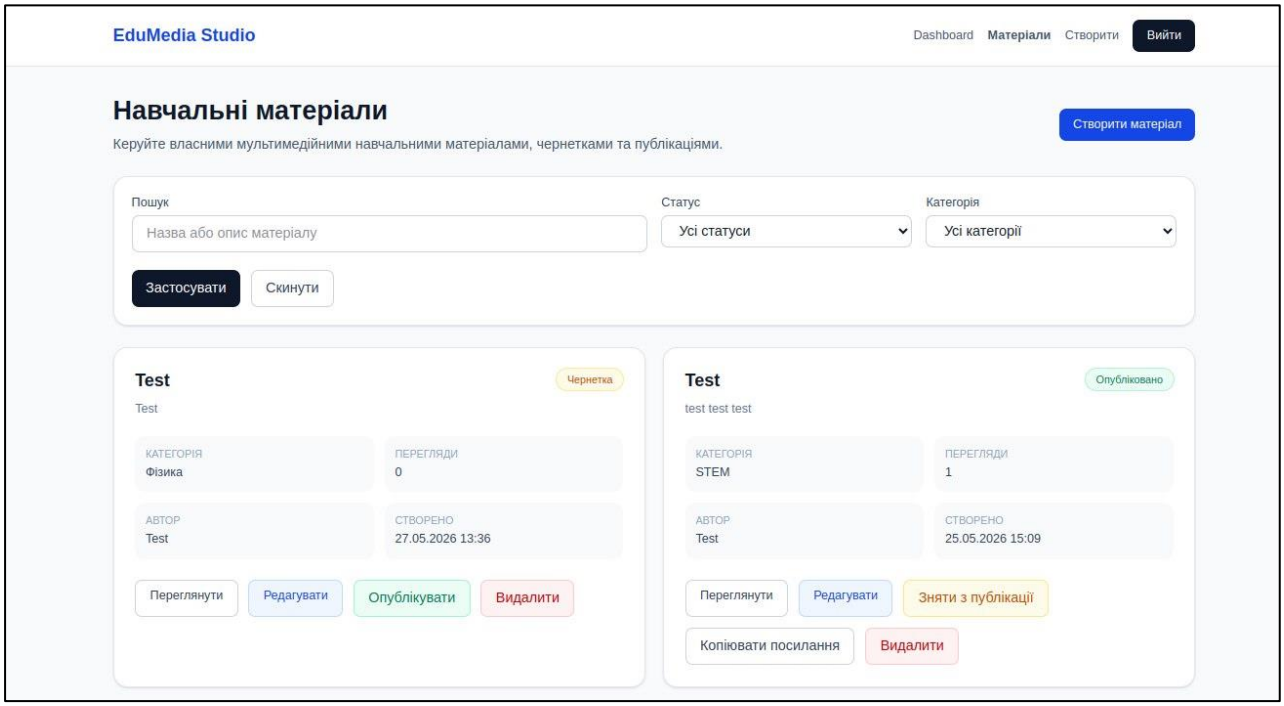


Рисунок 3.25 – Сторінка списку навчальних матеріалів

3.6.4 Створення навчального матеріалу

Для створення нового навчального матеріалу користувач натискає кнопку «Створити матеріал». Після цього відкривається форма створення матеріалу.

У формі потрібно вказати назву матеріалу, короткий опис, основний текстовий зміст, за потреби додати відеопосилання, обрати категорію та визначити статус матеріалу.

Матеріал може бути збережений як чернетка або одразу створений зі статусом опублікованого.

Назва матеріалу є обов’язковим полем. Якщо користувач намагається зберегти матеріал без назви, система виводить повідомлення про помилку та не надсилає запит на сервер.

На рисунку 3.26 наведено форму створення навчального матеріалу.

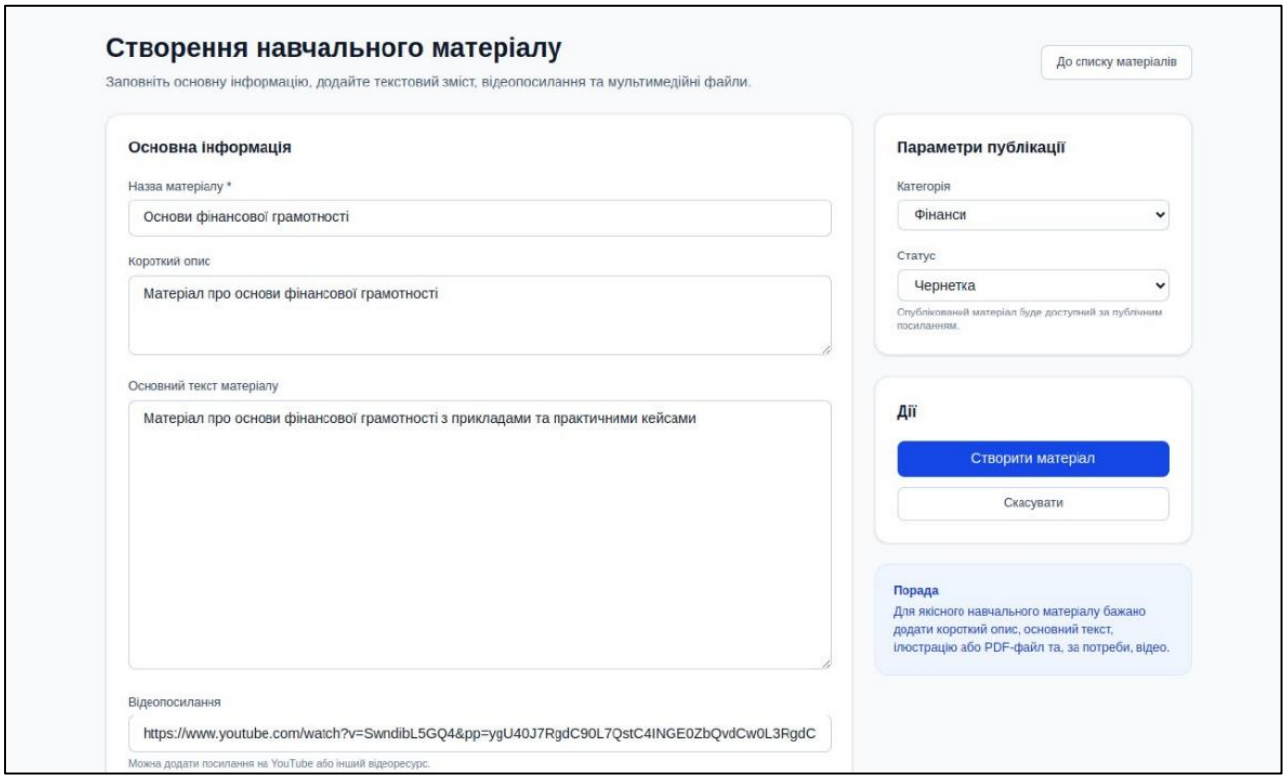


Рисунок 3.26 – Форма створення навчального матеріалу

Під час створення або редагування навчального матеріалу користувач може додати мультимедійні файли.

Система підтримує завантаження зображень і PDF-документів. Для цього у формі матеріалу потрібно натиснути кнопку вибору файлів і обрати один або декілька файлів на комп’ютері.

Після збереження матеріалу файли завантажуються на сервер і прикріплюються до відповідного навчального матеріалу. Якщо файл має недозволений тип або перевищує допустимий розмір, система повідомляє користувача про помилку.

На рисунку 3.27 наведено приклад додавання мультимедійних файлів до матеріалу.

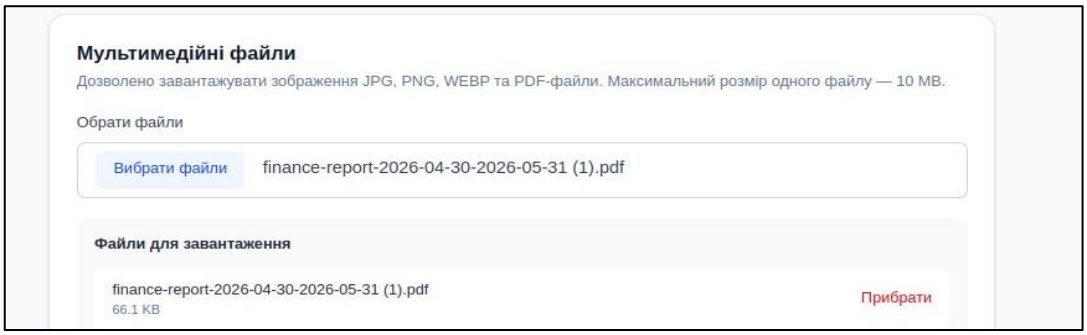


Рисунок 3.27 – Додавання мультимедійних файлів до матеріалу

3.6.5 Перегляд навчального матеріалу

Після створення або вибору матеріалу зі списку відкривається сторінка детального перегляду. На цій сторінці відображаються назва матеріалу, опис, основний текст, статус, категорія, автор, кількість переглядів, відеоматеріал і прикріплені файли. Якщо до матеріалу додано зображення, вони відображаються безпосередньо на сторінці. PDF-файли можна відкрити за посиланням. Якщо матеріал має відеопосилання на підтримуваний відеоресурс, система відображає відео на сторінці або надає посилання для відкриття. На рисунку 3.28 наведено сторінку перегляду навчального матеріалу.

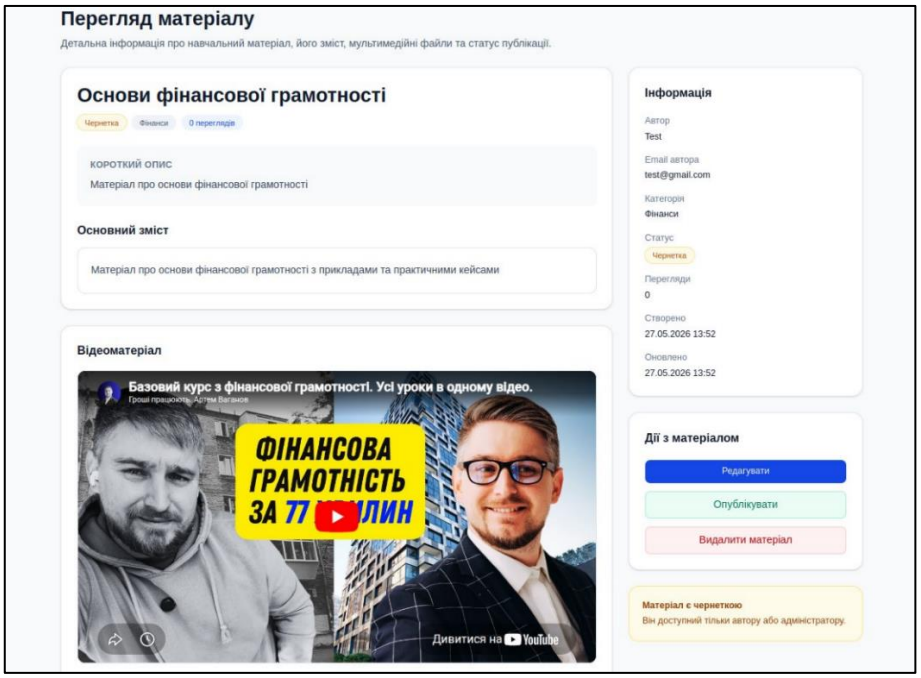


Рисунок 3.28 – Сторінка перегляду навчального матеріалу

3.6.6 Редагування та видалення матеріалу

Для зміни даних матеріалу користувач натискає кнопку «Редагувати». Після цього відкривається форма редагування, у якій уже заповнено поточні дані матеріалу. Користувач може змінити назву, опис, основний текст, відеопосилання, категорію, статус, додати нові файли або видалити раніше прикріплені.

Для видалення матеріалу потрібно натиснути кнопку «Видалити матеріал» і підтвердити дію.

Після підтвердження матеріал видаляється із системи та більше не відображається у списку.

На рисунку 3.29 наведено форму редагування навчального матеріалу.

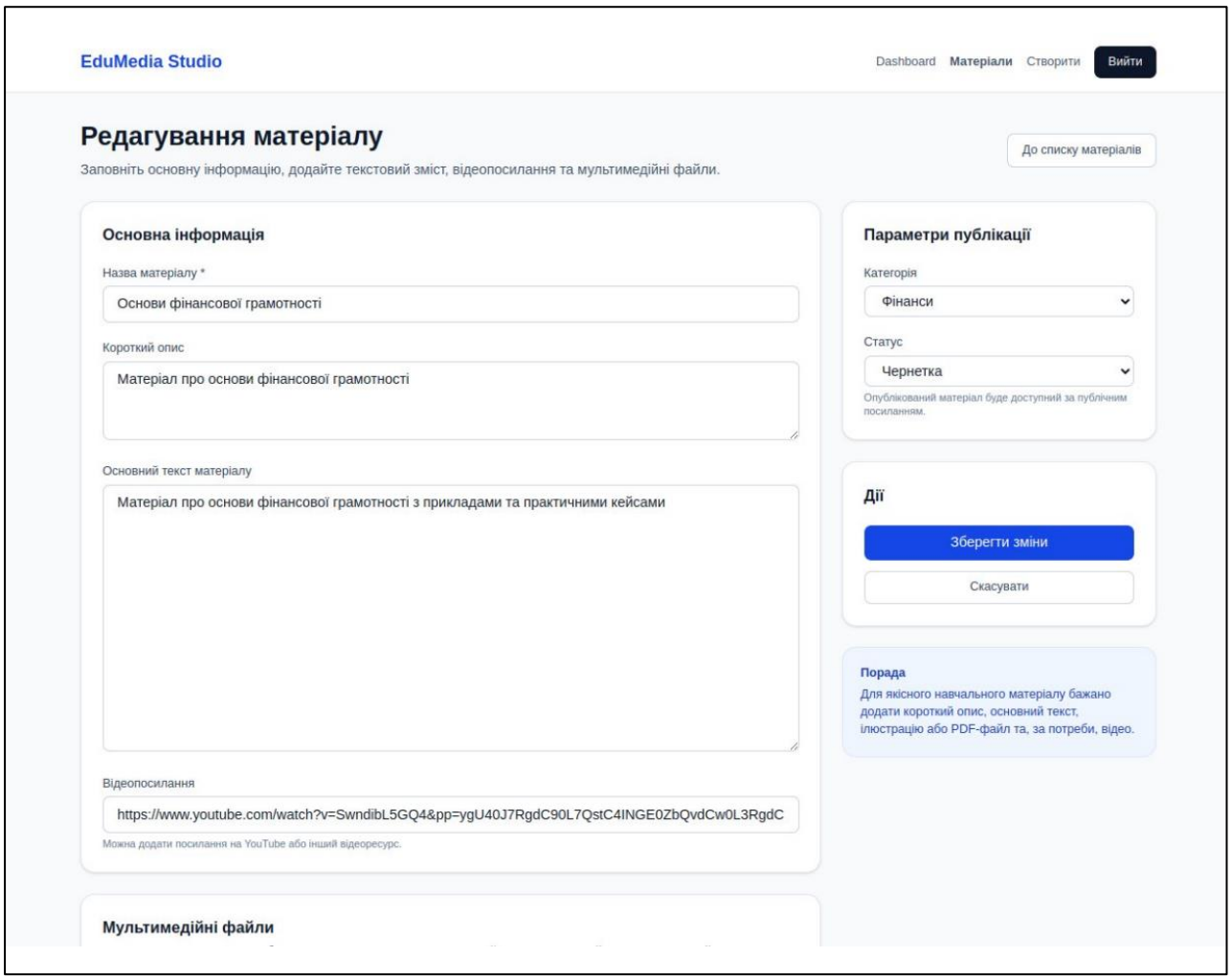


Рисунок 3.29 – Форма редагування навчального матеріалу

3.6.7 Публікація навчального матеріалу

Матеріал може мати один із двох статусів: «Чернетка» або «Опубліковано». Чернетка доступна лише авторизованому користувачу, який має право працювати з цим матеріалом. Опублікований матеріал доступний за публічним посиланням.

Для публікації потрібно натиснути кнопку «Опублікувати». Після цього система змінює статус матеріалу та формує публічне посилання. Користувач може скопіювати це посилання та передати його іншим особам.

Якщо потрібно припинити публічний доступ, користувач натискає кнопку «Зняти з публікації». Після цього матеріал знову стає чернеткою та не відкривається за публічним посиланням.

На рисунку 3.30 наведено приклад публікації навчального матеріалу.

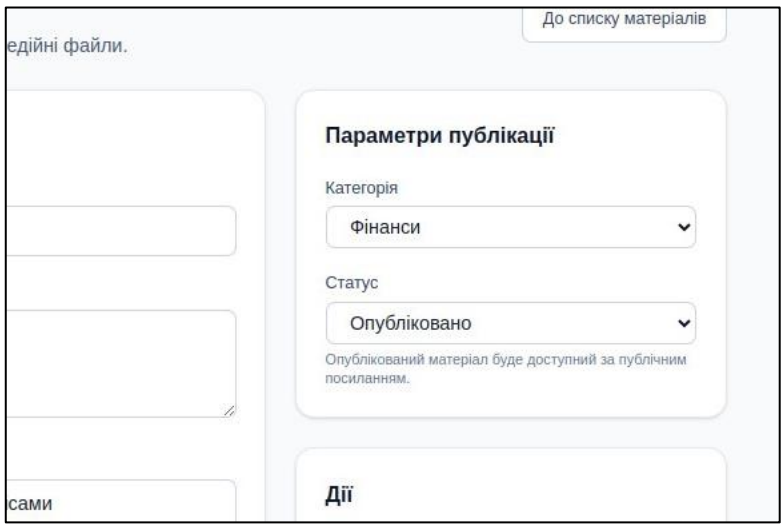


Рисунок 3.30 – Публікація навчального матеріалу

3.6.8 Публічний перегляд матеріалу

Публічний перегляд дозволяє відкрити опублікований навчальний матеріал без входу до системи. Для цього користувач переходить за публічним посиланням, яке було сформоване після публікації.

На публічній сторінці відображається назва матеріалу, опис, основний текст, автор, категорія, відеоматеріал і прикріплені файли. Також користувач може скопіювати поточне посилання на матеріал.

Якщо матеріал не опублікований або був знятий із публікації, публічна сторінка не відображає його вміст.

На рисунку 3.31 наведено сторінку публічного перегляду навчального матеріалу.

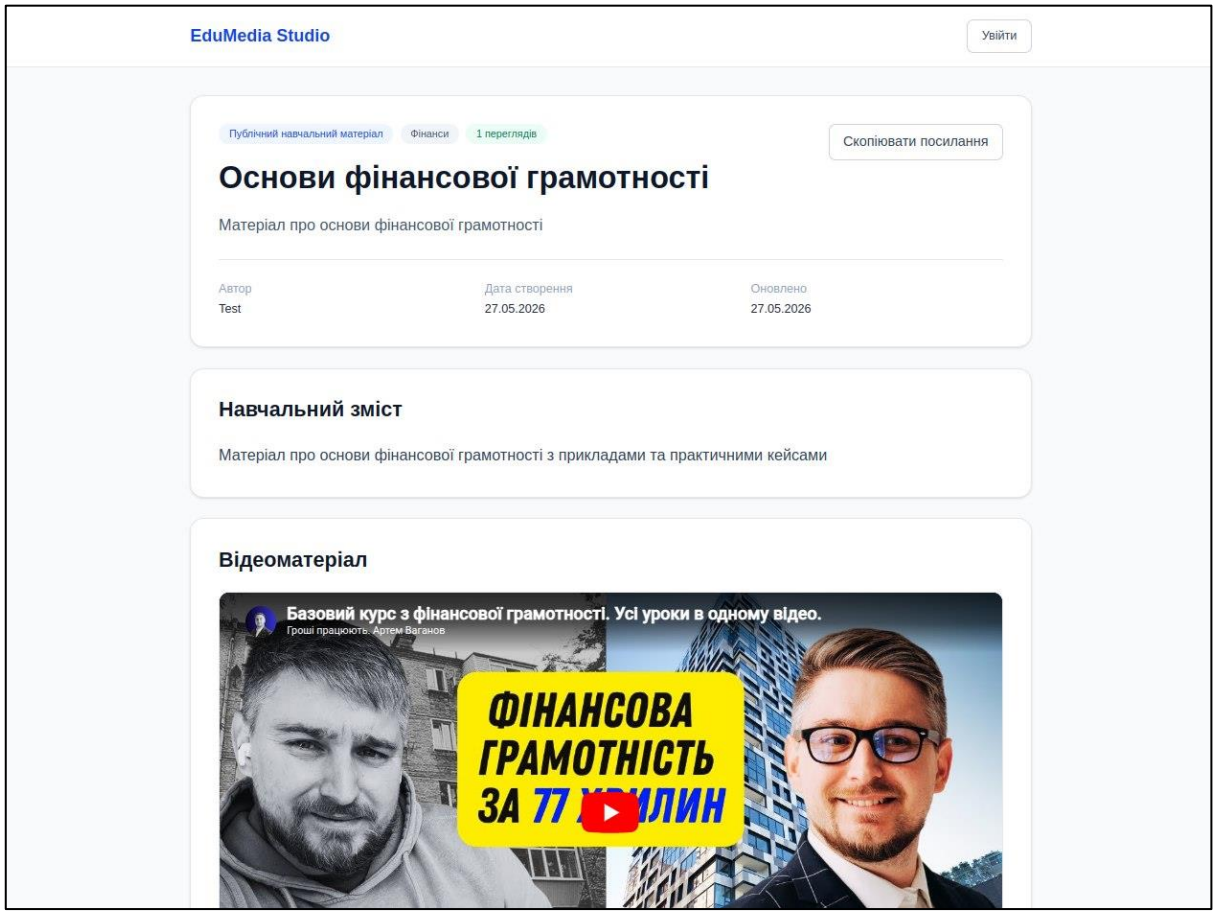


Рисунок 3.31 – Сторінка публічного перегляду навчального матеріалу

3.6.9 Керування категоріями

Адміністратор має доступ до сторінки керування категоріями. На цій сторінці він може переглядати список категорій, створювати нові категорії, редагувати їх назви й описи, а також видаляти категорії.

Категорії використовуються для структурування навчальних матеріалів за предметами або тематичними напрямами. Під час створення або редагування матеріалу автор може обрати одну з доступних категорій.

На рисунку 3.32 наведено сторінку керування категоріями.

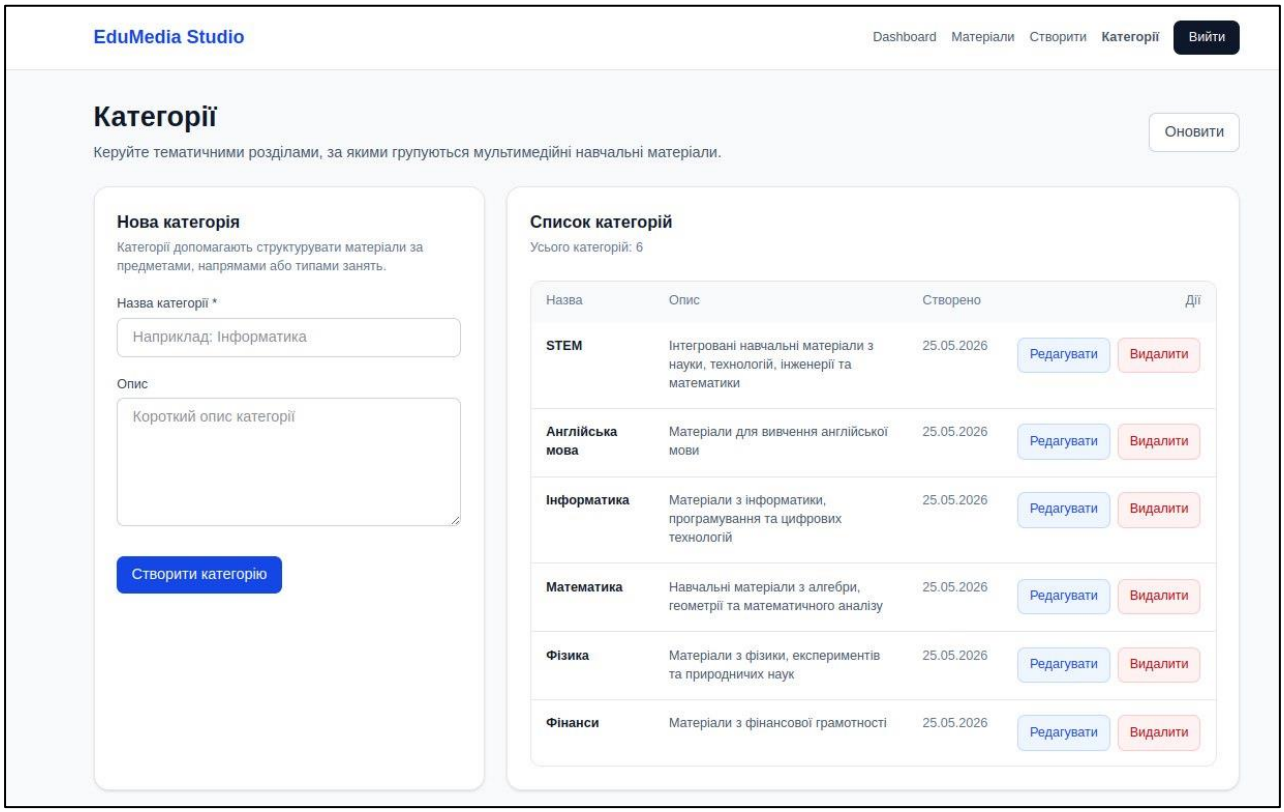


Рисунок 3.32 – Сторінка керування категоріями

Якщо користувач не має ролі адміністратора, він не може відкрити сторінку категорій. У такому випадку система перенаправляє його на дашборд.

ВИСНОВКИ

У ході виконання кваліфікаційної роботи було розроблено вебплатформу для створення та поширення мультимедійних навчальних матеріалів. Система дозволяє користувачу створювати обліковий запис, входити до системи, створювати навчальні матеріали, редагувати їх, додавати текстовий зміст, короткий опис, відеопосилання, зображення та PDF-файли.

Відповідно до мети кваліфікаційної роботи реалізовано вебплатформу з використанням таких технологій:

– Angular та Tailwind CSS для клієнтської частини;
– Node.js та Express.js для серверної частини й REST API;
– MySQL для збереження даних;
– Sequelize ORM для взаємодії з базою даних;
– JWT для захисту API-маршрутів;
– Multer для завантаження мультимедійних файлів;
– bcrypt для хешування паролів користувачів.

За поставленими задачами виконано такі етапи:

– сформовано функціональні та нефункціональні вимоги до системи;
– проведено аналіз предметної області;
– розглянуто аналоги та виконано їх порівняння;
– спроєктовано базу даних і побудовано ER-модель;
– побудовано UML-діаграми варіантів використання, діяльності, класів, послідовності та розгортання;
– реалізовано вебплатформу для створення та поширення мультимедійних навчальних матеріалів;
– виконано unit-тестування компонента створення навчального матеріалу;
– підготовлено керівництво користувача.

ПЕРЕЛІК ПОСИЛАНЬ

1. Angular Developer Documentation. URL: https://angular.dev/overview (дата звернення: 19.03.2026).

2. Canva for Education. URL: https://www.canva.com/education/ (дата звернення: 15.02.2026).

3. European Commission. Digital education content: guidelines for teachers and interoperability framework. URL: https://education.ec.europa.eu/focus-topics/digital-education/actions/plan/digital-education-content-guidelines-and-framework (дата звернення: 01.03.2026).

4. Express.js Guide. URL: https://expressjs.com/ (дата звернення: 19.03.2026).

5. Fowler M. UML Distilled: A Brief Guide to the Standard Object Modeling Language. Addison-Wesley Professional, 2003. 208 p.

6. Genially. URL: https://genially.com/ (дата звернення: 12.02.2026).

7. Google Classroom Help. URL: https://support.google.com/edu/classroom?sjid=14487554185294371778-EU#topic=10298088 (дата звернення: 17.02.2026).

8. ISO/IEC/IEEE 29148:2018. Systems and software engineering – Life cycle processes – Requirements engineering. URL: https://www.iso.org/standard/72089.html (дата звернення: 12.03.2026).

9. Mayer R. E. Multimedia Learning. Cambridge University Press, 2001. 210 p.

10. MDN Web Docs. *REST*. URL: https://developer.mozilla.org/en-US/docs/Glossary/REST (дата звернення: 21.04.2026).

11. Moodle.org. URL: https://moodle.org/ (дата звернення: 08.02.2026).

12. Mrzyglocki F. UML Summarized: Key Concepts and Diagrams for Software Engineers, Architects, and Designers. Phillip Mrzyglocki, 2023. 59 p.

13. MySQL Documentation. URL: https://dev.mysql.com/doc/ (дата звернення: 15.03.2026).

14. Node.js Documentation. URL: https://nodejs.org/docs/latest/api/ (дата звернення: 19.03.2026).

15. Npm bcrypt. URL: https://www.npmjs.com/package/bcrypt (дата звернення: 21.03.2026).

16. Npm cors. URL: https://www.npmjs.com/package/cors (дата звернення: 22.04.2026).

17. Npm dotenv. URL: https://www.npmjs.com/package/dotenv (дата звернення: 22.04.2026).

18. Npm helmet. URL: https://www.npmjs.com/package/helmet (дата звернення: 24.04.2026).

19. Npm jsonwebtoken. URL: https://www.npmjs.com/package/jsonwebtoken (дата звернення: 21.03.2026).

20. Npm multer. URL: https://www.npmjs.com/package/multer (дата звернення: 21.03.2026).

21. Npm nodemon. URL: https://www.npmjs.com/package/nodemon (дата звернення: 28.04.2026).

22. Sequelize Documentation. URL: https://sequelize.org/ (дата звернення: 16.03.2026).

23. SWEBOK Guide V4.0. Guide to the Software Engineering Body of Knowledge. URL: https://www.computer.org/education/bodies-of-knowledge/software-engineering (дата звернення: 14.03.2026).

24. Tailwind CSS Docs. URL: https://tailwindcss.com/docs/installation/framework-guides/angular (дата звернення: 25.04.2026).

25. Recommendation on Open Educational Resources (OER). *UNESCO*. URL: https://www.unesco.org/en/legal-affairs/recommendation-open-educational-resources-oer (дата звернення: 23.02.2026).

ДОДАТОК А

Angular MaterialFormComponent

```
import {
  Component,
  DestroyRef,
  OnInit,
  computed,
  inject,
  signal
} from '@angular/core';

import {
  FormBuilder,
  ReactiveFormsModule,
  Validators
} from '@angular/forms';

import {
  ActivatedRoute,
  Router,
  RouterLink
} from '@angular/router';

import { takeUntilDestroyed } from '@angular/core/rxjs-interop';

import { environment } from '../../../../environments/environment';

import {
  Material,
  MaterialStatus,
  MediaFile
} from '../../../core/models/material.model';

import { Category } from '../../../core/models/category.model';

import { MaterialService } from '../../../core/services/material.service';
import { CategoryService } from '../../../core/services/category.service';

@Component({
  selector: 'app-material-form',
  standalone: true,
  imports: [
    ReactiveFormsModule,
    RouterLink
  ],
  templateUrl: './material-form.component.html'
})
export class MaterialFormComponent implements OnInit {
  private readonly fb = inject(FormBuilder);
```

```
private readonly route = inject(ActivatedRoute);
private readonly router = inject(Router);
private readonly destroyRef = inject(DestroyRef);

private readonly materialService = inject(MaterialService);
private readonly categoryService = inject(CategoryService);

readonly categories = signal<Category[]>([]);
readonly material = signal<Material | null>(null);
readonly selectedFiles = signal<File[]>([]);

readonly isLoading = signal(false);
readonly isSaving = signal(false);
readonly errorMessage = signal('');
readonly successMessage = signal('');

readonly materialId = signal<number | null>(null);

readonly isEditMode = computed(() => this.materialId() !== null);

readonly form = this.fb.nonNullable.group({
  title: ['', [Validators.required, Validators.minLength(3)]],
  description: [''],
  content: [''],
  videoUrl: [''],
  categoryId: [''],
  status: this.fb.nonNullable.control<MaterialStatus>('draft')
});

ngOnInit(): void {
  this.loadCategories();

  const id = this.route.snapshot.paramMap.get('id');

  if (id) {
    this.materialId.set(Number(id));
    this.loadMaterial(Number(id));
  }
}

loadCategories(): void {
  this.categoryService
    .getCategories()
    .pipe(takeUntilDestroyed(this.destroyRef))
    .subscribe({
      next: (response) => {
        this.categories.set(response.categories);
      },
      error: () => {
        this.categories.set([]);
      }
    });
}

loadMaterial(id: number): void {
```

```
  this.isLoading.set(true);
  this.errorMessage.set('');

  this.materialService
    .getMaterialById(id)
    .pipe(takeUntilDestroyed(this.destroyRef))
    .subscribe({
      next: (response) => {
        const material = response.material;

        this.material.set(material);

        this.form.patchValue({
          title: material.title,
          description: material.description || '',
          content: material.content || '',
          videoUrl: material.videoUrl || '',
          categoryId: material.categoryId ? String(material.categoryId) : '',
          status: material.status
        });

        this.isLoading.set(false);
      },
      error: (error) => {
        this.errorMessage.set(
          error.error?.message || 'Не вдалося завантажити матеріал'
        );
        this.isLoading.set(false);
      }
    });
}

onFilesSelected(event: Event): void {
  const input = event.target as HTMLInputElement;

  if (!input.files || input.files.length === 0) {
    return;
  }

  const files = Array.from(input.files);
  this.selectedFiles.set(files);
}

removeSelectedFile(index: number): void {
  const files = [...this.selectedFiles()];
  files.splice(index, 1);
  this.selectedFiles.set(files);
}

submit(): void {
  this.errorMessage.set('');
  this.successMessage.set('');

  if (this.form.invalid) {
    this.form.markAllAsTouched();
```

```
    this.errorMessage.set('Перевірте правильність заповнення форми');
    return;
  }

  const formValue = this.form.getRawValue();

  const payload = {
    title: formValue.title.trim(),
    description: formValue.description.trim(),
    content: formValue.content.trim(),
    videoUrl: formValue.videoUrl.trim(),
    categoryId: formValue.categoryId ? Number(formValue.categoryId) : null,
    status: formValue.status
  };

  this.isSaving.set(true);

  const id = this.materialId();

  const request$ = id
    ? this.materialService.updateMaterial(id, payload)
    : this.materialService.createMaterial(payload);

  request$
    .pipe(takeUntilDestroyed(this.destroyRef))
    .subscribe({
      next: (response) => {
        const savedMaterialId = response.material.id;

        if (this.selectedFiles().length > 0) {
          this.uploadSelectedFiles(savedMaterialId);
          return;
        }

        this.isSaving.set(false);
        this.successMessage.set(
          id ? 'Матеріал оновлено' : 'Матеріал створено'
        );

        this.router.navigate(['/materials', savedMaterialId]);
      },
      error: (error) => {
        this.errorMessage.set(
          error.error?.message || 'Не вдалося зберегти матеріал'
        );
        this.isSaving.set(false);
      }
    });
}

uploadSelectedFiles(materialId: number): void {
  this.materialService
    .uploadFiles(materialId, this.selectedFiles())
    .pipe(takeUntilDestroyed(this.destroyRef))
    .subscribe({
```

```
      next: () => {
        this.selectedFiles.set([]);
        this.isSaving.set(false);
        this.successMessage.set('Матеріал і файли збережено');
        this.router.navigate(['/materials', materialId]);
      },
      error: (error) => {
        this.errorMessage.set(
          error.error?.message ||
            'Матеріал збережено, але файли не вдалося завантажити'
        );
        this.isSaving.set(false);
      }
    });
}

deleteExistingFile(file: MediaFile): void {
  const confirmed = window.confirm(
    `Видалити файл "${file.originalName}"?`
  );

  if (!confirmed) {
    return;
  }

  this.errorMessage.set('');
  this.successMessage.set('');

  this.materialService
    .deleteFile(file.id)
    .pipe(takeUntilDestroyed(this.destroyRef))
    .subscribe({
      next: () => {
        this.successMessage.set('Файл видалено');

        const id = this.materialId();

        if (id) {
          this.loadMaterial(id);
        }
      },
      error: (error) => {
        this.errorMessage.set(
          error.error?.message || 'Не вдалося видалити файл'
        );
      }
    });
}

getFileUrl(file: MediaFile): string {
  return `${environment.serverUrl}${file.filePath}`;
}

isImage(file: MediaFile): boolean {
  return file.mimeType.startsWith('image/');
```

```
  }

  formatFileSize(size: number): string {
    if (size < 1024) {
      return `${size} B`;
    }

    if (size < 1024 * 1024) {
      return `${(size / 1024).toFixed(1)} KB`;
    }

    return `${(size / 1024 / 1024).toFixed(1)} MB`;
  }

  getTitleError(): string {
    const control = this.form.controls.title;

    if (control.hasError('required')) {
      return 'Назва матеріалу є обов’язковою';
    }

    if (control.hasError('minlength')) {
      return 'Назва має містити щонайменше 3 символи';
    }

    return '';
  }
}
```

ДОДАТОК Б

Angular MaterialService

```
import { Injectable } from '@angular/core';
import {
  HttpClient,
  HttpParams
} from '@angular/common/http';
import { Observable } from 'rxjs';

import { environment } from '../../../environments/environment';

import {
  MaterialCreateRequest,
  MaterialResponse,
  MaterialsResponse,
  MaterialStatus,
  MaterialUpdateRequest,
  PublishMaterialResponse
} from '../models/material.model';

export interface MaterialListParams {
  search?: string;
  status?: MaterialStatus;
  categoryId?: number;
  authorId?: number;
  page?: number;
  limit?: number;
}

@Injectable({
  providedIn: 'root'
})
export class MaterialService {
  private readonly apiUrl = `${environment.apiUrl}/materials`;
  private readonly filesApiUrl = `${environment.apiUrl}/files`;

  constructor(private readonly http: HttpClient) {}

  getMaterials(params?: MaterialListParams): Observable<MaterialsResponse> {
    let httpParams = new HttpParams();

    if (params?.search) {
      httpParams = httpParams.set('search', params.search);
    }

    if (params?.status) {
      httpParams = httpParams.set('status', params.status);
    }
```

```
  if (params?.categoryId) {
    httpParams = httpParams.set('categoryId', params.categoryId);
  }

  if (params?.authorId) {
    httpParams = httpParams.set('authorId', params.authorId);
  }

  if (params?.page) {
    httpParams = httpParams.set('page', params.page);
  }

  if (params?.limit) {
    httpParams = httpParams.set('limit', params.limit);
  }

  return this.http.get<MaterialsResponse>(this.apiUrl, {
    params: httpParams
  });
}

getMyMaterials(): Observable<MaterialsResponse> {
  return this.http.get<MaterialsResponse>(`${this.apiUrl}/my`);
}

getMaterialById(id: number): Observable<MaterialResponse> {
  return this.http.get<MaterialResponse>(`${this.apiUrl}/${id}`);
}

createMaterial(data: MaterialCreateRequest): Observable<MaterialResponse> {
  return this.http.post<MaterialResponse>(this.apiUrl, data);
}

updateMaterial(
  id: number,
  data: MaterialUpdateRequest
): Observable<MaterialResponse> {
  return this.http.put<MaterialResponse>(`${this.apiUrl}/${id}`, data);
}

deleteMaterial(id: number): Observable<{ message: string }> {
  return this.http.delete<{ message: string }>(`${this.apiUrl}/${id}`);
}

publishMaterial(id: number): Observable<PublishMaterialResponse> {
  return this.http.patch<PublishMaterialResponse>(
    `${this.apiUrl}/${id}/publish`,
    {}
  );
}

unpublishMaterial(id: number): Observable<MaterialResponse> {
  return this.http.patch<MaterialResponse>(
    `${this.apiUrl}/${id}/unpublish`,
    {}
```

```
    );
  }

  uploadFiles(
  materialId: number,
  files: File[]
): Observable<{ message: string; files: unknown[] }> {
  const formData = new FormData();

  for (const file of files) {
    formData.append('files', file);
  }

  return this.http.post<{ message: string; files: unknown[] }>(
    `${this.apiUrl}/${materialId}/files`,
    formData
  );
}

  deleteFile(fileId: number): Observable<{ message: string }> {
    return this.http.delete<{ message: string }>(
      `${this.filesApiUrl}/${fileId}`
    );
  }
}
```

ДОДАТОК В

Express material.controller.js

```
import { Op } from 'sequelize';

import {
  Material,
  Category,
  User,
  MediaFile
} from '../models/index.js';

import { generateUniqueSlug } from '../utils/slug.helper.js';

const canManageMaterial = (user, material) => {
  return user.role === 'admin' || Number(material.authorId) === Number(user.id);
};

const listInclude = [
  {
    model: User,
    as: 'author',
    attributes: ['id', 'name', 'email', 'role']
  },
  {
    model: Category,
    as: 'category',
    attributes: ['id', 'name', 'description']
  }
];

const detailInclude = [
  {
    model: User,
    as: 'author',
    attributes: ['id', 'name', 'email', 'role']
  },
  {
    model: Category,
    as: 'category',
    attributes: ['id', 'name', 'description']
  },
  {
    model: MediaFile,
    as: 'files'
  }
];

export const getAllMaterials = async (req, res) => {
  try {
```

```
const {
  search,
  status,
  categoryId,
  authorId,
  page = 1,
  limit = 10
} = req.query;

const where = {};

if (req.user.role !== 'admin') {
  where.authorId = req.user.id;
} else if (authorId) {
  where.authorId = authorId;
}

if (status) {
  where.status = status;
}

if (categoryId) {
  where.categoryId = categoryId;
}

if (search) {
  where[Op.or] = [
    {
      title: {
        [Op.like]: `%${search}%`
      }
    },
    {
      description: {
        [Op.like]: `%${search}%`
      }
    }
  ];
}

const pageNumber = Number(page);
const limitNumber = Number(limit);
const offset = (pageNumber - 1) * limitNumber;

const { rows, count } = await Material.findAndCountAll({
  where,
  include: listInclude,
  order: [['createdAt', 'DESC']],
  limit: limitNumber,
  offset
});

return res.json({
  materials: rows,
  pagination: {
```

```
        total: count,
        page: pageNumber,
        limit: limitNumber,
        totalPages: Math.ceil(count / limitNumber)
      }
    });
  } catch (error) {
    return res.status(500).json({
      message: 'Failed to get materials',
      error: error.message
    });
  }
};

export const getMyMaterials = async (req, res) => {
  try {
    const materials = await Material.findAll({
      where: {
        authorId: req.user.id
      },
      include: listInclude,
      order: [['createdAt', 'DESC']]
    });

    return res.json({
      materials
    });
  } catch (error) {
    return res.status(500).json({
      message: 'Failed to get user materials',
      error: error.message
    });
  }
};

export const getMaterialById = async (req, res) => {
  try {
    const { id } = req.params;

    const material = await Material.findByPk(id, {
      include: detailInclude
    });

    if (!material) {
      return res.status(404).json({
        message: 'Material not found'
      });
    }

    if (!canManageMaterial(req.user, material)) {
      return res.status(403).json({
        message: 'Access denied'
      });
    }
```

```
    return res.json({
      material
    });
  } catch (error) {
    return res.status(500).json({
      message: 'Failed to get material',
      error: error.message
    });
  }
};

export const createMaterial = async (req, res) => {
  try {
    const {
      title,
      description,
      content,
      videoUrl,
      categoryId,
      status = 'draft'
    } = req.body;

    if (!title || title.trim() === '') {
      return res.status(400).json({
        message: 'Material title is required'
      });
    }

    if (status && !['draft', 'published'].includes(status)) {
      return res.status(400).json({
        message: 'Invalid material status'
      });
    }

    if (categoryId) {
      const category = await Category.findByPk(categoryId);

      if (!category) {
        return res.status(404).json({
          message: 'Category not found'
        });
      }
    }

    const slug = await generateUniqueSlug(title);

    const material = await Material.create({
      title,
      description,
      content,
      videoUrl,
      categoryId: categoryId || null,
      status,
      slug,
      authorId: req.user.id
```

```
    });

    const createdMaterial = await Material.findByPk(material.id, {
      include: detailInclude
    });

    return res.status(201).json({
      message: 'Material created successfully',
      material: createdMaterial
    });
  } catch (error) {
    return res.status(500).json({
      message: 'Failed to create material',
      error: error.message
    });
  }
};

export const updateMaterial = async (req, res) => {
  try {
    const { id } = req.params;

    const {
      title,
      description,
      content,
      videoUrl,
      categoryId,
      status
    } = req.body;

    const material = await Material.findByPk(id);

    if (!material) {
      return res.status(404).json({
        message: 'Material not found'
      });
    }

    if (!canManageMaterial(req.user, material)) {
      return res.status(403).json({
        message: 'Access denied'
      });
    }

    if (title !== undefined && title.trim() === '') {
      return res.status(400).json({
        message: 'Material title cannot be empty'
      });
    }

    if (status && !['draft', 'published'].includes(status)) {
      return res.status(400).json({
        message: 'Invalid material status'
      });
```

```
    }

    if (categoryId) {
      const category = await Category.findByPk(categoryId);

      if (!category) {
        return res.status(404).json({
          message: 'Category not found'
        });
      }
    }

    let newSlug = material.slug;

    if (title && title !== material.title) {
      newSlug = await generateUniqueSlug(title, material.id);
    }

    await material.update({
      title: title ?? material.title,
      description: description ?? material.description,
      content: content ?? material.content,
      videoUrl: videoUrl ?? material.videoUrl,
      categoryId: categoryId === undefined ? material.categoryId : categoryId,
      status: status ?? material.status,
      slug: newSlug
    });

    const updatedMaterial = await Material.findByPk(material.id, {
      include: detailInclude
    });

    return res.json({
      message: 'Material updated successfully',
      material: updatedMaterial
    });
  } catch (error) {
    return res.status(500).json({
      message: 'Failed to update material',
      error: error.message
    });
  }
};

export const deleteMaterial = async (req, res) => {
  try {
    const { id } = req.params;

    const material = await Material.findByPk(id);

    if (!material) {
      return res.status(404).json({
        message: 'Material not found'
      });
    }
```

```
    if (!canManageMaterial(req.user, material)) {
      return res.status(403).json({
        message: 'Access denied'
      });
    }

    await material.destroy();

    return res.json({
      message: 'Material deleted successfully'
    });
  } catch (error) {
    return res.status(500).json({
      message: 'Failed to delete material',
      error: error.message
    });
  }
};

export const publishMaterial = async (req, res) => {
  try {
    const { id } = req.params;

    const material = await Material.findByPk(id);

    if (!material) {
      return res.status(404).json({
        message: 'Material not found'
      });
    }

    if (!canManageMaterial(req.user, material)) {
      return res.status(403).json({
        message: 'Access denied'
      });
    }

    if (!material.slug) {
      const slug = await generateUniqueSlug(material.title, material.id);
      material.slug = slug;
    }

    material.status = 'published';

    await material.save();

    const updatedMaterial = await Material.findByPk(material.id, {
      include: detailInclude
    });

    return res.json({
      message: 'Material published successfully',
      publicUrl: `/api/public/materials/${material.slug}`,
      material: updatedMaterial
    });
```

```
  } catch (error) {
    return res.status(500).json({
      message: 'Failed to publish material',
      error: error.message
    });
  }
};

export const unpublishMaterial = async (req, res) => {
  try {
    const { id } = req.params;

    const material = await Material.findByPk(id);

    if (!material) {
      return res.status(404).json({
        message: 'Material not found'
      });
    }

    if (!canManageMaterial(req.user, material)) {
      return res.status(403).json({
        message: 'Access denied'
      });
    }

    material.status = 'draft';

    await material.save();

    const updatedMaterial = await Material.findByPk(material.id, {
      include: detailInclude
    });

    return res.json({
      message: 'Material unpublished successfully',
      material: updatedMaterial
    });
  } catch (error) {
    return res.status(500).json({
      message: 'Failed to unpublish material',
      error: error.message
    });
  }
};
export const getPublicMaterialBySlug = async (req, res) => {
  try {
    const { slug } = req.params;

    const material = await Material.findOne({
      where: {
        slug,
        status: 'published'
      },
      include: detailInclude
```

```
    });
    if (!material) {
      return res.status(404).json({
        message: 'Published material not found'
      });
    }

    await material.increment('views');

    await material.reload({
      include: detailInclude
    });

    return res.json({
      material
    });
  } catch (error) {
    return res.status(500).json({
      message: 'Failed to get public material',
      error: error.message
    });
  }
};
```

ДОДАТОК Г

Express Material.js

```
import { DataTypes } from 'sequelize';
import { sequelize } from '../config/database.js';

export const Material = sequelize.define(
  'Material',
  {
    id: {
      type: DataTypes.INTEGER.UNSIGNED,
      autoIncrement: true,
      primaryKey: true
    },

    title: {
      type: DataTypes.STRING(200),
      allowNull: false,
      validate: {
        notEmpty: {
          msg: 'Material title is required'
        },
        len: {
          args: [3, 200],
          msg: 'Title must be between 3 and 200 characters'
        }
      }
    },

    description: {
      type: DataTypes.TEXT,
      allowNull: true
    },

    content: {
      type: DataTypes.TEXT('long'),
      allowNull: true
    },

    videoUrl: {
      type: DataTypes.STRING,
      allowNull: true,
      validate: {
        isUrlOrEmpty(value) {
          if (value && value.trim() !== '') {
            const urlPattern = /^https?:\/\/.+/;

            if (!urlPattern.test(value)) {
              throw new Error('Video URL must be a valid URL');
            }
```

```
        }
       }
      }
    },

    status: {
      type: DataTypes.ENUM('draft', 'published'),
      allowNull: false,
      defaultValue: 'draft'
    },

    slug: {
      type: DataTypes.STRING,
      allowNull: true,
      unique: true
    },

    views: {
      type: DataTypes.INTEGER.UNSIGNED,
      allowNull: false,
      defaultValue: 0
    },

    authorId: {
      type: DataTypes.INTEGER.UNSIGNED,
      allowNull: false
    },

    categoryId: {
      type: DataTypes.INTEGER.UNSIGNED,
      allowNull: true
    }
  },
  {
    tableName: 'materials',
    timestamps: true
  }
);
```

ДОДАТОК Д

Unit-тест MaterialFormComponent

```
import { ComponentFixture, TestBed } from '@angular/core/testing';
import {
  ActivatedRoute,
  Router,
  convertToParamMap,
  provideRouter
} from '@angular/router';
import { of, throwError } from 'rxjs';

import { MaterialFormComponent } from './material-form.component';

import { MaterialService } from '../../../core/services/material.service';
import { CategoryService } from '../../../core/services/category.service';

import { Category } from '../../../core/models/category.model';
import { Material } from '../../../core/models/material.model';

describe('MaterialFormComponent', () => {
  let fixture: ComponentFixture<MaterialFormComponent>;
  let component: MaterialFormComponent;

  let materialService: jasmine.SpyObj<MaterialService>;
  let categoryService: jasmine.SpyObj<CategoryService>;
  let router: Router;

  let activatedRouteMock: {
    snapshot: {
      paramMap: ReturnType<typeof convertToParamMap>;
    };
  };

  const mockCategories: Category[] = [
    {
      id: 1,
      name: 'Інформатика',
      description: 'Матеріали з інформатики'
    },
    {
      id: 2,
      name: 'STEM',
      description: 'STEM-матеріали'
    }
  ];

  const mockMaterial: Material = {
    id: 10,
    title: 'Основи роботи з мультимедіа',
```

```
  description: 'Короткий опис навчального матеріалу',
  content: 'Основний текст навчального матеріалу',
  videoUrl: 'https://www.youtube.com/watch?v=test',
  status: 'draft',
  slug: 'osnovi-roboti-z-multimedia',
  views: 0,
  authorId: 1,
  categoryId: 1,
  category: mockCategories[0],
  files: []
};

beforeEach(async () => {
  materialService = jasmine.createSpyObj<MaterialService>('MaterialService', [
    'getMaterialById',
    'createMaterial',
    'updateMaterial',
    'uploadFiles',
    'deleteFile'
  ]);

  categoryService = jasmine.createSpyObj<CategoryService>('CategoryService', [
    'getCategories'
  ]);

  activatedRouteMock = {
    snapshot: {
      paramMap: convertToParamMap({})
    }
  };

  categoryService.getCategories.and.returnValue(
    of({
      categories: mockCategories
    })
  );

  materialService.getMaterialById.and.returnValue(
    of({
      material: mockMaterial
    })
  );

  materialService.createMaterial.and.returnValue(
    of({
      material: mockMaterial
    })
  );

  materialService.updateMaterial.and.returnValue(
    of({
      material: mockMaterial
    })
  );
```

```
  materialService.uploadFiles.and.returnValue(
    of({
      message: 'Files uploaded successfully',
      files: []
    })
  );

  materialService.deleteFile.and.returnValue(
    of({
      message: 'File deleted successfully'
    })
  );

  await TestBed.configureTestingModule({
    imports: [MaterialFormComponent],
    providers: [
      provideRouter([]),
      {
        provide: ActivatedRoute,
        useValue: activatedRouteMock
      },
      {
        provide: MaterialService,
        useValue: materialService
      },
      {
        provide: CategoryService,
        useValue: categoryService
      }
    ]
  }).compileComponents();

  router = TestBed.inject(Router);
});

function createComponent(materialId?: string): void {
  activatedRouteMock.snapshot.paramMap = convertToParamMap(
    materialId ? { id: materialId } : {}
  );

  fixture = TestBed.createComponent(MaterialFormComponent);
  component = fixture.componentInstance;
  fixture.detectChanges();
}

it('should create component and load categories', () => {
  createComponent();

  expect(component).toBeTruthy();
  expect(categoryService.getCategories).toHaveBeenCalled();
  expect(component.categories()).toEqual(mockCategories);
});

it('should not submit form when title is empty', () => {
  createComponent();
```

```
  component.form.patchValue({
    title: '',
    description: 'Опис матеріалу',
    content: 'Текст матеріалу',
    videoUrl: '',
    categoryId: '1',
    status: 'draft'
  });

  component.submit();

  expect(component.form.invalid).toBeTrue();
  expect(component.errorMessage()).toBe(
    'Перевірте правильність заповнення форми'
  );
  expect(materialService.createMaterial).not.toHaveBeenCalled();
});

it('should not submit form when title is shorter than 3 characters', () => {
  createComponent();

  component.form.patchValue({
    title: 'AB',
    description: 'Опис матеріалу',
    content: 'Текст матеріалу',
    videoUrl: '',
    categoryId: '1',
    status: 'draft'
  });

  component.submit();

  expect(component.form.invalid).toBeTrue();
  expect(component.getTitleError()).toBe(
    'Назва має містити щонайменше 3 символи'
  );
  expect(materialService.createMaterial).not.toHaveBeenCalled();
});

it('should create material with valid form data', () => {
  createComponent();

  const navigateSpy = spyOn(router, 'navigate').and.returnValue(
    Promise.resolve(true)
  );

  component.form.setValue({
    title: 'Новий навчальний матеріал',
    description: 'Опис нового матеріалу',
    content: 'Основний текст нового матеріалу',
    videoUrl: 'https://www.youtube.com/watch?v=test',
    categoryId: '1',
    status: 'published'
  });
```

```
  component.submit();

  expect(materialService.createMaterial).toHaveBeenCalledWith({
    title: 'Новий навчальний матеріал',
    description: 'Опис нового матеріалу',
    content: 'Основний текст нового матеріалу',
    videoUrl: 'https://www.youtube.com/watch?v=test',
    categoryId: 1,
    status: 'published'
  });

  expect(materialService.uploadFiles).not.toHaveBeenCalled();
  expect(navigateSpy).toHaveBeenCalledWith(['/materials', mockMaterial.id]);
});

it('should create material with null category when category is not selected', () => {
  createComponent();

  spyOn(router, 'navigate').and.returnValue(Promise.resolve(true));

  component.form.setValue({
    title: 'Матеріал без категорії',
    description: '',
    content: 'Текст матеріалу',
    videoUrl: '',
    categoryId: '',
    status: 'draft'
  });

  component.submit();

  expect(materialService.createMaterial).toHaveBeenCalledWith({
    title: 'Матеріал без категорії',
    description: '',
    content: 'Текст матеріалу',
    videoUrl: '',
    categoryId: null,
    status: 'draft'
  });
});

it('should upload selected files after material creation', () => {
  createComponent();

  const navigateSpy = spyOn(router, 'navigate').and.returnValue(
    Promise.resolve(true)
  );

  const file = new File(['image-content'], 'lesson-image.png', {
    type: 'image/png'
  });

  component.selectedFiles.set([file]);

  component.form.setValue({
```

```
    title: 'Матеріал із файлом',
    description: 'Опис матеріалу',
    content: 'Текст матеріалу',
    videoUrl: '',
    categoryId: '1',
    status: 'draft'
  });

  component.submit();

  expect(materialService.createMaterial).toHaveBeenCalled();
  expect(materialService.uploadFiles).toHaveBeenCalledWith(mockMaterial.id, [
    file
  ]);
  expect(component.selectedFiles()).toEqual([]);
  expect(navigateSpy).toHaveBeenCalledWith(['/materials', mockMaterial.id]);
});

it('should load material data in edit mode', () => {
  createComponent('10');

  expect(component.isEditMode()).toBeTrue();
  expect(materialService.getMaterialById).toHaveBeenCalledWith(10);

  expect(component.form.controls.title.value).toBe(mockMaterial.title);
  expect(component.form.controls.description.value).toBe(
    mockMaterial.description || ''
  );
  expect(component.form.controls.content.value).toBe(
    mockMaterial.content || ''
  );
  expect(component.form.controls.videoUrl.value).toBe(
    mockMaterial.videoUrl || ''
  );
  expect(component.form.controls.categoryId.value).toBe('1');
  expect(component.form.controls.status.value).toBe(mockMaterial.status);
});

it('should update existing material in edit mode', () => {
  createComponent('10');

  const navigateSpy = spyOn(router, 'navigate').and.returnValue(
    Promise.resolve(true)
  );

  component.form.patchValue({
    title: 'Оновлений навчальний матеріал',
    description: 'Оновлений опис',
    content: 'Оновлений текст',
    videoUrl: '',
    categoryId: '',
    status: 'published'
  });

  component.submit();
```

```
  expect(materialService.updateMaterial).toHaveBeenCalledWith(10, {
    title: 'Оновлений навчальний матеріал',
    description: 'Оновлений опис',
    content: 'Оновлений текст',
    videoUrl: '',
    categoryId: null,
    status: 'published'
  });

  expect(materialService.createMaterial).not.toHaveBeenCalled();
  expect(navigateSpy).toHaveBeenCalledWith(['/materials', mockMaterial.id]);
});

it('should show error message when create material request fails', () => {
  createComponent();

  materialService.createMaterial.and.returnValue(
    throwError(() => ({
      error: {
        message: 'Failed to create material'
      }
    }))
  );

  component.form.setValue({
    title: 'Новий матеріал',
    description: 'Опис',
    content: 'Текст',
    videoUrl: '',
    categoryId: '1',
    status: 'draft'
  });

  component.submit();

  expect(component.errorMessage()).toBe('Failed to create material');
  expect(component.isSaving()).toBeFalse();
});

it('should remove selected file from list', () => {
  createComponent();

  const firstFile = new File(['first'], 'first.png', {
    type: 'image/png'
  });

  const secondFile = new File(['second'], 'second.pdf', {
    type: 'application/pdf'
  });

  component.selectedFiles.set([firstFile, secondFile]);

  component.removeSelectedFile(0);

  expect(component.selectedFiles()).toEqual([secondFile]);
```

```
  });

  it('should format file size correctly', () => {
    createComponent();

    expect(component.formatFileSize(500)).toBe('500 B');
    expect(component.formatFileSize(2048)).toBe('2.0 KB');
    expect(component.formatFileSize(2 * 1024 * 1024)).toBe('2.0 MB');
  });
});
```