ДЕРЖАВНИЙ УНІВЕРСИТЕТ
ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ
НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ
ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ
КАФЕДРА ТЕХНОЛОГІЙ ЦИФРОВОГО РОЗВИТКУ

КВАЛІФІКАЦІЙНА РОБОТА

на тему: «Цифрова web-платформа гравців Albion Online з функціоналом публікації новин та гайдів на основі React.js »

на здобуття освітнього ступеня бакалавра
зі спеціальності 121 Інженерія програмного забезпечення
освітньо-професійної програми «Технології цифрового розвитку»

Кваліфікаційна робота містить результати власних досліджень. Використання ідей, результатів і текстів інших авторів мають посилання на відповідне джерело

_____ Артем ТЕКЛЮК
(підпис)

Виконав: здобувач вищої освіти групи ТЦР-41

Артем ТЕКЛЮК

Керівник Ірина ГНІДЕНКО
старший викладач кафедри .

Рецензент _____

Київ 2026

ДЕРЖАВНИЙ УНІВЕРСИТЕТ
ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ

Навчально-науковий інститут інформаційних технологій

Кафедра Технологій цифрового розвитку

Ступінь вищої освіти Бакалавр

Спеціальність 121 Інженерія програмного забезпечення

Освітньо-професійна програма «Технології цифрового розвитку»

ЗАТВЕРДЖУЮ
Завідувач кафедри
Технологій цифрового розвитку
_____ Вікторія ЖЕБКА
« ____ » _____ 2026 р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ

Теклюк Артем Олексійович

1. Тема кваліфікаційної роботи: «Цифрова web-платформа гравців Albion Online з функціоналом публікації новин та гайдів на основі React.js», керівник кваліфікаційної роботи старший викладач кафедри Ірина ГНІДЕНКО, затверджені наказом Державного університету інформаційно-комунікаційних технологій від «20» лютого 2026 року №51.

2. Строк подання кваліфікаційної роботи «18» травня 2026 року.

3. Вихідні дані до кваліфікаційної роботи: інформаційні ресурси та програмні компоненти для розробки модулі обробки, зберігання та відображення даних у середовищі програмування JavaScript (Node.js, React), механізми взаємодії з API гри Albion Online для отримання інформації про персонажів

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити)

1.Аналіз предметної області та постановка задачі створення веб-платформи для гравців Albion Online з функціоналом публікації гайдів, новин та ігрового контенту.

2.Огляд існуючих веб-платформ для ігрових спільнот, технологій збору та обробки даних, аналіз можливостей сучасних веб-фреймворків і бібліотек для реалізації інтерактивних веб-застосунків.
3.Розробка, програмна реалізація, тестування та оцінювання ефективності роботи веб-платформи для публікації білдів, новин і гайдів для гравців Albion Online.

5. Перелік графічного матеріалу: *презентація*
1. Презентація PowerPoint.

6. Дата видачі завдання «20» лютого 2026 року.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1	Аналіз предметної області та огляд сучасних веб-платформ і технологій для створення платформ ігрових спільнот та публікації контенту в Albion Online	20.02.2026 р.-10.04.2026	Виконано
2	Вивчення технічної документації API Albion Online, веб-фреймворків React, Node.js та технологій JavaScript	10.04.2026 р. - 12.04.2026 р	Виконано
3	Розробка архітектури програмної системи: модулі взаємодії з API, обробки та зберігання даних у PostgreSQL, а також побудова користувацького інтерфейсу	12.04.2026 р. - 16.04.2026 р	Виконано
4	Програмна реалізація веб-платформи з використанням React, TailwindCSS та Node.js для публікації гайдів, новин і білдів	16.04.2026 р. - 20.04.2026 р.	Виконано
5	Реалізація та налаштування механізмів обробки даних, системи авторизації (JWT), тестування веб-платформи в різних сценаріях використання	20.04.2026 р. - 22.04.2026 р.	Виконано
6	Завершальне тестування веб-платформи, перевірка функціональності, оптимізація та підготовка демонстраційних матеріалів	22.04.2026 р. - 26.04.2026 р.	Виконано

Здобувач вищої освіти _____ Артем Теклюк
(підпис)

Керівник
кваліфікаційної роботи _____ Ірина ГНІДЕНКО
(підпис)

РЕФЕРАТ

Текстова частина бакалаврської роботи: 57 сторінок, 28 рисунків, 6 таблиць, 32 джерела.

Мета роботи – розробка веб-платформи для гравців Albion Online з функціоналом публікації білдів, новин та гайдів, що забезпечує зручний інтерфейс користувача, систему авторизації, управління контентом та ефективну взаємодію клієнтської та серверної частини.

Об'єкт дослідження – процеси створення веб-платформ для спільнот гравців онлайн-ігор та управління контентом.

Предмет дослідження – методи та засоби розробки веб-застосунків із використанням сучасних технологій frontend і backend для забезпечення публікації контенту, взаємодії користувачів та управління даними.

Короткий зміст роботи: У результаті виконання кваліфікаційної роботи було розроблено веб-платформу для спільноти гравців Albion Online, яка забезпечує публікацію новин, гайдів та профілів користувачів, а також підтримує ефективну взаємодію між користувачами системи. У межах роботи спроєктовано клієнт-серверну архітектуру застосунку, реалізовано frontend-частину з використанням React і TailwindCSS та backend-логіку на основі Node.js і Express.

Також було розроблено структуру бази даних PostgreSQL, реалізовано механізми авторизації та управління контентом, а також API для обміну даними між клієнтською та серверною частинами. Проведене тестування підтвердило стабільність роботи системи, коректність обробки даних та зручність використання веб-платформи.

КЛЮЧОВІ СЛОВА: ВЕБ-ПЛАТФОРМА, REACT, NODE.JS, POSTGRESQL, TAILWINDCSS, JWT, CRUD, ВЕБ-РОЗРОБКА, КЛІЄНТ-СЕРВЕРНА АРХІТЕКТУРА, ALBION ONLINE

ЗМІСТ

ВСТУП

Актуальність дослідження. У сучасних умовах стрімкого розвитку індустрії онлайн-ігор особливого значення набувають інформаційні платформи, що забезпечують ефективну взаємодію між гравцями, обмін досвідом та доступ до актуального контенту. Однією з популярних багатокористувацьких ігор є Albion Online, яка характеризується складною ігровою економікою, розвиненою системою взаємодії між гравцями та високим рівнем конкуренції. У зв'язку з цим виникає потреба у створенні спеціалізованих веб-платформ, які дозволяють централізовано публікувати гайди, білді, новини та іншу корисну інформацію для ігрової спільноти.

Наявні рішення часто мають обмежений функціонал, незручний інтерфейс або не забезпечують належного рівня інтерактивності та персоналізації контенту. Крім того, значна частина інформації розпорошена між різними ресурсами, що ускладнює її пошук та використання. Це зумовлює необхідність розробки сучасної веб-платформи, яка б поєднувала зручний користувацький інтерфейс, ефективні механізми обробки та зберігання даних, а також підтримувала взаємодію з API гри для отримання актуальної інформації.

Актуальність роботи також визначається використанням сучасних технологій веб-розробки, таких як React.js, TailwindCSS, Node.js та PostgreSQL, які дозволяють створювати масштабовані, продуктивні та зручні у використанні інформаційні системи. Розробка такої платформи сприятиме підвищенню якості інформаційного забезпечення гравців, покращенню комунікації в ігровій спільноті та оптимізації процесів публікації та споживання контенту.

Об'єкт дослідження – процеси створення веб-платформ для спільнот гравців онлайн-ігор та управління контентом.

Предмет дослідження – методи та засоби розробки веб-застосунків із використанням сучасних технологій frontend і backend для забезпечення публікації контенту, взаємодії користувачів та управління даними.

Мета роботи – розробка веб-платформи для гравців Albion Online з функціоналом публікації білдів, новин та гайдів, що забезпечує зручний інтерфейс користувача, систему авторизації, управління контентом та ефективну взаємодію клієнтської та серверної частини.

Наукові завдання:

1. Проаналізувати предметну область веб-платформ для ігрових спільнот та визначити особливості організації контенту для гравців Albion Online.
2. Дослідити сучасні підходи до розробки веб-застосунків, зокрема клієнт-серверну архітектуру та використання frontend і backend технологій.
3. Провести аналіз існуючих платформ і сервісів для публікації ігрового контенту, визначити їх переваги та недоліки.
4. Обґрунтувати вибір технологічного стеку (React.js, TailwindCSS, Node.js, PostgreSQL) для реалізації веб-платформи.
5. Розробити архітектуру веб-платформи з урахуванням модульності, масштабованості та зручності взаємодії користувачів.
6. Спроєктувати структуру бази даних для зберігання користувачів, новин, гайдів та білдів.
7. Реалізувати механізми взаємодії з API гри Albion Online для отримання актуальних даних.
8. Розробити клієнтську частину веб-платформи з використанням сучасних підходів до UI/UX дизайну.
9. Реалізувати серверну частину застосунку з підтримкою REST API та обробки запитів.
10. Впровадити систему автентифікації та авторизації користувачів.
11. Провести тестування функціональності веб-платформи та оцінити її ефективність.

Методи дослідження:

1. Аналіз літературних та інформаційних джерел – для дослідження сучасних підходів до розробки веб-платформ, систем управління контентом та онлайн-сервісів для ігрових спільнот.

2. Методи системного аналізу – для визначення структури веб-платформи, її основних компонентів, функціональних можливостей та взаємозв’язків між клієнтською і серверною частинами.

3. Методи проєктування програмного забезпечення – для розробки архітектури клієнт-серверного застосунку, структури бази даних PostgreSQL та логіки взаємодії між модулями системи.

4. Методи програмної реалізації – для створення frontend-частини з використанням React і TailwindCSS, backend-частини на Node.js та Express, а також реалізації API для обміну даними.

5. Методи тестування програмного забезпечення – для перевірки працездатності системи, коректності авторизації, публікації контенту, обробки запитів та взаємодії з базою даних.

6. Методи оцінювання ефективності – для визначення зручності використання веб-платформи, стабільності її роботи та відповідності реалізованого функціоналу поставленим вимогам.

Аналіз сучасних підходів до розробки веб-платформ та систем управління контентом.

1. Проєктування архітектури клієнт-серверного застосунку.
2. Розробка frontend-частини з використанням React та TailwindCSS.
3. Реалізація backend-логіки на Node.js та Express.
4. Проєктування та використання бази даних PostgreSQL.
5. Тестування функціональності системи та оцінка її ефективності.

Використання сучасних мов програмування JavaScript та відповідних веб-технологій дозволяє створювати ефективні рішення для розробки інтерактивних веб-платформ. Застосування таких інструментів, як React.js та Node.js, забезпечує реалізацію механізмів обробки, зберігання та відображення даних, пов’язаних із публікацією гайдів, новин та білдів для гравців Albion Online. Для підвищення зручності взаємодії користувача з системою реалізовано веб-інтерфейс із використанням сучасних бібліотек та підходів до UI/UX-дизайну, що забезпечує наочне представлення інформації та інтуїтивне керування контентом.

Розроблена веб-платформа дозволяє користувачам створювати, редагувати та переглядати ігровий контент, а також отримувати актуальну інформацію у зручному форматі. Використання бази даних PostgreSQL забезпечує надійне зберігання даних та швидкий доступ до них, а інтеграція з API гри дозволяє отримувати актуальні відомості про ігрові елементи. Система підтримує роботу в режимі реального часу, що сприяє оперативному оновленню контенту та підвищує її практичну цінність для користувачів.

Результати виконаної роботи можуть бути використані у навчальному процесі для демонстрації сучасних підходів до розробки веб-застосунків, а також у практичній діяльності для створення подібних інформаційних систем. Розроблена веб-платформа демонструє ефективність використання сучасного стеку технологій для реалізації масштабованих і функціональних веб-сервісів, що відповідають потребам користувачів у сфері онлайн-ігор та цифрових спільнот.

Апробацію результатів кваліфікаційної роботи здійснено шляхом публікації тез доповідей у матеріалах Всеукраїнської науково-технічної конференції «Технології цифрового розвитку: програмні системи та децентралізація» (Збірник тез. – К.: ДУІКТ, 2026):

1. Теклюк А. О. РОЗРОБКА ТА ОПТИМІЗАЦІЯ КЛІЄНТСЬКОГО ІНТЕРФЕЙСУ ВЕБ-ПЛАТФОРМИ ДЛЯ ІГРОВОЇ СПІЛЬНОТИ З ВИКОРИСТАННЯМ REACT.JS ТА TAILWINDCSS «Теорія модернізації в контексті сучасної світової науки». Збірник тез. – К.:МЦНД, 2026.
2. Теклюк А. О. Розробка та оптимізація клієнтського інтерфейсувеб-платформи для ігрової спільноти звикористанням React.js та TailwindCSS «Синергія інновацій: полытика, економіка та менеджмент в цифоровому світі ». Збірник тез. – К.: ДУІКТ, 2026.

1 ТЕОРЕТИЧНІ ОСНОВИ РОЗРОБКИ ВЕБ-ПЛАТФОРМ ДЛЯ ІГРОВИХ СПІЛЬНОТ

1.1 Особливості веб-платформ для онлайн-ігор та їх функціональні можливості

Веб-платформа для геймерів являє собою спеціалізовану інформаційну систему, яка функціонує у веб-середовищі та призначена для забезпечення взаємодії користувачів, обміну ігровим досвідом, а також публікації та споживання контенту, пов’язаного з онлайн-іграми. Такі платформи поєднують у собі елементи соціальних мереж, інформаційних порталів та систем управління контентом, що дозволяє створювати єдине середовище для гравців [1].

Основною метою веб-платформ для геймерів є централізація інформації, яка може бути розпорошена між різними джерелами, та забезпечення зручного доступу до неї. Користувачі отримують можливість не лише переглядати новини та аналітичні матеріали, але й самостійно створювати контент, зокрема публікувати гайди, ділитися власними стратегіями гри, обговорювати нововведення та взаємодіяти з іншими учасниками спільноти.

Веб-платформи для онлайн-ігор характеризуються інтерактивністю, динамічністю та орієнтацією на користувача. Вони забезпечують персоналізацію контенту, швидкий доступ до актуальної інформації та підтримку комунікації між гравцями. Особливої актуальності такі платформи набувають у контексті багатокористувацьких онлайн-ігор, таких як Albion Online, де ефективна взаємодія між користувачами та обмін досвідом є важливими складовими успішного ігрового процесу.

До універсальних платформ належать соціальні та комунікаційні сервіси, такі як Reddit та Discord, які забезпечують можливість обговорення ігрових тем, обміну досвідом та швидкої взаємодії між користувачами. Зокрема, Reddit виступає у ролі форуму, де користувачі можуть створювати тематичні обговорення,

публікувати новини та отримувати зворотний зв’язок від спільноти. Discord, у свою чергу, забезпечує голосову та текстову комунікацію в реальному часі, що робить його зручним інструментом для координації дій гравців та організації ігрових груп.

Окрему категорію становлять фан-сайти та тематичні портали, присвячені конкретним іграм. Такі ресурси зазвичай містять структуровану інформацію про ігровий процес, включаючи гайди, новини, аналітичні матеріали та бази знань. Вони створюються як ентузіастами, так і професійними командами, та орієнтовані на більш глибоке розкриття механік гри.

Вікі-ресурси дозволяють користувачам спільно створювати та редагувати інформацію, формуючи повноцінні бази знань щодо ігрових механік, предметів, персонажів та стратегій. Такі платформи є особливо корисними для нових гравців, оскільки забезпечують систематизований доступ до великого обсягу інформації.

Таблиця 1.1

Функціональні можливості веб-платформ для геймерів

№	Функція	Опис	Приклад реалізації
1	Реєстрація та авторизація	Забезпечує створення облікового запису користувача та доступ до персоналізованих функцій	JWT, OAuth
2	Профіль користувача	Містить інформацію про гравця, його активність, публікації та досягнення	Discord, Reddit
3	Публікація контенту	Дозволяє створювати гайди, білді, новини та інші матеріали	Фан-сайти, блоги
4	Перегляд та пошук	Забезпечує доступ до контенту та можливість його швидкого пошуку	Wiki-платформи
5	Коментування та взаємодія	Дозволяє залишати коментарі, оцінювати та обговорювати матеріали	Reddit
6	Система рейтингів	Формує оцінку контенту або користувачів (лайки, дизлайки, рейтинг)	Reddit, форуми
7	Новини та оновлення	Відображає актуальні новини гри та події	Офіційні сайти, фан-портали
8	Інтеграція з API	Отримання актуальних ігрових даних (статистика, предмети, персонажі)	Albion API

Продовження таблиці 1.1

№	Функція	Опис	Приклад реалізації
9	Адміністрування	Управління контентом, користувачами та модерація	CMS системи
10	Адаптивний інтерфейс	Забезпечує коректну роботу на різних пристроях (ПК, мобільні)	React + TailwindCSS

Основні функціональні можливості веб-платформ для геймерів формуються з урахуванням потреб користувачів у взаємодії, обміні інформацією та створенні контенту. Система профілів користувачів, яка дозволяє ідентифікувати учасників платформи, відображати їх активність, збережені матеріали та досягнення. Наявність персоналізованого профілю сприяє формуванню спільноти та підвищує рівень залученості користувачів [2].

Функціонал публікації контенту, зокрема гайдів та білдів, що дозволяє гравцям ділитися власним досвідом, стратегіями та підходами до проходження гри. Такі матеріали є особливо цінними для нових користувачів, оскільки допомагають швидше адаптуватися до ігрового середовища та зрозуміти його механіки. Окрім цього, веб-платформи забезпечують можливість публікації новин, які відображають актуальні події, оновлення гри та зміни в ігровому балансі.

Механізми взаємодії між користувачами, зокрема система коментарів та оцінювання контенту за допомогою лайків. Це дозволяє формувати зворотний зв’язок, визначати популярність матеріалів та стимулює користувачів до створення якісного контенту. Таким чином, інтерактивність виступає ключовим елементом сучасних веб-платформ для геймерів.

Специфіка веб-платформ для гри Albion Online визначається особливостями самої гри, яка має складну економічну систему, орієнтовану на взаємодію між гравцями. Віртуальна економіка гри базується на принципах попиту та пропозиції, що зумовлює необхідність наявності інструментів для аналізу ринку, цін та ресурсів [3]. Крім того, значну роль відіграють білді – індивідуальні конфігурації спорядження персонажа, які визначають стиль гри та ефективність у різних

ситуаціях. Наявність платформи для публікації та обговорення білдів є важливим фактором підвищення конкурентоспроможності гравців.

Веб-платформа може слугувати інструментом координації дій учасників гільдії, обміну інформацією та планування спільних активностей. Таким чином, врахування специфіки гри дозволяє створити більш ефективну та орієнтовану на користувача веб-платформу. Основні функції веб-платформ для онлайн-ігор доцільно узагальнити у вигляді табл. 1.2.

Таблиця 1.2

Основні функції веб-платформ для онлайн-ігор

№	Функція	Призначення	Значення для користувача
1	Профілі користувачів	Зберігання інформації про користувача, його активність та досягнення	Персоналізація досвіду, ідентифікація в системі
2	Публікація гайдів та білдів	Створення та розміщення користувацького контенту	Обмін досвідом та стратегіями гри
3	Новини та оновлення	Інформування про події, патчі та зміни в грі	Актуальність інформації для гравців
4	Система коментарів	Обговорення контенту між користувачами	Зворотний зв’язок та комунікація
5	Система оцінювання (лайки)	Оцінка якості контенту	Визначення популярності матеріалів
6	Пошук та фільтрація	Швидкий доступ до потрібної інформації	Економія часу користувача
7	Інтеграція з ігровими сервісами	Отримання даних через API (персонажі, предмети)	Актуальні та динамічні дані
8	Система ролей	Розмежування прав доступу (користувач, адміністратор)	Безпека та контроль контенту
9	Адаптивний інтерфейс	Підтримка різних пристроїв	Зручність використання
10	Адміністрування контенту	Модерація та управління платформою	Якість і достовірність інформації

Як видно з табл. 1.2, сучасні веб-платформи для онлайн-ігор поєднують у собі функції управління контентом, взаємодії користувачів та доступу до актуальної

інформації. Реалізація зазначених можливостей забезпечує зручність використання системи, підвищує рівень залученості користувачів та сприяє формуванню активної ігрової спільноти. Таким чином, наявність розглянутих функцій є необхідною умовою для створення ефективної веб-платформи для гравців Albion Online.

1.2 Аналіз сучасних технологій frontend та backend для створення веб-застосунків

Frontend є однією з ключових складових сучасного веб-застосунку, оскільки саме він забезпечує взаємодію користувача з системою, формує візуальне представлення даних і впливає на загальне сприйняття програмного продукту. Для веб-платформи, орієнтованої на гравців Albion Online, frontend має не лише забезпечувати коректне відображення контенту, але й підтримувати швидку навігацію, зручний доступ до новин, гайдів, білдів і профілів користувачів [4]. У такій системі особливу роль відіграють динамічність інтерфейсу, адаптивність дизайну та здатність швидко оновлювати дані без перевантаження сторінки. Саме тому вибір frontend-технологій є принципово важливим для досягнення високої якості кінцевого продукту.

При розробці сучасних веб-платформ найчастіше використовуються бібліотеки та фреймворки, які дозволяють створювати інтерактивні користувацькі інтерфейси, організовувати логіку відображення компонентів та ефективно працювати зі станом застосунку. Серед таких рішень особливе місце займає React, який набув значного поширення завдяки своїй гнучкості, продуктивності та зручності використання. Для стилізації інтерфейсу та прискорення процесу верстки доцільним є застосування TailwindCSS, який дозволяє формувати сучасний адаптивний дизайн без необхідності створення великої кількості окремих CSS-файлів.

React є однією з найбільш популярних JavaScript-бібліотек для створення користувацьких інтерфейсів. Його основне призначення полягає у побудові

динамічних односторінкових застосунків, у яких інтерфейс формується з окремих незалежних компонентів. Такий підхід особливо ефективний при розробці складних веб-платформ, де інтерфейс містить багато повторюваних або логічно відокремлених елементів, наприклад навігаційне меню, блок новин, список гайдів, картки білдів, форму авторизації, профіль користувача, секцію коментарів тощо.

Компонентний підхід дозволяє поділити інтерфейс на окремі функціональні частини, кожна з яких відповідає за власну логіку та відображення. Це значно спрощує підтримку коду, його повторне використання та подальше розширення функціоналу системи. Наприклад, один раз створений компонент картки гайда може використовуватись у різних розділах платформи без необхідності дублювання коду [5]. Такий підхід не лише покращує структуру проєкту, а й підвищує зручність його супроводу.

React доцільно використовувати завдяки зручній роботі зі станом застосунку. У будь-якій сучасній платформі користувацький інтерфейс постійно взаємодіє з даними: зберігає авторизаційний стан користувача, відображає нові записи, фільтрує контент, реагує на натискання кнопок, форму пошуку чи зміну параметрів. React дозволяє ефективно організувати цю логіку за допомогою хуків, локального стану компонентів та глобального керування даними. У результаті інтерфейс стає більш передбачуваним, а код — логічно структурованим.

Серед причин вибору React варто також виділити його широку поширеність і активну підтримку спільнотою розробників. Наявність великої кількості бібліотек, документації, готових рішень та прикладів значно спрощує процес розробки та дозволяє швидше реалізовувати потрібний функціонал. Для дипломного проєкту це є суттєвою перевагою, оскільки дає змогу зосередитися не на технічних труднощах інтеграції базових можливостей, а на розробці корисного прикладного функціоналу системи.

React обрано як основну frontend-технологію через його компонентну архітектуру, високу продуктивність, зручність у роботі зі станом, підтримку динамічного оновлення інтерфейсу, можливість створення односторінкових застосунків та хорошу масштабованість. Сукупність цих характеристик робить

React доцільним вибором для розробки веб-платформи, орієнтованої на активну ігрову спільноту та регулярне оновлення контенту.

Важливою складовою frontend-розробки є не лише логіка роботи інтерфейсу, а й його візуальне оформлення. Якісний інтерфейс повинен бути зрозумілим, адаптивним, сучасним і зручним для користувача. Для досягнення цих цілей у процесі розробки веб-платформи доцільно використовувати TailwindCSS — сучасний CSS-фреймворк, який базується на принципі utility-first. Це означає, що стилізація інтерфейсу виконується за допомогою невеликих службових класів, які задають конкретні властивості елементів: відступи, кольори, розміри, вирівнювання, типографіку, межі, тіні та інші параметри [6].

Однією з головних причин вибору TailwindCSS є швидкість верстки. У традиційному підході розробник створює окремі CSS-класи, описує для них стилі у зовнішніх файлах, а потім підключає ці класи до HTML-структури. У випадку з TailwindCSS велика частина стилізації здійснюється безпосередньо в розмітці компонентів за допомогою готових utility-класів. Це дозволяє значно скоротити час розробки інтерфейсу та спростити процес внесення змін у зовнішній вигляд окремих елементів.

Для дипломного проєкту, де важливо не лише реалізувати функціональність, але й продемонструвати сучасний підхід до розробки, TailwindCSS є особливо доречним. Він дає змогу швидко побудувати акуратний, структурований і візуально цілісний інтерфейс без значного перевантаження проєкту окремими CSS-файлами. Це особливо важливо для веб-платформи Albion Online, де потрібно оформити різні типи сторінок: головну сторінку з новинами, сторінки окремих гайдів, форми входу та реєстрації, картки білдів, профілі користувачів, панель адміністратора тощо.

Веб-платформи повинні коректно працювати на пристроях із різними розмірами екранів — настільних комп’ютерах, ноутбуках, планшетах і смартфонах. TailwindCSS має вбудовану систему адаптивних класів, що дозволяє легко змінювати вигляд інтерфейсу залежно від ширини екрана. У результаті один і той самий компонент може зберігати функціональність і привабливість як на великому

моніторі, так і на мобільному пристрої [7].

TailwindCSS сприяє уніфікації стилю застосунку. Використання стандартизованих класів допомагає дотримуватися єдиної візуальної логіки на всіх сторінках платформи. Це важливо для створення цілісного користувацького досвіду, коли всі елементи інтерфейсу виглядають узгоджено та передбачувано. Для ігрової веб-платформи така узгодженість є важливою, адже користувач повинен легко орієнтуватися в розділах, сприймати інтерфейс як логічно побудовану систему та не витрачати зайвий час на розуміння структури сторінок.

Додатковою перевагою є те, що TailwindCSS добре інтегрується з React. Оскільки React будує інтерфейс із компонентів, а TailwindCSS дозволяє швидко стилізувати ці компоненти, поєднання цих технологій забезпечує високу ефективність розробки. Розробник може створювати повноцінні інтерфейсні блоки, які одночасно містять логіку та стилі, що спрощує структуру проєкту та робить код більш зрозумілим.

TailwindCSS також дозволяє зменшити кількість зайвого CSS-коду. У традиційних стилях часто виникає проблема накопичення невикористовуваних класів, дублювання стилів та складності супроводу великої кількості правил [8]. Використання utility-first підходу дає змогу уникнути цієї проблеми, оскільки стилізація здійснюється точково та безпосередньо там, де вона потрібна. Це позитивно впливає на чистоту коду та спрощує його підтримку.

Для веб-платформи Albion Online важливо також забезпечити привабливий зовнішній вигляд контенту. Гайди, новини, білді та профілі користувачів повинні бути оформлені таким чином, щоб користувачу було зручно сприймати інформацію, швидко знаходити ключові блоки та взаємодіяти з інтерфейсом. TailwindCSS дозволяє легко налаштовувати типографіку, сітки, відступи, кольорові акценти та візуальні стани елементів, що є важливим для побудови сучасного ігрового порталу.

Для реалізації frontend-частини веб-платформи для гравців Albion Online доцільно використовувати React та TailwindCSS. React забезпечує побудову динамічного, компонентного та масштабованого інтерфейсу, що особливо важливо

для платформи з великою кількістю інтерактивного контенту. TailwindCSS, у свою чергу, дозволяє швидко реалізувати сучасний адаптивний дизайн і спрощує процес верстки. Поєднання цих технологій дозволяє створити зручний, продуктивний та візуально цілісний користувацький інтерфейс, який відповідає вимогам сучасних веб-застосунків.

Backend відповідає за обробку запитів, бізнес-логіку застосунку та взаємодію з базою даних. Для розробки веб-платформи для гравців Albion Online доцільно використовувати Node.js у поєднанні з фреймворком Express.

Node.js забезпечує виконання JavaScript на сервері, що дозволяє використовувати єдину мову програмування як на frontend, так і на backend. Це спрощує розробку та підтримку проєкту. Крім того, Node.js добре підходить для обробки великої кількості одночасних запитів, що є важливим для веб-платформи з активною аудиторією [9].

Express є мінімалістичним фреймворком для Node.js, який дозволяє швидко створювати серверну логіку, маршрути (routes) та API. Він спрощує організацію коду та забезпечує гнучкість при реалізації функціоналу, такого як обробка запитів користувачів, авторизація та робота з контентом.

Для взаємодії між клієнтською та серверною частинами використовується REST API. Такий підхід передбачає обмін даними у форматі JSON через HTTP-запити. REST API дозволяє чітко розділити frontend і backend, забезпечує масштабованість системи та спрощує інтеграцію з іншими сервісами, зокрема API гри Albion Online.

Рис. 1.1 демонструє, що frontend-частина веб-платформи взаємодіє із сервером через REST API, а backend забезпечує обробку запитів, роботу з базою даних PostgreSQL та інтеграцію із зовнішнім API гри. Такий підхід забезпечує модульність, масштабованість і зручність подальшого розширення функціоналу системи.

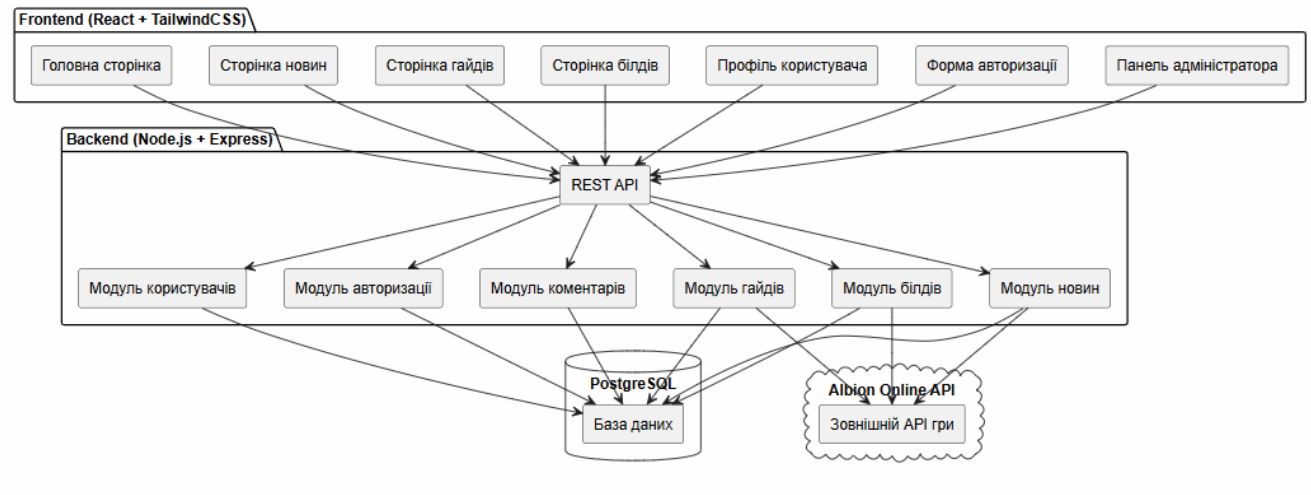


Рис. 1.1 Взаємодія frontend і backend частин системи

Для зберігання даних у веб-платформі використовується реляційна система керування базами даних PostgreSQL.

PostgreSQL обрано завдяки його надійності, підтримці складних запитів та чіткій структурі даних. У веб-платформі необхідно працювати з пов’язаними сутностями, такими як користувачі, гайди, новини та коментарі, що зручно реалізується саме в реляційній базі даних [10].

На відміну від MongoDB, яка є нереляційною базою даних, PostgreSQL краще підходить для систем, де важлива цілісність даних та зв’язки між таблицями. Наприклад, один користувач може мати багато публікацій, коментарів та оцінок, і ці зв’язки зручно реалізовувати через SQL-структуру.

Крім того, PostgreSQL забезпечує:

- підтримку транзакцій;
- високу стабільність;
- ефективну роботу зі складними запитами;
- можливість масштабування.

Використання Node.js, Express та REST API забезпечує ефективну реалізацію серверної частини, а PostgreSQL — надійне зберігання та обробку даних у веб-платформі для гравців Albion Online.

Node.js у поєднанні з Express є найбільш доцільним вибором для розробки

веб-платформи, оскільки забезпечує високу продуктивність, гнучкість та ефективну інтеграцію з frontend-технологіями (табл. 1.3).

Таблиця 1.3

Порівняння backend технологій для веб-застосунків

Критерій	Node.js (Express)	Django (Python)	Laravel (PHP)
Мова програмування	JavaScript	Python	PHP
Продуктивність	Висока (асинхронність)	Середня	Середня
Швидкість розробки	Висока	Висока	Висока
Масштабованість	Висока	Середня	Середня
Архітектура	Гнучка (REST API)	MVC	MVC
Робота з реальним часом	Відмінна	Обмежена	Обмежена
Простота навчання	Середня	Висока	Висока
Підходить для SPA	Так (ідеально з React)	Так	Так
Екосистема	Дуже велика (npm)	Велика	Велика
Використання у проєкті	Обрано	Не обрано	Не обрано

1.3 Порівняльний аналіз інструментів для розробки інтерактивних веб-систем

У процесі розробки сучасних веб-застосунків важливим етапом є вибір технологій, які забезпечать необхідний рівень продуктивності, масштабованості та зручності розробки. Для реалізації інтерактивних веб-систем широко використовуються різні frontend та backend інструменти, серед яких найбільш популярними є React, Angular, Vue для клієнтської частини та Node.js, Django, Laravel для серверної частини [11].

У сфері frontend-розробки React, Angular та Vue є основними інструментами для створення динамічних інтерфейсів. React відзначається гнучкістю та компонентною архітектурою, що дозволяє ефективно будувати масштабовані застосунки. Angular є повноцінним фреймворком із жорсткою структурою, що забезпечує стандартизацію, але потребує більше часу для освоєння. Vue, у свою

чергу, поєднує простоту та гнучкість, що робить його зручним для невеликих та середніх проєктів.

Щодо backend-технологій, Node.js забезпечує продуктивність завдяки асинхронній моделі обробки запитів, що є необхідним для систем із великою кількістю одночасних користувачів. Django характеризується високим рівнем безпеки та швидкою розробкою завдяки вбудованим інструментам. Laravel забезпечує зручність створення веб-застосунків завдяки зрозумілій архітектурі та великій кількості готових рішень.

Для обґрунтування вибору технологій доцільно провести їх порівняння за ключовими критеріями, такими як швидкість роботи, масштабованість та простота використання (табл. 1.4).

Таблиця 1.4

Порівняння інструментів для розробки веб-систем

Технологія	Тип	Швидкість	Масштабованість	Простота використання
React	Frontend	Висока	Висока	Середня
Angular	Frontend	Середня	Висока	Низька
Vue	Frontend	Висока	Середня	Висока
Node.js (Express)	Backend	Висока	Висока	Середня
Django	Backend	Середня	Середня	Висока
Laravel	Backend	Середня	Середня	Висока

Аналіз даних, свідчить про те, що React є хорошим вибором для реалізації frontend-частини веб-платформи завдяки високій продуктивності та масштабованості. Для серверної частини доцільно використовувати Node.js, оскільки він забезпечує ефективну обробку запитів та добре інтегрується з сучасними frontend-технологіями. Таким чином, обраний технологічний стек відповідає вимогам до розробки інтерактивної веб-платформи для гравців Albion Online.

Висновок до розділу 1

Проведено аналіз теоретичних основ розробки веб-платформ для ігрових спільнот. Розглянуто особливості таких платформ, їх основні функціональні можливості та роль у забезпеченні взаємодії між користувачами. Встановлено, що сучасні веб-платформи для геймерів поєднують у собі функції соціальних мереж, інформаційних ресурсів та систем управління контентом, що дозволяє створювати єдине середовище для обміну досвідом і публікації матеріалів.

Проведений аналіз сучасних технологій frontend та backend показав, що для створення інтерактивних веб-застосунків доцільно використовувати React та TailwindCSS для клієнтської частини, а також Node.js з використанням Express для серверної логіки. Обґрунтовано вибір PostgreSQL як надійної та ефективної системи керування базами даних для зберігання структурованої інформації.

У результаті порівняльного аналізу інструментів розробки встановлено, що обраний технологічний стек забезпечує необхідний рівень продуктивності, масштабованості та зручності реалізації. Таким чином, проведене дослідження підтверджує доцільність використання зазначених технологій для розробки веб-платформи для гравців Albion Online, що стане основою для подальшого проєктування та програмної реалізації системи.

2 ПРОЄКТУВАННЯ ВЕБ-ПЛАТФОРМИ ДЛЯ ГРАВЦІВ ALBION ONLINE

2.1 Формування функціональних вимог до веб-платформи

Після проведення аналізу предметної області та обґрунтування вибору технологій наступним етапом є формування вимог до розроблюваної веб-платформи для гравців Albion Online. Визначення функціональних та нефункціональних вимог дозволяє чітко окреслити можливості системи, встановити її основні характеристики та забезпечити відповідність очікуванням користувачів. Саме на основі сформованих вимог здійснюється подальше проєктування архітектури, структури бази даних та програмної реалізації веб-платформи [12].

Функціональні вимоги визначають основні можливості системи та описують дії, які користувач може виконувати під час роботи з веб-платформою. Для розроблюваної веб-платформи для гравців Albion Online функціональні можливості спрямовані на забезпечення зручної взаємодії користувачів, публікації контенту та доступу до актуальної інформації.

До основних функціональних вимог належать: реєстрація та авторизація користувачів, що забезпечують доступ до персоналізованих можливостей системи; створення та редагування гайдів і білдів, що дозволяє користувачам ділитися власним досвідом; перегляд новин, який забезпечує доступ до актуальної інформації про гру; пошук по платформі, що дозволяє швидко знаходити необхідний контент; а також можливість коментування матеріалів для забезпечення взаємодії між користувачами (табл 2.1).

Таблиця 2.1

Функціональні вимоги до веб-платформи

№	Функціональна вимога	Опис
1	Реєстрація користувача	Можливість створення нового облікового запису
2	Авторизація (логін)	Вхід користувача до системи з використанням логіну та пароля
3	Створення гайдів	Додавання нових матеріалів (гайдів, білдів)
4	Редагування контенту	Можливість змінювати власні публікації
5	Перегляд новин	Доступ до актуальних новин гри
6	Пошук інформації	Пошук гайдів, новин та користувачів
7	Коментування	Можливість залишати коментарі до публікацій
8	Перегляд профілю	Доступ до інформації про користувача
9	Система ролей	Розмежування прав доступу (користувач, адміністратор)
10	Видалення контенту	Можливість видалення власних або модерованих матеріалів

Функціональні можливості веб-платформи охоплюють основні потреби користувачів, забезпечуючи створення, перегляд та взаємодію з контентом. Реалізація зазначених вимог дозволить сформувати зручне та ефективне середовище для гравців Albion Online.

Нефункціональні вимоги визначають якісні характеристики системи, що впливають на її ефективність, надійність та зручність використання. Для веб-платформи для гравців Albion Online такі вимоги є не менш важливими, ніж функціональні, оскільки вони забезпечують стабільну роботу системи, захист даних користувачів та комфортну взаємодію з інтерфейсом.

До основних нефункціональних вимог належать: висока швидкість обробки запитів і завантаження сторінок, що забезпечує оперативний доступ до контенту; безпека, яка передбачає захист даних користувачів та контроль доступу до системи; масштабованість, що дозволяє системі ефективно працювати при зростанні кількості користувачів; а також зручність використання (UX), яка забезпечує інтуїтивно зрозумілий інтерфейс і комфортну взаємодію з платформою (табл. 2.2).

Таблиця 2.2

Нефункціональні вимоги до веб-платформи

№	Нефункціональна вимога	Опис
1	Швидкість	Швидке завантаження сторінок та обробка запитів користувачів
2	Безпека	Захист персональних даних, використання авторизації та контролю доступу
3	Масштабованість	Можливість обробки великої кількості користувачів без втрати продуктивності
4	UX (зручність)	Інтуїтивно зрозумілий інтерфейс та простота навігації
5	Надійність	Стабільна робота системи без збоїв
6	Адаптивність	Коректна робота на різних пристроях
7	Продуктивність	Оптимізація ресурсів сервера та клієнта
8	Доступність	Можливість доступу до системи 24/7
9	Сумісність	Підтримка різних браузерів
10	Розширюваність	Можливість додавання нового функціоналу

Нефункціональні вимоги забезпечують якість роботи веб-платформи та визначають її ефективність у реальних умовах використання. Їх дотримання є необхідною умовою для створення стабільної, безпечної та зручної системи для гравців Albion Online.

2.2 Проєктування архітектури клієнт-серверної системи

Для реалізації вебзастосунку Albion Portal обрано класичну клієнт-серверну архітектуру з розподілом відповідальності між трьома логічними рівнями:

1. рівень представлення (React-клієнт);
2. прикладний рівень (Node.js/Express API);
3. рівень даних (PostgreSQL).

Такий підхід забезпечує масштабованість, спрощує супровід і дозволяє незалежно розвивати фронтенд та бекенд.

Клієнтська частина реалізована як SPA на React із маршрутизацією сторінок /builds, /news, /login, /register, /admin, що підтверджується у App.jsx (line 18). Для роботи з API використовується окремий axios-екземпляр із базовою адресою сервера та автоматичним додаванням JWT-токена до заголовка Authorization (axios.js (line 3), axios.js (line 7)) [16].

Стан авторизації централізовано зберігається в AuthContext, де виконуються збереження/видалення токена в localStorage і декодування payload для формування об’єкта поточного користувача (AuthContext.jsx (line 14)).

Серверна частина побудована на Express і поділена на окремі модулі маршрутів: /auth, /builds, /news, /admin (index.js (line 20)).

Для захисту ресурсів застосовано middleware requireAuth та requireAdmin, які перевіряють валідність JWT та роль користувача (auth.js (line 6), auth.js (line 23)).

Модуль auth реалізує реєстрацію та вхід із хешуванням пароля (bcrypt) і видачею JWT на 7 діб (routes/auth.js (line 28), routes/auth.js (line 68)).

Модулі builds і news підтримують публічне читання контенту та адміністративні CRUD-операції з публікацією/закріпленням (routes/builds.js (line 12), routes/news.js (line 6)) [17].

Адміністративний модуль додатково надає керування користувачами і статистику (routes/admin.js (line 7), routes/admin.js (line 79)).

На рівні БД у проєкті явно зафіксовано таблиці users та news, а також рольову модель admin/user (migrate.sql (line 2), migrate.sql (line 12)).

Логічно в архітектурі також використовується сутність builds, що видно з API-контрактів і SQL-запитів у модулі білдів (routes/builds.js (line 15), routes/builds.js (line 81)).

У результаті отримано модульну клієнт-серверну систему, де:

- клієнт відповідає за UX, стан сесії і формування HTTP-запитів;
- сервер реалізує бізнес-логіку, авторизацію та валідацію;
- база даних забезпечує цілісність збережених сутностей і зв’язків [17].

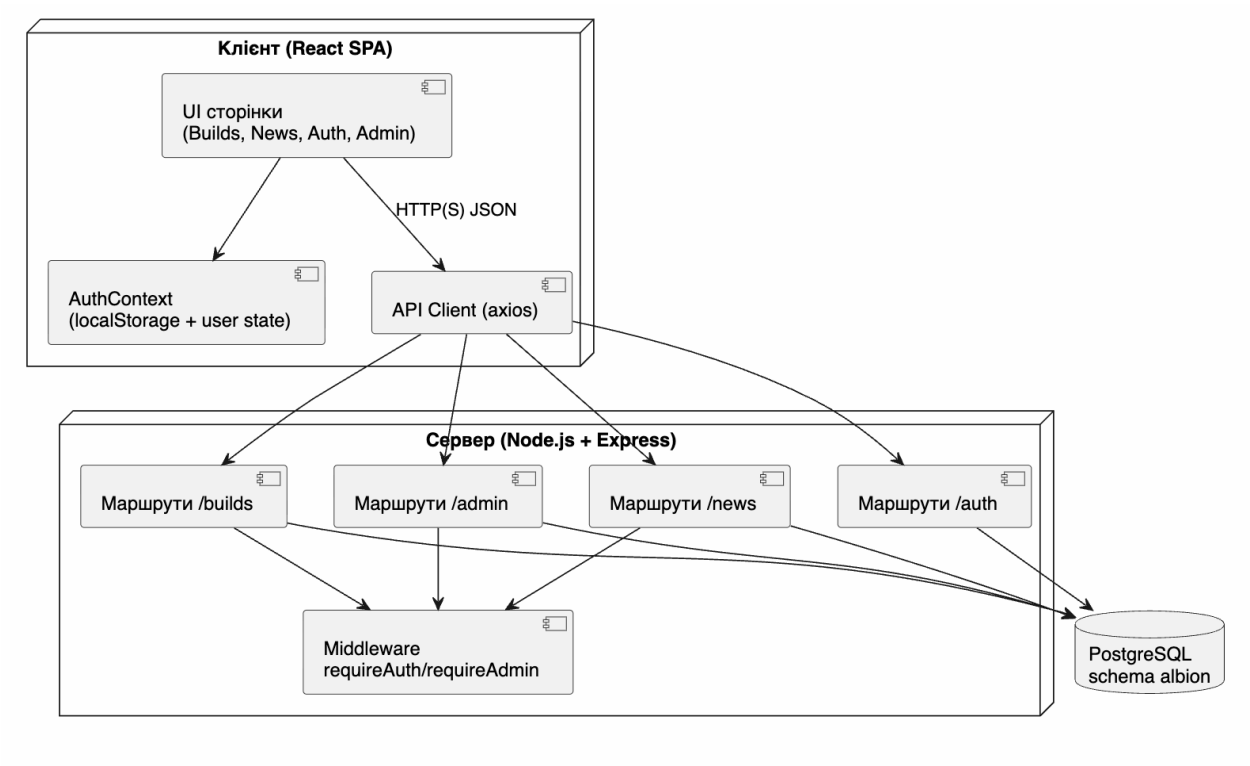


Рис. 2.1 Загальна архітектура клієнт-серверної системи Albion Portal

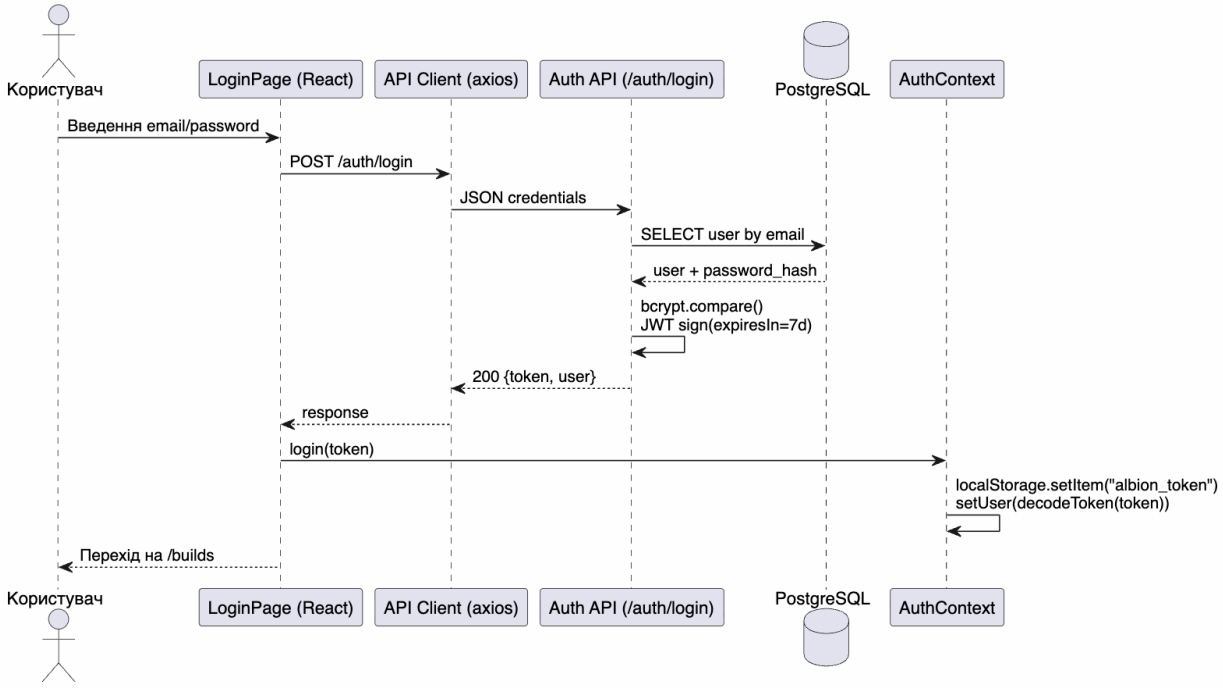


Рис. 2.2 Діаграма послідовності автентифікації користувача (JWT)

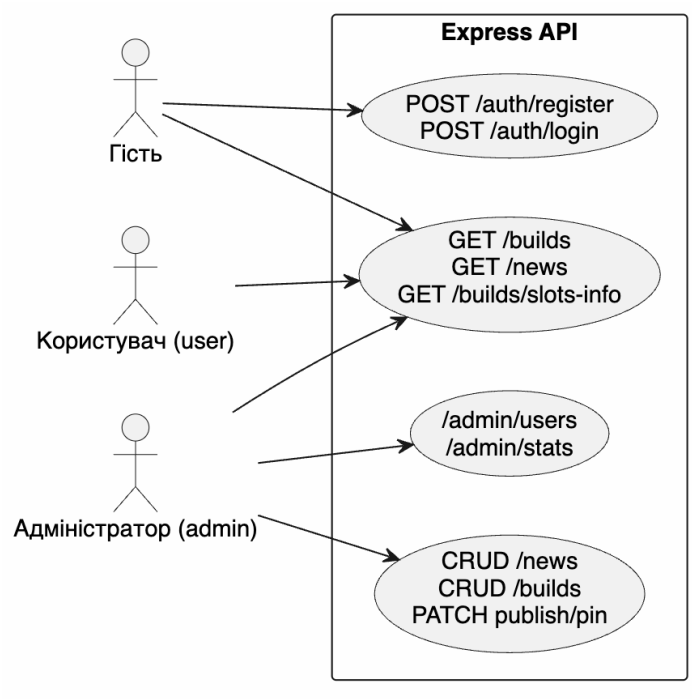


Рис. 2.3 Модель доступу за ролями в API

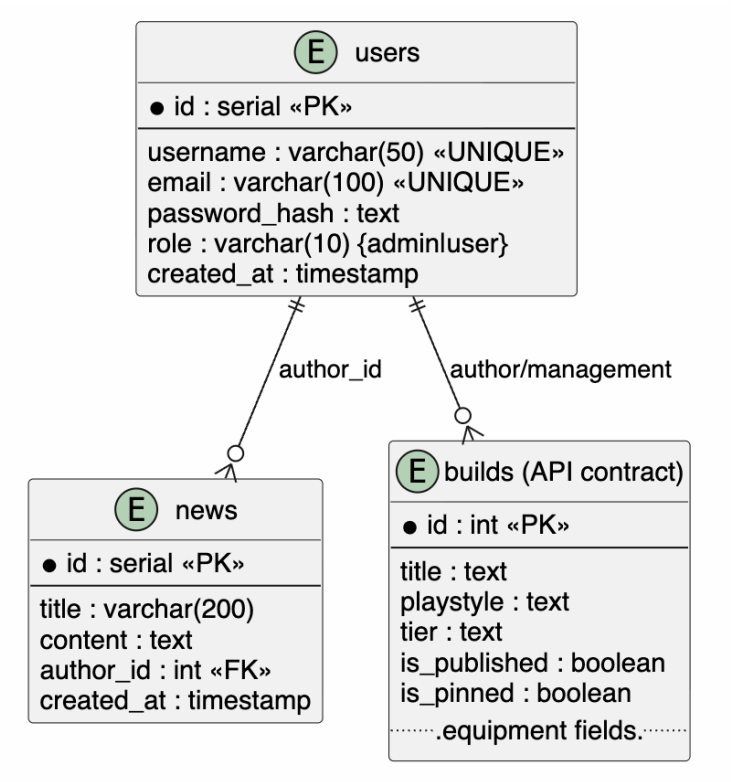


Рис. 2.4 Логічна модель даних (ядро)

На клієнтському боці маршрутизація реалізована через react-router-dom, а доступ до API здійснюється через централізований HTTP-клієнт axios. На сервері

реалізовано окремі модулі маршрутів (/builds, /news, /auth, /admin) і налаштовано CORS для локального клієнта [18].

Фрагмент маршрутизації SPA:

```
<Routes>
  <Route path="/" element={<Navigate to="/builds" replace />} />
  <Route path="/builds" element={<BuildsPage />} />
  <Route path="/news" element={<NewsPage />} />
  <Route path="/login" element={<LoginPage />} />
  <Route path="/register" element={<RegisterPage />} />
  <Route path="/admin" element={<AdminPage />} />
</Routes>
```

Фрагмент клієнта API:

```
const api = axios.create({
baseURL: "http://localhost:3001",
});
```

Фрагмент підключення серверних маршрутів:

```
app.use("/builds", buildsRoutes);
app.use("/news", newsRoutes);
app.use("/auth", authRoutes);
app.use("/admin", adminRoutes);
```

Отже, архітектура реалізує модель, у якій React-вузол відповідає за інтерфейс і відправлення запитів, Express - за бізнес-логіку, а БД - за персистентне зберігання даних [18].

Проєктування механізму автентифікації (JWT)

Автентифікація побудована за токен-орієнтованою схемою: після успішного входу сервер формує JWT, клієнт зберігає його у localStorage і додає до кожного наступного захищеного запиту [17-19].

Фрагмент логіки входу на сервері:

```
router.post("/login", async (req, res) => {
  const { email, password } = req.body;
```

```
  // пошук користувача, перевірка bcrypt...
  const token = jwt.sign(
    { id: user.id, username: user.username, role: user.role },
    JWT_SECRET,
    { expiresIn: "7d" }
  );
  res.json({ token, user: { id: user.id, username: user.username, role: user.role } });
});
```

Фрагмент збереження токена на клієнті:

```
function login(token) {
  localStorage.setItem("albion_token", token);
  setUser(decodeToken(token));
}
```

Фрагмент автоматичної підстановки токена в заголовки:

```
api.interceptors.request.use((config) => {
  const token = localStorage.getItem("albion_token");
  if (token) {
    config.headers["Authorization"] = `Bearer ${token}`;
  }
  return config;
});
```

Таким чином, механізм JWT забезпечує безстанову (stateless) автентифікацію та зручний контроль доступу в API.

Модель розмежування доступу за ролями

У системі застосовано рольову модель доступу (RBAC) з двома ролями: user і admin. Перевірка виконується в middleware-рівні: спочатку валідність токена (requireAuth), далі — відповідність ролі (requireAdmin).

Фрагмент перевірки авторизації:

```
function requireAuth(req, res, next) {
  const header = req.headers["authorization"];
```

```
  if (!header || !header.startsWith("Bearer ")) {
    return res.status(401).json({ error: "Необхідна авторизація" });
  }
  const token = header.slice(7);
  const payload = jwt.verify(token, JWT_SECRET);
  req.user = payload;
  next();
}
```

Фрагмент перевірки ролі:

```
function requireAdmin(req, res, next) {
  if (req.user?.role !== "admin") {
    return res.status(403).json({ error: "Доступ заборонено: потрібна роль admin" });
  }
  next();
}
```

Фрагмент захисту адміністративного модуля:

```
// Всі роути захищені — тільки admin
router.use(requireAuth, requireAdmin);
```

Отже, доступ до критичних операцій (керування користувачами, контентом, статистикою) обмежений лише роллю адміністратора.

Логічна модель даних системи

Ядро моделі даних побудоване на таблицях users і news, а також сутності builds, яка активно використовується серверними маршрутами. Таблиця користувачів містить роль, унікальні поля і хеш пароля, що забезпечує безпечне зберігання облікових даних [18, 19].

Фрагмент DDL для users:

```
CREATE TABLE IF NOT EXISTS albion.users (
  id SERIAL PRIMARY KEY,
  username VARCHAR(50) UNIQUE NOT NULL,
```

```
  email VARCHAR(100) UNIQUE NOT NULL,
  password_hash TEXT NOT NULL,
  role VARCHAR(10) NOT NULL DEFAULT 'user' CHECK (role IN ('admin', 'user')),
  created_at TIMESTAMP DEFAULT NOW()
);
```

Фрагмент DDL для news:

```
CREATE TABLE IF NOT EXISTS albion.news (
  id SERIAL PRIMARY KEY,
  title VARCHAR(200) NOT NULL,
  content TEXT NOT NULL,
  author_id INTEGER REFERENCES albion.users(id) ON DELETE SET NULL,
  created_at TIMESTAMP DEFAULT NOW()
);
```

Фрагмент вставки запису builds у серверній логіці:

```
INSERT INTO albion.builds (
  title, ..., is_published, is_pinned
) VALUES (... ) RETURNING *
```

Логічна структура БД підтримує ключові бізнес-сценарії системи: реєстрацію та ролі користувачів, публікацію новин, а також створення і адміністрування білдів [19].

2.3 Моделювання бази даних та структури взаємодії компонентів

У цьому підрозділі формалізовано модель даних системи Albion Portal та описано структуру взаємодії клієнтських і серверних компонентів під час виконання основних бізнес-сценаріїв.

Концептуальна модель даних

Основними сутностями системи є [20]:

1. users - облікові записи користувачів (ідентифікація, роль, дата створення).

2. news - новини порталу з прив’язкою до автора.
3. builds - ігрові білди (набір спорядження, параметри публікації, категоризація).

Ключові зв’язки:

1. users (1) - (M) news через news.author_id.
2. users (1) - (M) builds (автор/адміністрування контенту на рівні бізнес-логіки).
3. Роль users.role визначає права доступу до операцій CRUD та адміністративних маршрутів.

Логічна модель БД

Логічна модель реалізована в PostgreSQL-схемі albion. Для сутності users задано обмеження унікальності username, email і рольовий CHECK (admin | user), що забезпечує цілісність довідникових значень. Для сутності news визначено зовнішній ключ на users з політикою ON DELETE SET NULL, яка дозволяє зберігати новину навіть при видаленні автора [20].

Приклад логічних обмежень:

- PRIMARY KEY для ідентифікації записів;
- UNIQUE для запобігання дублюванню акаунтів;
- FOREIGN KEY для контролю міжтабличних зв’язків;
- CHECK для обмеження домену ролей.

Фізична організація даних

На фізичному рівні структура орієнтована на типові операції читання та модерації контенту:

1. Вибірка публічних записів з фільтрацією is_published = true.
2. Сортування за ознакою закріплення (is_pinned) і новизною (id DESC / created_at DESC).
3. Розділення публічних і адміністративних запитів (/news vs /news/all, /builds vs /builds/all).

Доцільні індекси для продуктивності:

- news(is_published, is_pinned, id DESC);
- builds(is_published, is_pinned, id DESC);
- users(email), users(username) (фактично вже покриті UNIQUE).

Структура взаємодії компонентів

Взаємодія компонентів реалізована за схемою:

1. React UI формує дію користувача (вхід, перегляд, створення/редагування).
2. API client (axios) надсилає HTTP-запит на Express API.
3. Middleware (requireAuth, requireAdmin) перевіряє JWT та роль.
4. Route handler виконує валідацію, SQL-запит і повертає JSON-відповідь.
5. UI оновлює стан і відображення сторінки.

Для автентифікованих запитів JWT зберігається в localStorage і автоматично додається до заголовка Authorization: Bearer <token>.

Сценарна взаємодія (приклад "Вхід у систему")

1. Користувач вводить email/password у формі входу.
2. Клієнт надсилає POST /auth/login.
3. Сервер перевіряє користувача в БД і порівнює пароль через bcrypt.
4. У разі успіху генерується JWT (з id, username, role).
5. Клієнт зберігає токен, декодує payload і встановлює сесію користувача.
6. Подальші захищені запити проходять через рольову перевірку.

На рисунку 2.5 продемонстровано ER-модель даних системи Albion Portal. Діаграма відображає ключові сутності предметної області (users, news, builds), їхні атрибути та типи зв'язків між ними. Для таблиці users показано рольову ознаку role, що використовується в механізмі авторизації доступу. Зв'язок users (1) — (M) news реалізується через зовнішній ключ author_id, що дозволяє встановити автора кожної новини. Сутність builds представляє контентну частину порталу (білдів) із полями стану публікації (is_published) та закріплення (is_pinned), які визначають порядок і правила відображення в клієнтському інтерфейсі. Таким чином, діаграма формалізує логічний каркас БД і підтверджує цілісність структури даних, необхідної для реалізації бізнес-функцій системи [21].

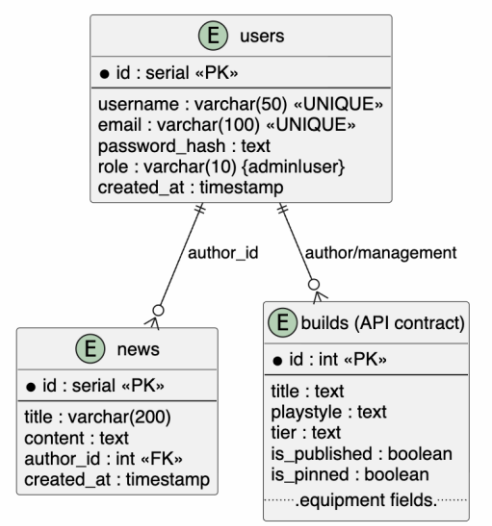


Рис. 2.5 Логічна модель даних (ядро)

На рисунку 2.6 показано структуру взаємодії компонентів у клієнт-серверній архітектурі застосунку. Діаграма відображає повний цикл обробки запиту: від ініціації дії користувачем у React-інтерфейсі до виконання SQL-операцій у PostgreSQL та повернення результату назад у UI. Окремо акцентовано роль проміжного шару Auth Middleware, який здійснює перевірку JWT і прав доступу перед виконанням захищених операцій. Наявність шару AuthContext + axios демонструє централізоване керування сесією користувача й автоматичне додавання токена до API-запитів. Отже, Рис. ілюструє узгоджену роботу всіх рівнів системи та підтверджує, що взаємодія компонентів побудована за принципами модульності, безпеки і керованості потоком даних.

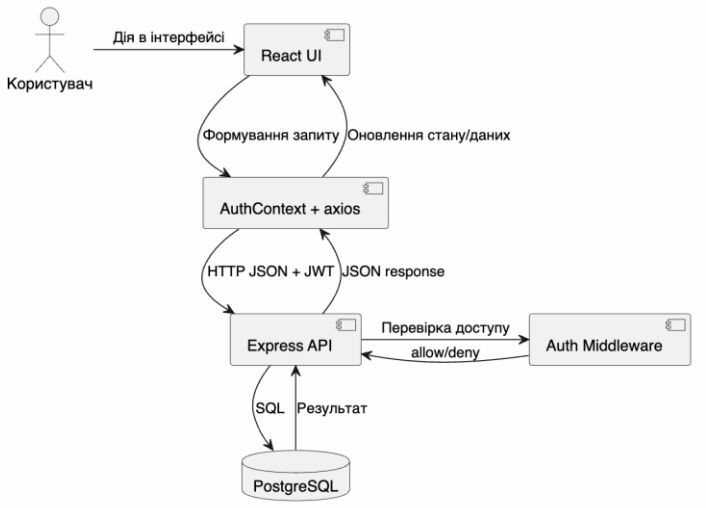


Рис. 2.6 Структура взаємодії компонентів

Висновок до розділу 2

У другому розділі сформовано теоретико-проєктну основу вебсистеми Albion Portal і визначено ключові принципи її побудови. Було обґрунтовано вибір клієнт-серверного підходу, описано логіку поділу системи на фронтенд, бекенд і рівень зберігання даних, а також визначено технологічний стек для реалізації кожного рівня. Окрема увага приділена структурі компонентів і ролі кожного модуля в загальній архітектурі застосунку.

У межах розділу виконано проєктування взаємодії між компонентами, моделі авторизації та механізмів контролю доступу на основі ролей. Подано структурні та поведінкові схеми, що відображають рух даних між інтерфейсом, API та базою даних, а також порядок обробки запитів користувача. Такий опис дозволив формалізувати роботу системи ще до етапу повної реалізації та зменшити ризики помилок у подальшій розробці.

Також було змодельовано структуру бази даних із виділенням основних сутностей, їх атрибутів і зв’язків, необхідних для підтримки функціоналу порталу. Проєктні рішення, зафіксовані в другому розділі, забезпечили цілісність майбутньої реалізації, сумісність між окремими модулями та можливість подальшого масштабування системи відповідно до вимог користувачів.

3 ПРОГРАМНА РЕАЛІЗАЦІЯ ВЕБ-ПЛАТФОРМИ ALBION PORTAL

3.1 Реалізація серверної частини та налаштування API

Серверну частину системи Albion Portal реалізовано на платформі Node.js із використанням фреймворку Express. Такий підхід дозволив побудувати легку модульну архітектуру API, у якій кожен функціональний блок винесено в окремий маршрутний модуль. Точка входу сервера виконує базову ініціалізацію: підключення CORS, парсинг JSON-запитів і реєстрацію маршрутів для роботи з автентифікацією, новинами, білдами та адміністративними операціями.

Налаштування API виконано за REST-принципом. Публічні ендпоінти забезпечують читання контенту, тоді як операції створення, редагування та видалення доступні лише авторизованим користувачам з роллю адміністратора. Для цього в сервері застосовано middleware-рівень перевірки токена та ролі (requireAuth, requireAdmin). У результаті на рівні API забезпечено розмежування доступу між гостем, авторизованим користувачем і адміністратором [22, 23].

Модуль автентифікації включає реєстрацію і вхід користувача з перевіркою вхідних даних, хешуванням пароля через bcryptjs та генерацією JWT. Після успішного логіну клієнт отримує токен, який надалі передається в заголовку Authorization. Така схема зберігає безстанність (stateless) серверної сесії і спрощує масштабування.

Важливою частиною реалізації стала інтеграція з PostgreSQL через драйвер pg та централізований модуль підключення до БД. Обробка помилок у маршрутах уніфікована: сервер повертає релевантні HTTP-коди (400, 401, 403, 404, 409, 500) та структуровані JSON-відповіді. Це підвищує передбачуваність API для клієнта і спрощує налагодження системи під час тестування.

index.js M

backend > src > index.js > ...

```
const express = require("express");
const cors = require("cors");
const app = express();

const buildsRoutes = require("./routes/builds");
const newsRoutes = require("./routes/news");
const authRoutes = require("./routes/auth");
const adminRoutes = require("./routes/admin");

app.use(
  cors({
    origin: ["http://localhost:5173", "http://localhost:5174"],
    methods: ["GET", "POST", "PUT", "DELETE", "PATCH", "OPTIONS"],
    allowedHeaders: ["Content-Type", "Authorization"],
  })
);

app.use(express.json());

app.use("/builds", buildsRoutes);
app.use("/news", newsRoutes);
app.use("/auth", authRoutes);
app.use("/admin", adminRoutes);

const PORT = 3001;
app.listen(PORT, () => {
  console.log(`Backend running on http://localhost:${PORT}`);
});
```

Рис. 3.1 Ініціалізація Express-сервера та підключення маршрутів API

auth.js U

backend > src > middleware > auth.js > ...

```
const jwt = require("jsonwebtoken");

const JWT_SECRET = process.env.JWT_SECRET || "albion_secret_key_change_in_prod";

// Перевіряє JWT токен, додає req.user = { id, username, role }
function requireAuth(req, res, next) {
  const header = req.headers["authorization"];
  if (!header || !header.startsWith("Bearer ")) {
    return res.status(401).json({ error: "Необхідна авторизація" });
  }

  const token = header.slice(7);
  try {
    const payload = jwt.verify(token, JWT_SECRET);
    req.user = payload;
    next();
  } catch {
    return res.status(401).json({ error: "Недійсний або прострочений токен" });
  }
}

// Перевіряє що роль = 'admin' (використовується після requireAuth)
function requireAdmin(req, res, next) {
  if (req.user?.role !== "admin") {
    return res.status(403).json({ error: "Доступ заборонено: потрібна роль admin" });
  }
  next();
}

module.exports = { requireAuth, requireAdmin, JWT_SECRET };
```

Рис. 3.2 Реалізація middleware авторизації та перевірки ролей

```
auth.js U
backend > src > middleware > auth.js > ...
const jwt = require("jsonwebtoken");

const JWT_SECRET = process.env.JWT_SECRET || "albion_secret_key_change_in_prod";

// Перевіряє JWT токен, додає req.user = { id, username, role }
function requireAuth(req, res, next) {
  const header = req.headers["authorization"];
  if (!header || !header.startsWith("Bearer ")) {
    return res.status(401).json({ error: "Необхідна авторизація" });
  }

  const token = header.slice(7);
  try {
    const payload = jwt.verify(token, JWT_SECRET);
    req.user = payload;
    next();
  } catch {
    return res.status(401).json({ error: "Недійсний або прострочений токен" });
  }
}

// Перевіряє що роль = 'admin' (використовується після requireAuth)
function requireAdmin(req, res, next) {
  if (req.user?.role !== "admin") {
    return res.status(403).json({ error: "Доступ заборонено: потрібна роль admin" });
  }
  next();
}

module.exports = { requireAuth, requireAdmin, JWT_SECRET };
```

Рис. 3.3 Реєстрація та вхід користувача (JWT + bcrypt) у модулі auth

```
news.js U
backend > src > routes > news.js > ...
const express = require("express");
const router = express.Router();
const db = require("../db");
const { requireAuth, requireAdmin } = require("../middleware/auth");

// GET /news — публічний (тільки опубліковані, закріплені зверху)
router.get("/", async (req, res) => {
  try {
    const result = await db.query(`
      SELECT n.id, n.title, n.content, n.created_at, n.updated_at,
             n.is_published, n.is_pinned,
             u.username as author
      FROM albion.news n
      LEFT JOIN albion.users u ON n.author_id = u.id
      WHERE n.is_published = true
      ORDER BY n.is_pinned DESC, n.id DESC
    `);
    res.json(result.rows);
  } catch (err) {
    console.error("Error getting news:", err);
    res.status(500).json({ error: "Помилка сервера" });
  }
});

// GET /news/all — для адміна (всі новини)
router.get("/all", requireAuth, requireAdmin, async (req, res) => {
  try {
    const result = await db.query(`
      SELECT n.id, n.title, n.content, n.created_at, n.updated_at,
             n.is_published, n.is_pinned,
```

Рис. 3.4 Реалізація API для керування новинами (публічні та admin-ендпоінти)

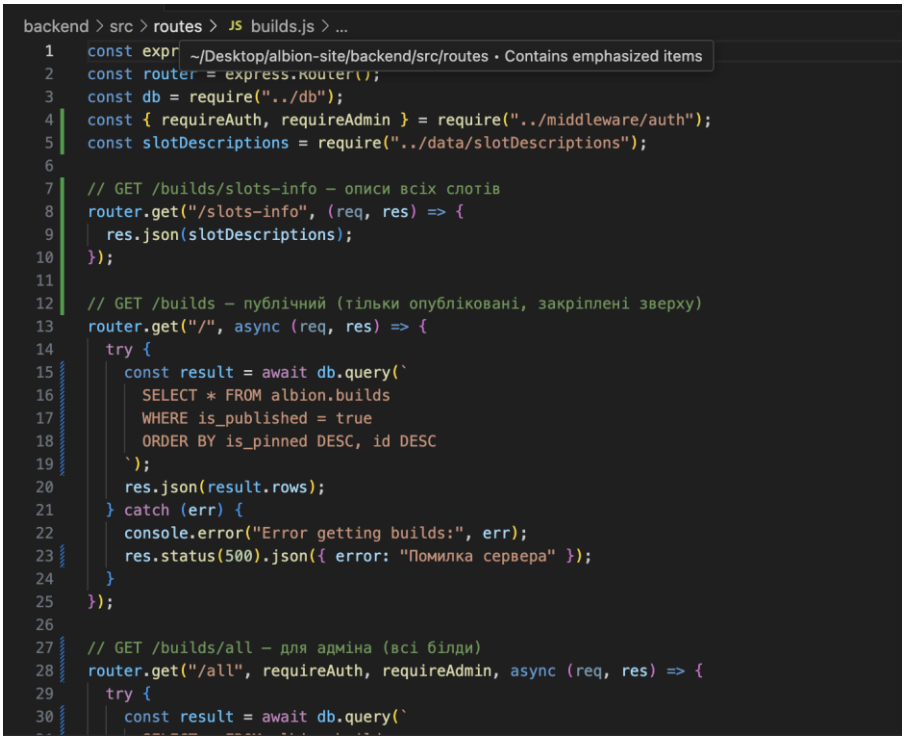


Рис. 3.5 Реалізація API для керування білдами (CRUD, publish, pin)

Рис. 3.6 Адміністративні ендпоінти керування користувачами і статистикою

Рис. 3.7 Підключення серверної частини до PostgreSQL

Серверна частина отримала чітку модульну структуру, захист маршрутів через JWT і рольову модель доступу, а також надійний механізм взаємодії з PostgreSQL. Практична цінність виконаної роботи полягає в тому, що створений API уже придатний для повноцінної інтеграції з клієнтською частиною, подальшого тестування та розвитку функціоналу без зміни базової архітектури [24, 25].

3.2 Розробка клієнтської частини з використанням React та TailwindCSS

Клієнтську частину Albion Portal реалізовано як односторінковий застосунок (SPA) на базі React, що забезпечує швидку навігацію між сторінками без повного перезавантаження браузера. Для маршрутизації використано react-router-dom, завдяки чому окремо організовано сторінки перегляду білдів, новин, авторизації, реєстрації та адміністративну панель. Такий підхід підвищує зручність користування системою та спрощує подальше масштабування інтерфейсу.

Візуальну частину побудовано за допомогою TailwindCSS, що дало змогу сформувати єдиний стиль компонентів, адаптивну структуру сторінок і швидко реалізувати сучасний інтерфейс без надлишкового CSS-коду. Компонентний підхід

React дозволив винести повторювані елементи (навігація, форми, картки контенту) в окремі модулі, що покращує читабельність коду та пришвидшує внесення змін. Управління станом автентифікації реалізовано через AuthContext: після входу токен зберігається в localStorage, декодується на клієнті, а роль користувача впливає на доступні елементи інтерфейсу [26].

Для взаємодії з сервером застосовано централізований axios-клієнт із базовою адресою API та автоматичним додаванням JWT у заголовок Authorization. Це дозволяє уніфікувати роботу з HTTP-запитами та забезпечити коректну роботу захищених маршрутів. Практичний результат підрозділу полягає у створенні повноцінного клієнтського інтерфейсу, готового до демонстрації основних сценаріїв: реєстрації, входу, перегляду новин і білдів, а також адміністративного керування контентом.

App.jsx M

frontend > src > App.jsx > ...

```jsx
import { BrowserRouter, Routes, Route, Navigate } from "react-router-dom";
import { AuthProvider } from "./context/AuthContext";
import Navbar from "./components/Navbar";
import BuildsPage from "./pages/BuildsPage";
import NewsPage from "./pages/NewsPage";
import LoginPage from "./pages/LoginPage";
import RegisterPage from "./pages/RegisterPage";
import AdminPage from "./pages/AdminPage";

export default function App() {
  return (
    <AuthProvider>
      <BrowserRouter>
        <div className="min-h-screen bg-zinc-950 text-white">
          <Navbar />
          {/* Відступ під фіксований navbar */}
          <div className="pt-20">
            <Routes>
              <Route path="/" element={<Navigate to="/builds" replace />} />
              <Route path="/builds" element={<BuildsPage />} />
              <Route path="/news" element={<NewsPage />} />
              <Route path="/login" element={<LoginPage />} />
              <Route path="/register" element={<RegisterPage />} />
              <Route path="/admin" element={<AdminPage />} />
            </Routes>
          </div>
        </div>
      </BrowserRouter>
    </AuthProvider>
  );
```

Рис. 3.8 Маршрутизація клієнтського застосунку React

Рис. 3.9 Керування станом автентифікації в AuthContext

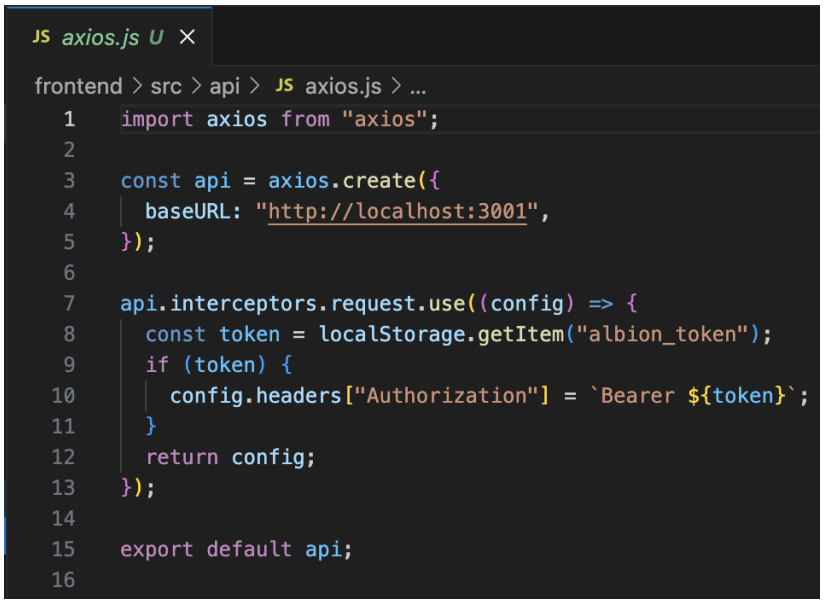


Рис. 3.10 Налаштування axios-клієнта для взаємодії з API

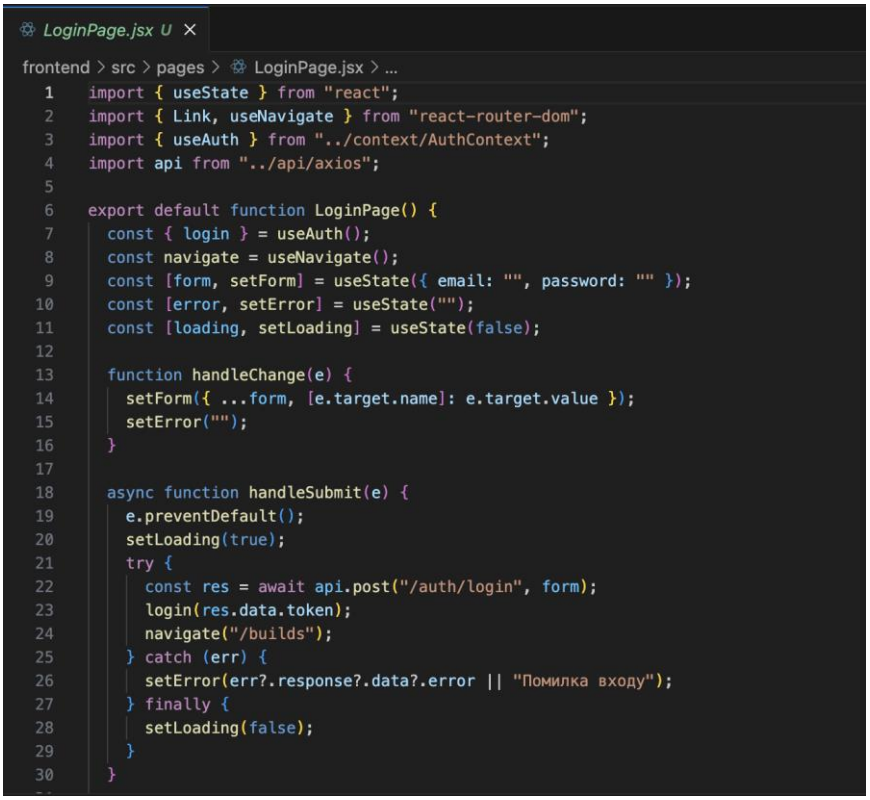


Рис. 3.11 Реалізація сторінки входу користувача

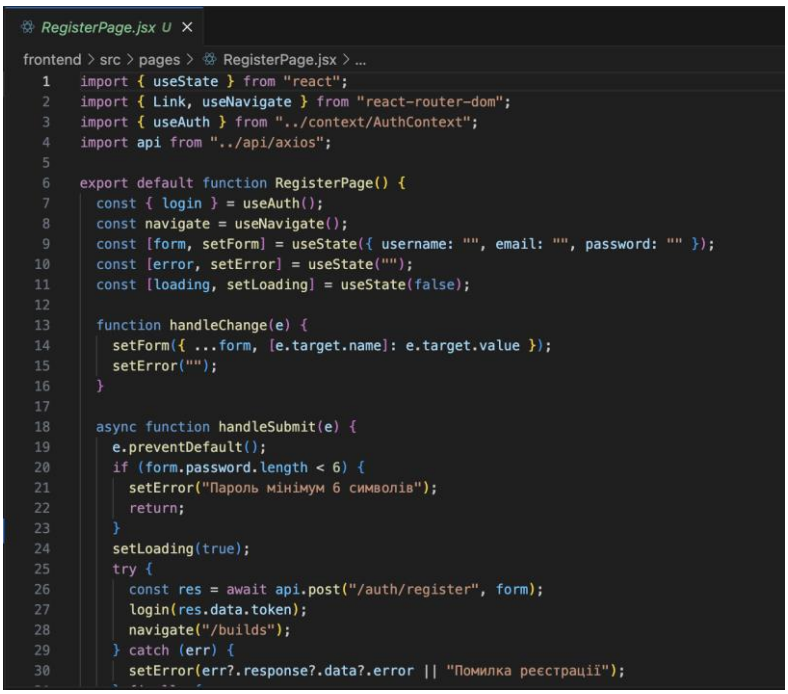


Рис. 3.12 Реалізація сторінки реєстрації користувача

AdminPage.jsx U

frontend > src > pages > AdminPage.jsx > ...

```jsx
import { useEffect, useState } from 'react';
import api from '../api/axios';

export default function AdminPage() {
  const [users, setUsers] = useState([]);
  const [stats, setStats] = useState(null);
  const [loading, setLoading] = useState(true);
  const [error, setError] = useState('');

  useEffect(() => {
    fetchData();
  }, []);

  async function fetchData() {
    setLoading(true);
    try {
      const [usersRes, statsRes] = await Promise.all([
        api.get('/admin/users'),
        api.get('/admin/stats')
      ]);
      setUsers(usersRes.data);
      setStats(statsRes.data);
    } catch (err) {
      setError('Помилка завантаження даних');
      console.error(err);
    } finally {
      setLoading(false);
    }
  }

```

Рис. 3.13 Адміністративна сторінка керування системою

На цьому етапі демонструється інтерфейс сторінки входу користувача в системі Albion Portal. Форма авторизації містить поля для введення email і пароля, а також кнопку переходу до процедури реєстрації нового облікового запису [27].

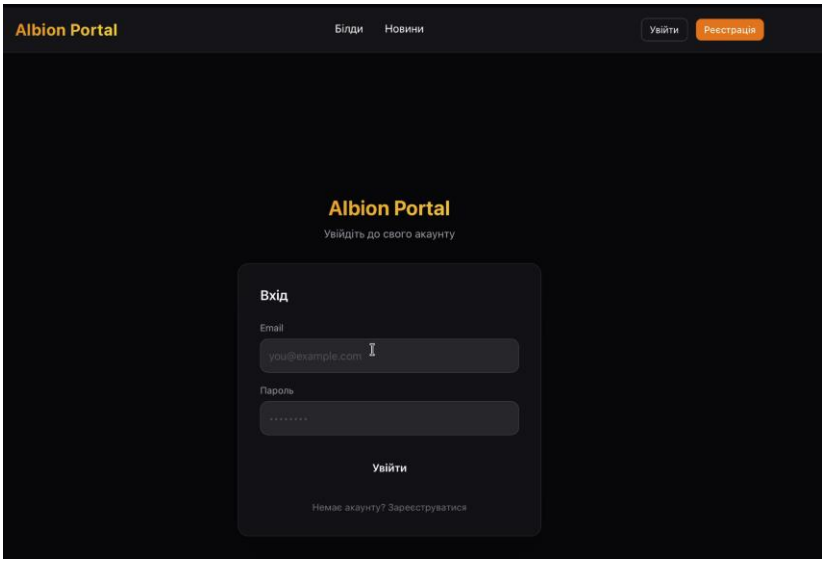


Рис. 3.14 Сторінка входу користувача в клієнтському застосунку Albion Portal

На рисунку показано реалізацію екрана автентифікації, через який користувач виконує вхід до системи та отримує доступ до персоналізованого функціоналу відповідно до своєї ролі.

Далі демонструється сторінка керування білдами для користувача з роллю адміністратора. На екрані показано форму створення нового білду з вибором tier, стилю гри, вартості та детальним заповненням слотів спорядження [28].

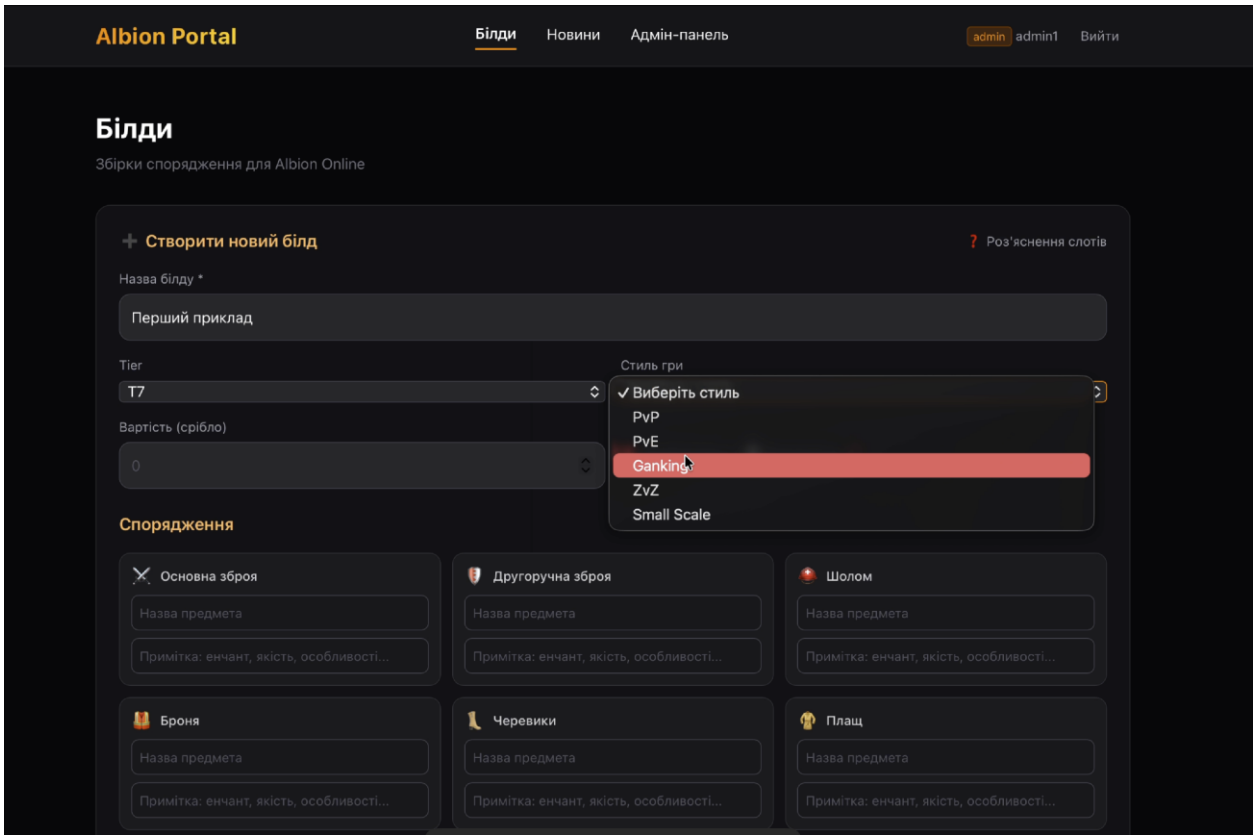


Рис. 3.15 Інтерфейс створення нового білду в адміністративному режимі

На рисунку відображено реалізацію адміністративного функціоналу додавання білдів, який дозволяє структуровано формувати ігрові збірки та публікувати їх у системі для подальшого перегляду користувачами.

Після збереження форми демонструється результат відображення створеного білду в загальному списку. На картці видно основні метадані (tier, стиль гри, вартість) і структурований набір спорядження за слотами [26-28].

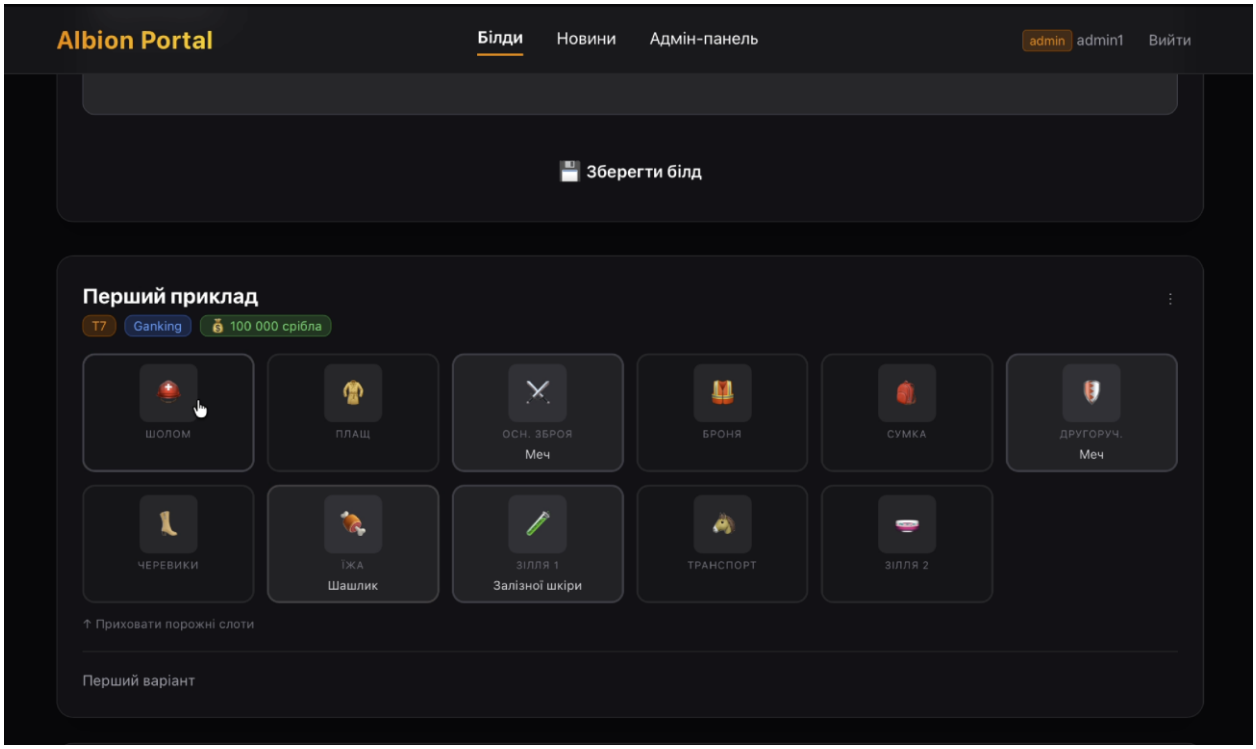


Рис. 3.16 Відображення створеного білду в інтерфейсі клієнтського застосунку

На рисунку продемонстровано, що після створення запису дані коректно відображаються у вигляді картки білду, що підтверджує працездатність зв’язки «форма введення → API → база даних → оновлення UI». Наступним кроком демонструється сторінка новин в адміністративному режимі, де доступна форма публікації нового інформаційного запису із заголовком і текстом. Такий сценарій ілюструє повний життєвий цикл контенту: від моменту введення даних адміністратором до їх збереження на сервері та подальшої візуалізації в інтерфейсі без додаткових ручних дій. Це свідчить про коректну синхронізацію між клієнтською та серверною частинами системи, а також про узгодженість структури даних на всіх етапах обробки.

Додатково результати демонстрації підтверджують, що механізм валідації форми та передавання параметрів запиту працює стабільно: запис публікується лише після успішного проходження перевірок і повернення позитивної відповіді від API. Після цього інтерфейс оновлюється, а новина стає доступною для перегляду в адміністративному списку, що підвищує зручність модерації та оперативність керування інформаційним вмістом.

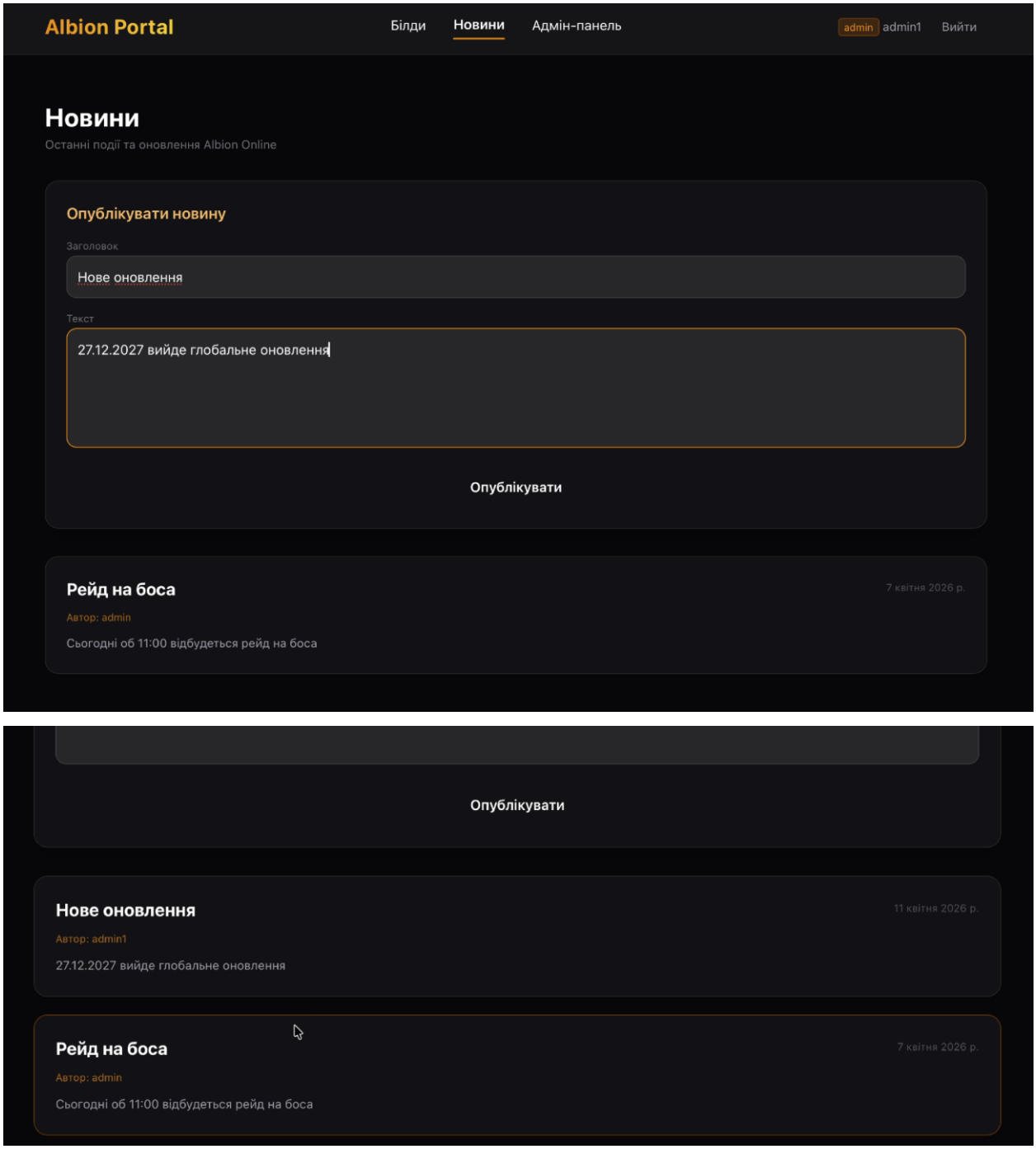


Рис. 3.17 Інтерфейс створення та публікації новини в системі Albion Portal

На рисунку показано реалізацію модуля новин: адміністратор може додавати публікації, після чого новина відразу відображається у стрічці з автором і датою створення.

Додатково демонструється інформаційне модальне вікно з поясненням слотів спорядження, яке допомагає користувачеві коректно формувати білд і розуміти призначення кожного елемента екіпірування [27].

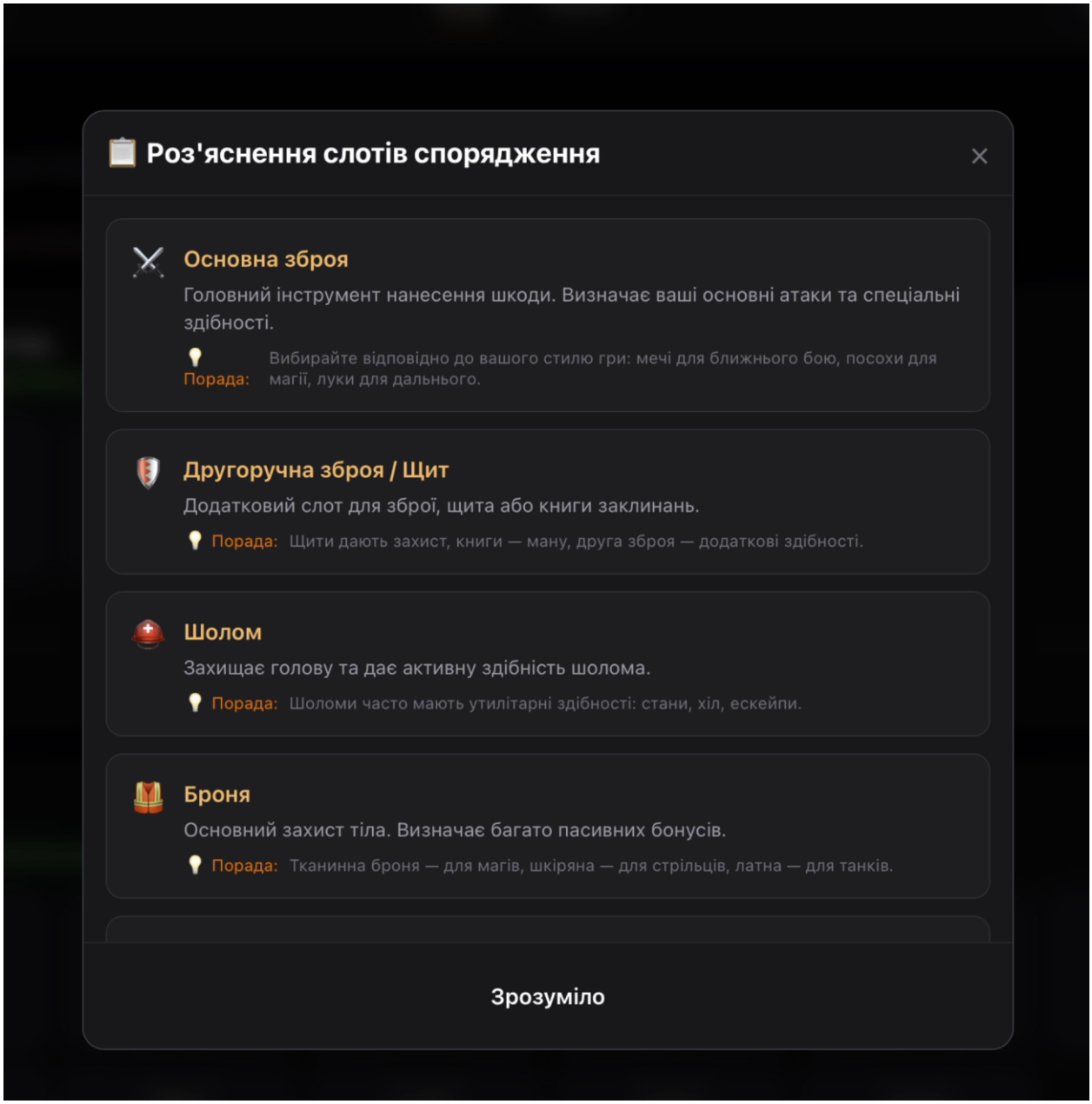


Рис. 3.18 Модальне вікно довідки щодо слотів спорядження

На рисунку показано реалізацію вбудованої довідкової підсистеми інтерфейсу, що підвищує зрозумілість роботи з формою створення білдів і покращує користувацький досвід під час заповнення даних.

3.3 Реалізація системи автентифікації, ролей та взаємодії з базою даних PostgreSQL

У підсистемі безпеки Albion Portal реалізовано JWT-автентифікацію, рольову модель доступу (user/admin) та пряму інтеграцію серверної частини з PostgreSQL через pg. Підхід побудовано так, щоб клієнт працював із токеном, сервер перевіряв права доступу на кожному захищеному маршруті, а база даних зберігала лише хеші паролів і ролі користувачів [29, 30].

Автентифікація користувачів (реєстрація/вхід)

Фрагмент реєстрації (bcrypt + запис у БД):

```
// backend/src/routes/auth.js
const password_hash = await bcrypt.hash(password, 10);
const result = await db.query(
  "INSERT INTO albion.users (username, email, password_hash, role) VALUES ($1, $2, $3, 'user') RETURNING id, username, role",
  [username, email, password_hash]
);
```

Фрагмент входу (перевірка пароля + JWT):

```
// backend/src/routes/auth.js
const valid = await bcrypt.compare(password, user.password_hash);
if (!valid) {
  return res.status(401).json({ error: "Невірний email або пароль" });
}

const token = jwt.sign(
  { id: user.id, username: user.username, role: user.role },
  JWT_SECRET,
  { expiresIn: "7d" }
);
```

Фрагмент збереження токена на клієнті:

```
// frontend/src/context/AuthContext.jsx
function login(token) {
  localStorage.setItem("albion_token", token);
  setUser(decodeToken(token));
}
```

Рольова модель доступу (RBAC)

Middleware перевірки авторизації та ролі:

```
// backend/src/middleware/auth.js
function requireAuth(req, res, next) {
  const header = req.headers["authorization"];
  if (!header || !header.startsWith("Bearer ")) {
    return res.status(401).json({ error: "Необхідна авторизація" });
  }

  const token = header.slice(7);
  const payload = jwt.verify(token, JWT_SECRET);
  req.user = payload;
  next();
}

function requireAdmin(req, res, next) {
  if (req.user?.role !== "admin") {
    return res.status(403).json({ error: "Доступ заборонено: потрібна роль admin"
});
  }
  next();
}
```

Застосування до admin-маршрутів:

```
// backend/src/routes/admin.js
router.use(requireAuth, requireAdmin);
```

Взаємодія з PostgreSQL

Підключення до БД:

```
// backend/src/db.js
const pool = new Pool({
  user: "kilim",
  host: "localhost",
  database: "albion",
  password: "",
  port: 5432,
});
```

Модель користувача в БД (роль + хеш):

```
-- backend/src/db/migrate.sql
CREATE TABLE IF NOT EXISTS albion.users (
  id SERIAL PRIMARY KEY,
  username VARCHAR(50) UNIQUE NOT NULL,
  email VARCHAR(100) UNIQUE NOT NULL,
  password_hash TEXT NOT NULL,
  role VARCHAR(10) NOT NULL DEFAULT 'user' CHECK (role IN ('admin', 'user')),
  created_at TIMESTAMP DEFAULT NOW()
);
```

Клієнт додає JWT у кожен захищений запит:

```
// frontend/src/api/axios.js
api.interceptors.request.use((config) => {
  const token = localStorage.getItem("albion_token");
  if (token) {
    config.headers["Authorization"] = `Bearer ${token}`;
  }
  return config;
});
```

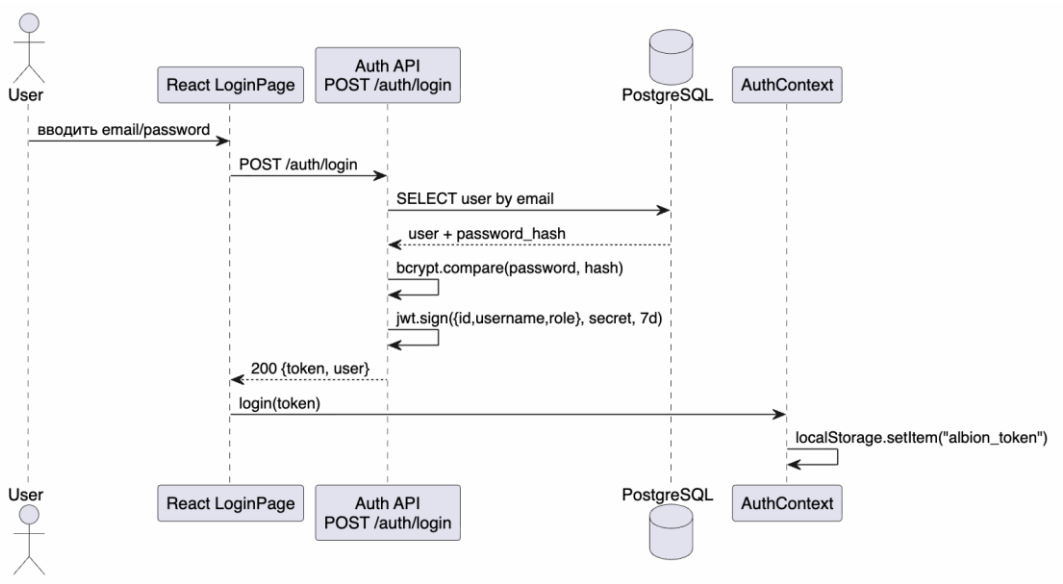


Рис. 3.19 Діаграма послідовності автентифікації користувача

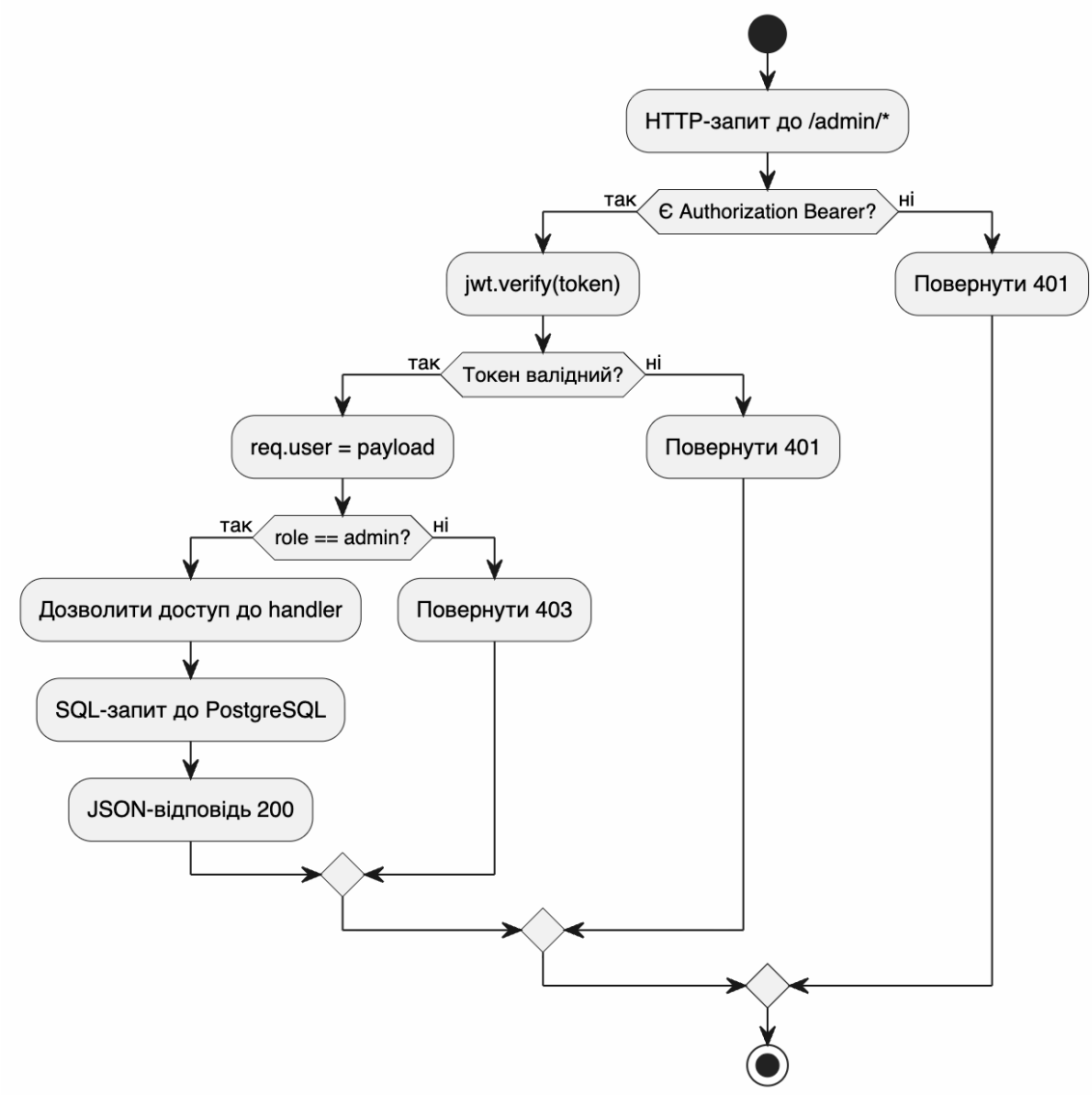


Рис. 3.20 Контроль доступу до захищеного admin-ендпоінта

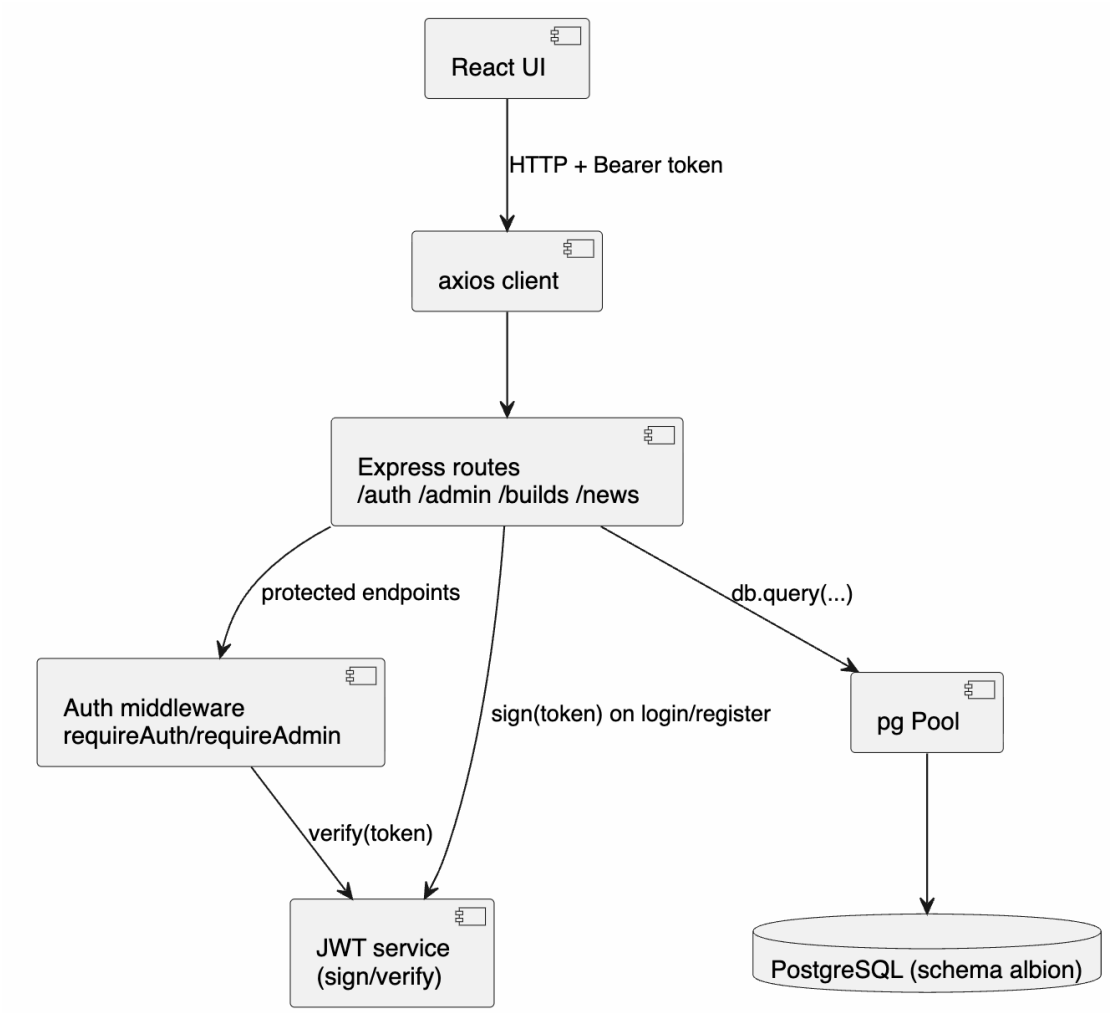


Рис. 3.21 Компонентна схема безпеки і доступу до БД

Висновок до розділу 3

У третьому розділі виконано практичну реалізацію вебсистеми Albion Portal на основі клієнт-серверної архітектури. Було розгорнуто серверну частину на Node.js/Express, налаштовано API-маршрути для автентифікації, роботи з новинами, білдами та адміністративними функціями. На клієнтському рівні реалізовано React-інтерфейс із маршрутизацією, формами входу/реєстрації та сторінками керування контентом.

У процесі розробки впроваджено механізм JWT-автентифікації, рольову модель доступу user/admin та взаємодію з PostgreSQL через централізований модуль підключення. Перевірено ключові сценарії: створення облікового запису, вхід у систему, публікація новин, створення білдів, відображення даних у

клієнтському інтерфейсі, а також обмеження доступу до адміністративних операцій. Отримані результати підтвердили коректну роботу логіки безпеки і стабільність обміну даними між фронтендом, бекендом та базою даних.

Практичним підсумком розділу є працездатний програмний продукт, придатний для демонстрації та подальшого розвитку. Реалізована структура коду залишається модульною і зручною для масштабування, а обрані технології забезпечують достатню гнучкість для додавання нових функцій, покращення UX та розширення бізнес-логіки в наступних етапах роботи.

ВИСНОВКИ

У кваліфікаційній роботі проаналізовано теоретичні основи створення веб-платформ для ігрових спільнот, зокрема специфіку онлайн-ігор, вимоги до інтерактивності, зручності інтерфейсу та швидкості обміну даними. У межах першого розділу розглянуто сучасні frontend- і backend-технології та виконано порівняння інструментів, що дало змогу обґрунтувати вибір технологічного стеку для реалізації платформи.

У другому розділі виокремлено функціональні вимоги до веб-платформи для гравців Albion Online, спроєктовано архітектуру клієнт-серверної системи та модель взаємодії компонентів. Також виконано моделювання структури бази даних із визначенням ключових сутностей, зв’язків і правил доступу, необхідних для стабільної роботи системи.

У третьому розділі розроблено програмну реалізацію веб-платформи Albion Portal: серверну частину на Node.js/Express, клієнтську частину на React із використанням TailwindCSS, REST API, механізми автентифікації та авторизації на основі JWT, а також інтеграцію з PostgreSQL. Для адміністративного рівня реалізовано керування новинами, білдами та користувачами відповідно до рольової моделі доступу.

У результаті роботи продемонстровано працездатність створеного програмного рішення на практичних сценаріях: реєстрація та вхід користувачів, публікація новин, створення білдів, відображення контенту та обмеження доступу за ролями. Поставлену мету досягнуто, а отриманий результат має практичну цінність і може бути використаний як основа для подальшого розвитку платформи.

ПЕРЕЛІК ПОСИЛАНЬ

1. Fielding R. Architectural Styles and the Design of Network-based Software Architectures. – Irvine: University of California, 2000. – 162 p.
2. Flanagan D. JavaScript: The Definitive Guide. – 7th ed. – Sebastopol: O'Reilly Media, 2020. – 704 p.
3. Simpson K. You Don't Know JS Yet: Get Started. – 2nd ed. – O'Reilly Media, 2020. – 143 p.
4. Marijn Haverbeke. Eloquent JavaScript. – 3rd ed. – No Starch Press, 2018. – 472 p.
5. Banks A., Porcello E. Learning React. – 2nd ed. – O'Reilly Media, 2020. – 350 p.
6. Accomazzo A., Murray N., Lerner A. Fullstack React. – New York: Fullstack.io, 2017. – 826 p.
7. Osmani A. Learning JavaScript Design Patterns. – O'Reilly Media, 2012. – 254 p.
8. Casciaro M., Mammino L. Node.js Design Patterns. – 3rd ed. – Packt Publishing, 2020. – 526 p.
9. Tilkov S., Vinoski S. Node.js: Using JavaScript to Build High-Performance Network Programs // IEEE Internet Computing. – 2010. – Vol. 14, No. 6. – P. 80–83.
10. Richardson L., Amundsen M., Ruby S. RESTful Web APIs. – O'Reilly Media, 2013. – 408 p.
11. Rob Conery. The Imposter's Handbook. – Big Machine, 2016. – 414 p.
12. Silberschatz A., Korth H., Sudarshan S. Database System Concepts. – 7th ed. – McGraw-Hill, 2019. – 1376 p.
13. Stonebraker M., Hellerstein J. Readings in Database Systems. – MIT Press, 2015. – 432 p.
14. PostgreSQL Global Development Group. PostgreSQL Documentation [Електронний ресурс]. – Режим доступу: https://www.postgresql.org/docs/ (дата звернення: 18.04.2026).

15. Meta Platforms. React Documentation [Електронний ресурс]. – Режим доступу: https://react.dev (дата звернення: 18.04.2026).

16. OpenJS Foundation. Node.js Documentation [Електронний ресурс]. – Режим доступу: https://nodejs.org/en/docs (дата звернення: 18.04.2026).

17. Tailwind Labs. TailwindCSS Documentation [Електронний ресурс]. – Режим доступу: https://tailwindcss.com/docs (дата звернення: 18.04.2026).

18. React Team. React Documentation [Електронний ресурс]. Режим доступу: https://react.dev/ (дата звернення: 17.04.2026).

19. Remix Software. React Router Documentation [Електронний ресурс]. Режим доступу: https://reactrouter.com/ (дата звернення: 17.04.2026).

20. Tailwind Labs. Tailwind CSS Documentation [Електронний ресурс]. Режим доступу: https://tailwindcss.com/docs (дата звернення: 17.04.2026).

21. Vite Team. Vite Guide [Електронний ресурс]. Режим доступу: https://vite.dev/guide/ (дата звернення: 17.04.2026).

22. Axios Authors. Axios Documentation [Електронний ресурс]. Режим доступу: https://axios-http.com/docs/intro (дата звернення: 17.04.2026).

23. OpenJS Foundation. Node.js Documentation [Електронний ресурс]. Режим доступу: https://nodejs.org/docs/latest/api/ (дата звернення: 17.04.2026).

24. Express.js Team. Express 5.x API Reference [Електронний ресурс]. Режим доступу: https://expressjs.com/en/5x/api.html (дата звернення: 17.04.2026).

25. Express.js Team. Using middleware in Express [Електронний ресурс]. Режим доступу: https://expressjs.com/en/guide/using-middleware.html (дата звернення: 17.04.2026).

26. PostgreSQL Global Development Group. PostgreSQL Documentation [Електронний ресурс]. Режим доступу: https://www.postgresql.org/docs/current/ (дата звернення: 17.04.2026).

27. PostgreSQL Global Development Group. CREATE TABLE [Електронний ресурс]. Режим доступу: https://www.postgresql.org/docs/current/sql-createtable.html (дата звернення: 17.04.2026).

28. node-postgres contributors. node-postgres (pg) Documentation

[Електронний ресурс]. Режим доступу: https://node-postgres.com/ (дата звернення: 17.04.2026).

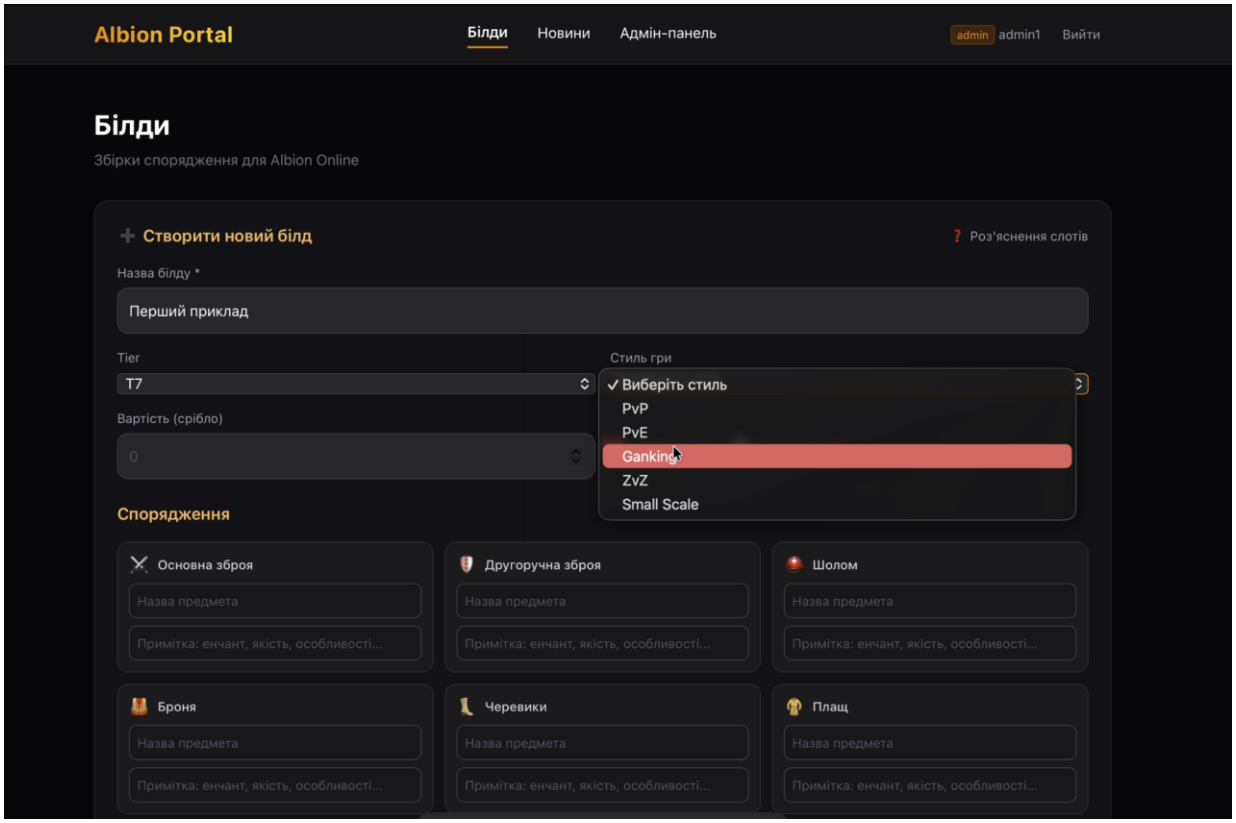
29. Jones M., Bradley J., Sakimura N. RFC 7519: JSON Web Token (JWT) [Електронний ресурс]. Режим доступу: https://datatracker.ietf.org/doc/html/rfc7519 (дата звернення: 17.04.2026).

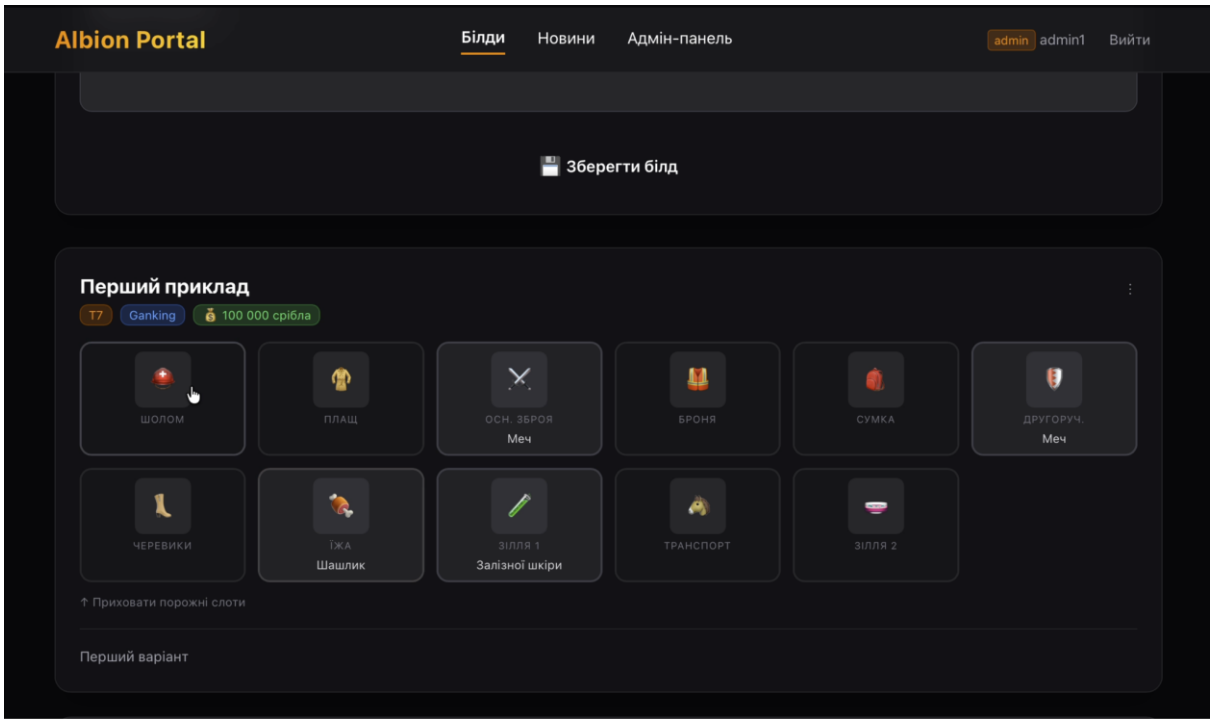
30. OWASP Foundation. Authentication Cheat Sheet [Електронний ресурс]. Режим доступу: https://cheatsheetseries.owasp.org/cheatsheets/Authentication_Cheat_Sheet.html (дата звернення: 17.04.2026).

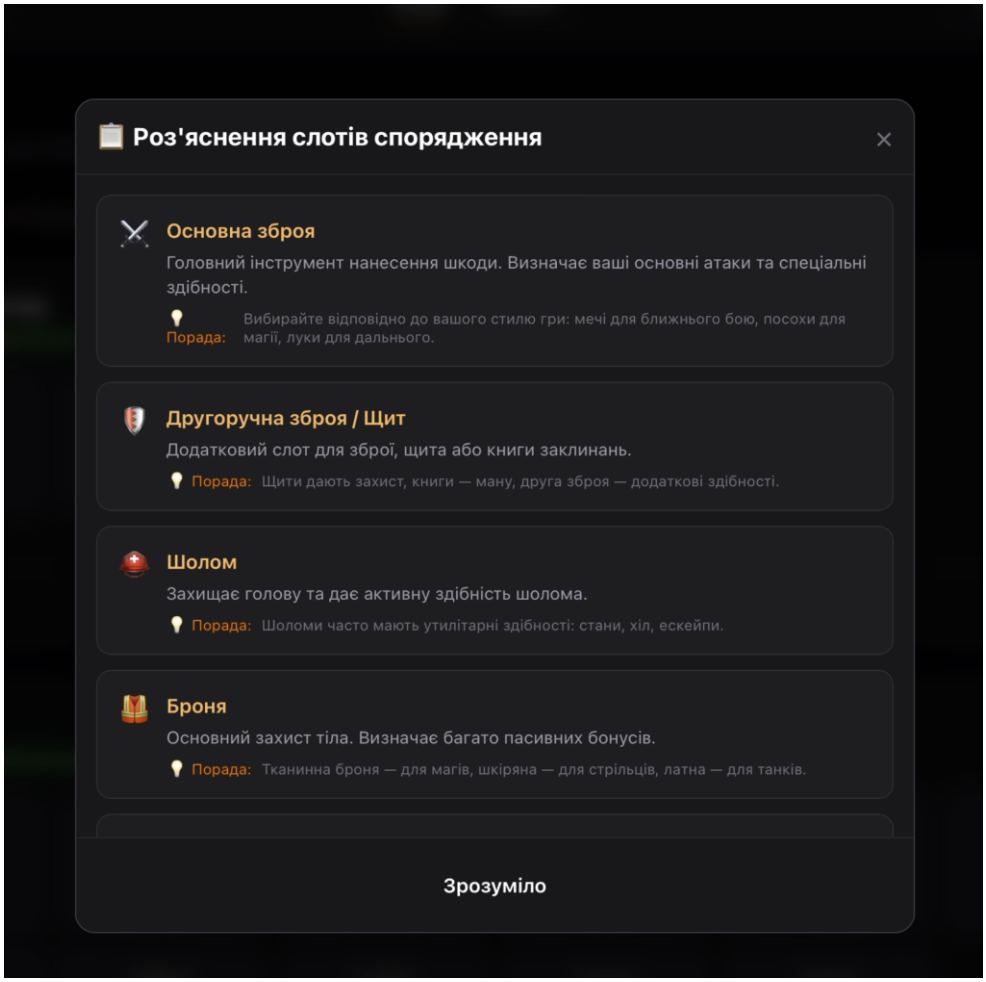
31. OWASP Foundation. JSON Web Token for Java Cheat Sheet [Електронний ресурс]. Режим доступу: https://cheatsheetseries.owasp.org/cheatsheets/JSON_Web_Token_for_Java_Cheat_Sheet.html (дата звернення: 17.04.2026).

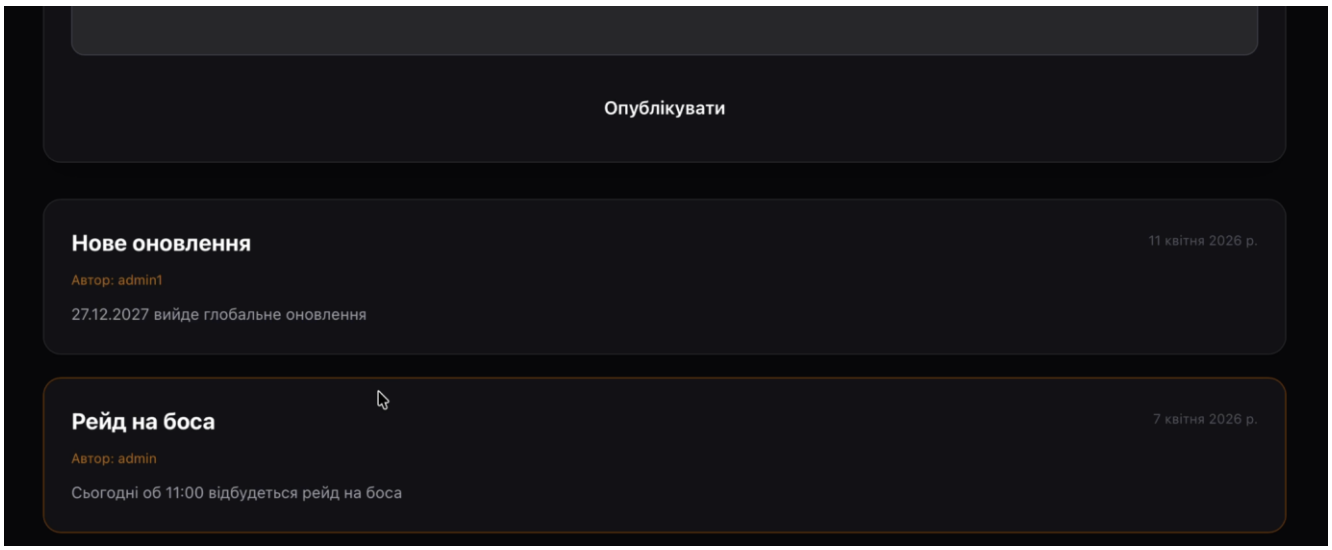
32. IETF. RFC 9110: HTTP Semantics [Електронний ресурс]. Режим доступу: https://datatracker.ietf.org/doc/html/rfc9110 (дата звернення: 17.04.2026).

ДЕМОНСТРАЦІЙНІ МАТЕРІАЛИ






Роз'яснення слотів спорядження
Основна зброя
Головний інструмент нанесення шкоди. Визначає ваші основні атаки та спеціальні здібності.
Порада: Вибирайте відповідно до вашого стилю гри: мечі для ближнього бою, посохи для магії, луки для дальнього.
Другоручна зброя / Щит
Додатковий слот для зброї, щита або книги заклинань.
Порада: Щити дають захист, книги — ману, друга зброя — додаткові здібності.
Шолом
Захищає голову та дає активну здібність шолома.
Порада: Шоломи часто мають утилітарні здібності: стани, хіл, ескейпи.
Броня
Основний захист тіла. Визначає багато пасивних бонусів.
Порада: Тканинна броня — для магів, шкіряна — для стрільців, латна — для танків.
Зрозуміло


Опублікувати
Нове оновлення
11 квітня 2026 р.
Автор: admin1
27.12.2027 вийде глобальне оновлення
Рейд на боса
7 квітня 2026 р.
Автор: admin
Сьогодні об 11:00 відбудеться рейд на боса