МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

ДЕРЖАВНИЙ УНІВЕРСИТЕТ "ЖИТОМИРСЬКА ПОЛІТЕХНІКА"

Кафедра інженерії програмного забезпечення

# КУРСОВА РОБОТА

з дисципліни: «Моделювання та аналіз програмного забезпечення»

на тему:

**«Платформа шаблонного створення вебресурсів»**

студента 4 курсу групи ІПЗ-22-3
спеціальності 121 «Інженерія
програмного забезпечення»
Леус Вадим Олександрович
(прізвище, ім'я та по-батькові)

Керівник: В.о. завідувача кафедри КН
Левківський В.Л.
Дата захисту: " ___ " грудня 2025р.
Національна шкала _____
Кількість балів: _____
Оцінка: ECTS _____

Члени комісії _____ Сугоняк І.І.
(підпис) (прізвище та ініціали)
_____ Кравченко С.М.
(підпис) (прізвище та ініціали)
_____ Левківський В.Л.
(підпис) (прізвище та ініціали)

Житомир - 2025

ДЕРЖАВНИЙ УНІВЕРСИТЕТ «ЖИТОМИРСЬКА ПОЛІТЕХНІКА»
Факультет інформаційно-комп'ютерних технологій
Кафедра інженерії програмного забезпечення
Освітній рівень: бакалавр
Спеціальність 121 «Інженерія програмного забезпечення»

«ЗАТВЕРДЖУЮ»
Завідувач кафедри ІПЗ
_____ Тетяна Вакалюк
"__" _____ 2025 р.

ЗАВДАННЯ
НА КУРСОВУ РОБОТУ СТУДЕНТУ
Леусу Вадиму Олександровичу

1. Тема роботи: Платформа шаблонного створення вебресурсів
керівник роботи: В.о. завідувача кафедри КН Левківський В.Л.
2. Строк подання студентом: " __" _____2025 р.
3. Вихідні дані до роботи: Розробити архітектуру для платформи шаблонного створення вебресурсів
4. Зміст розрахунково-пояснювальної записки (перелік питань, які підлягають розробці):
   1. Постановка завдання
   2. Аналіз вимог користувача
   3. Модель програмного комплексу на логічному рівні
   4. Фізична модель
   5. Прототип програмного комплексу
5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень)
   1. Презентація
   2. Діаграми
6. Консультанти розділів проекту (роботи)

| Розділ | Прізвище, ініціали та посади консультанта | Підпис, дата | |
|---|---|---|---|
| | | завдання видав | завдання прийняв |
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |

7. Дата видачі завдання " __" _____2025 р.

## Календарний план

| № з/п | Назва етапів курсового проекту (роботи) | Строк виконання етапів роботи | Примітки |
|---|---|---|---|
| 1 | Постановка задачі | 25.10.2025 | Виконано |
| 2 | Аналіз вимог користувача | 27.10.2025 | Виконано |
| 3 | Формулювання технічного завдання | 31.10.2025 | Виконано |
| 4 | Опрацювання літературних джерел | 01.11.2025 | Виконано |
| 5 | Моделювання системи на логічному рівні | 04.11.2025 | Виконано |
| 6 | Фізична модель | 07.11.2025 | Виконано |
| 7 | Генерування програмного коду для прототипу | 13.11.2025 | Виконано |
| 8 | Написання пояснювальної записки | 24.11.2025 | Виконано |
| 9 | Захист | 09.12.2025 | Виконано |

**Студент** _____ Леус В.О.
(підпис) (прізвище та ініціали)

**Керівник роботи** _____ Левківський В.Л.
(підпис) (прізвище та ініціали)

**Зміст**


РЕФЕРАТ ..... **5**
ВСТУП ..... **6**
РОЗДІЛ 1. АНАЛІЗ ВИМОГ КОРИСТУВАЧА ТА КОНЦЕПТУАЛЬНЕ МОДЕЛЮВАННЯ ..... **8**
1.1. Технічне завдання на розробку системи ..... **8**
1.2. Обґрунтування вибору засобів моделювання ..... **11**
1.3. Аналіз вимог до програмного продукту ..... **13**
Висновки до першого розділу: ..... **15**
РОЗДІЛ 2. РОЗРОБКА МОДЕЛІ ПРОГРАМНОГО КОМПЛЕКСУ НА ЛОГІЧНОМУ РІВНІ ..... **16**
2.1. Алгоритм роботи та стани програмної системи ..... **16**
2.2. Об’єктно-орієнтована модель системи ..... **21**
2.3. Комунікації і послідовність взаємодії об’єктів системи ..... **25**
Висновки до другого розділу ..... **30**
РОЗДІЛ 3. ФІЗИЧНА МОДЕЛЬ ТА ПРОТОТИП ПРОГРАМНОГО КОМПЛЕКСУ ..... **30**
3.1. Взаємодія компонентів системи ..... **31**
3.2. Архітектура програмного комплексу та його розгортання ..... **32**
3.3. Генерування програмного коду для прототипу програмного комплексу **35**
Висновки до третього розділу ..... **35**
ВИСНОВКИ ..... **41**
ДЖЕРЕЛА ІНФОРМАЦІЇ ..... **42**
ДОДАТКИ ..... **43**

## РЕФЕРАТ

Пояснювальна записка до курсової роботи на тему «Платформа шаблонного створення веб-ресурсів» складається зі вступу, трьох розділів, висновків та джерел інформації.

Текстова частина викладена на 43 сторінках друкованого тексту.

Джерела інформації містять 11 найменувань і займають 1 сторінку. Пояснювальна записка має 5 сторінок додатків. У роботі наведено 18 рисунків. Загальний обсяг роботи - 48 сторінок

У першому розділі було проведено аналіз вимог предметної області, розроблено технічне завдання на розробку програмного комплексу «Kendr», обрано інструментальні засоби для моделювання, описано функціональні та нефункціональні вимоги до програмного продукту. На основі вимог побудовано діаграму варіантів використання та ключові діаграми діяльності, що моделюють поведінку системи.

У другому розділі викладено детальну інформацію щодо структурного моделювання системи. Розроблено об’єктно-орієнтовану модель системи, визначено ключові сутності, їх атрибути та зв'язки за допомогою діаграми класів. Також проаналізовано архітектурні рішення та патерни проектування, що використані в системі. Завершується розділ діаграмами послідовності для ключових сценаріїв взаємодії.

У третьому розділі детально описано фізичну архітектуру та реалізацію системи. Побудовано діаграму компонентів для візуалізації логічної структури та діаграму розгортання для відображення фізичного розміщення компонентів. Наприкінці розділу продемонстровано процес кодогенерації та зворотного проектування.

Висновки містять ключові результати аналізу вимог до програмного продукту та описують етапи розробки системи. Ці етапи включають створення технічного завдання, розробку UML-моделі логічного та фізичного рівнів системи, а також кодогенерацію з використанням засобів UML-моделювання.

# ВСТУП

Станом на 2025 рік інформаційні технології продовжують активно трансформувати спосіб створення та взаємодії з цифровим контентом. Одним із ключових напрямків є розвиток платформ, що спрощують процес розробки та розгортання веб-ресурсів, роблячи їх доступними для широкого кола користувачів без глибоких знань у програмуванні. Платформи для шаблонного створення веб-ресурсів стали невід'ємною частиною сучасного цифрового середовища.

Зростання популярності таких платформ обумовлено їх здатністю забезпечувати швидке, економічно ефективне створення веб-сайтів, лендінгів, портфоліо та інших онлайн-представництв. Це стосується як індивідуальних користувачів, так і малого та середнього бізнесу, що стимулює розробку нових інноваційних рішень у цій сфері.

Сучасні системи шаблонного створення веб-ресурсів не тільки дозволяють генерувати візуально привабливі сторінки, але й відкривають нові можливості для інтеграції з іншими сервісами, такі як системи керування контентом (CMS), електронна комерція, аналітичні інструменти та соціальні мережі. Такі системи стають основою для автоматизації процесів представництва в інтернеті, маркетингу та інших галузях.

Метою курсової роботи є дослідження архітектури, функціоналу та особливостей моделювання програмного комплексу «Kendr», який є платформою шаблонного створення веб-ресурсів.

**Завданням на курсову роботу є:**

- Аналіз сучасних рішень та підходів у розробці веб-платформ для створення веб-ресурсів;
- Проектування архітектури програмного забезпечення із використанням CASE-засобів;
- Моделювання функціональних компонентів: автентифікація, створення та керування веб-сайтами, модерація контенту;

- Аналіз взаємодії сервісів у системі;
- Розробка діаграм та моделей для уніфікації процесів проектування;
- Практична реалізація ключових компонентів системи.

**Предмет дослідження:** методи та засоби проектування програмного забезпечення із використанням CASE-технологій.

**Об'єкт дослідження:** процеси проектування програмного забезпечення веб-платформ для шаблонного створення веб-ресурсів.

# РОЗДІЛ 1. АНАЛІЗ ВИМОГ КОРИСТУВАЧА ТА КОНЦЕПТУАЛЬНЕ МОДЕЛЮВАННЯ

## 1.1. Технічне завдання на розробку системи

**Найменування програмного засобу:** Повне найменування програмної системи: «Платформа шаблонного створення веб-ресурсів» (надалі «система»). Коротка назва програмної системи - «Kendr».

**Призначення розробки та область застосування:** Програмна система «Kendr» призначена для широкого кола користувачів (малого бізнесу, фрілансерів, приватних осіб), які бажають швидко створювати веб-ресурси (візитівки, портфоліо, невеликі магазини) на основі готових шаблонів без навичок програмування. Система може бути використана в особистих, професійних та комерційних цілях.

**Мета:** Веб-платформа дозволить користувачам надавати простий та інтуїтивно зрозумілий інструмент для швидкого створення, налаштування та керування власними веб-сайтами через мережу інтернет.

**Найменування розробника та замовника:** Розробник даного продукту - студент групи ПЗ-22-3 (надалі «розробник»). Замовник програмного продукту - кафедра інженерії програмного забезпечення Державного університету «Житомирська політехніка» в межах виконання курсової з дисципліни «Моделювання та аналіз програмного забезпечення» (надалі «замовник»).

**Документ на підставі якого ведеться розробка:** Робота ведеться на підставі навчального плану за напрямом 121 «Інженерія програмного забезпечення».

**Вимоги до функціональних характеристик:** Програмна система має забезпечувати:

- Реєстрацію та аутентифікацію користувачів;
- Створення та керування веб-сайтами на основі шаблонів;
- Налаштування контенту та зовнішнього вигляду сайту;

- Для сайтів-магазинів: додавання товарів, керування категоріями та обробка замовлень;
- Адміністрування та модерацію сайтів (призупинення, видача попереджень);
- Отримання сповіщень та звернення до служби підтримки.

**Організація вхідних і вихідних даних:**

- **Вхідні дані:**
  - Дані користувачів (логін, пароль, профільна інформація);
  - Контент для сайтів (тексти, зображення);
  - Дані про товари (назва, ціна, кількість);
  - Налаштування сайту (вибір шаблону, кольорова схема).
- **Вихідні дані:**
  - Візуалізація створених веб-сайтів;
  - Дані профілю користувача та перелік його сайтів;
  - Попередження від адміністрації;
  - Сповіщення.

**Часові характеристики і розмір пам’яті, необхідної для роботи програми:**

- Час реакції на дії користувача: до 0,5 сек.
- Час з’єднання з сервером: до 1 сек.
- Обсяг оперативної пам’яті: не менше 1 ГБ.
- Доступність системи: 99% цілодобово.

**Вимоги до надійного функціонування:** Програма повинна нормально функціонувати за безперебійної роботи ПК та інтернет-з’єднання. У разі виникнення апаратних або програмних збоїв необхідно забезпечити:

- Автоматичне відновлення сесії: Система повинна запам’ятовувати активну сесію користувача та її стан.
- Збереження даних у режимі реального часу: Всі зміни контенту сайту, налаштування, дані про товари та замовлення повинні моментально

фіксуватися на сервері. Це дозволяє уникнути втрати інформації при непередбачених ситуаціях.

- Резервування бази даних: Для запобігання втраті даних у разі збою обладнання або програмних помилок слід налаштувати регулярне резервне копіювання бази даних.
- Моніторинг і виявлення збоїв: Впровадити механізми автоматичного моніторингу стану системи.
- Робота в автономному режимі: У разі втрати з'єднання з інтернетом, редактор сайту повинен, по можливості, зберігати незбережені зміни локально. Після відновлення підключення система автоматично синхронізує зміни з сервером.

**Контроль вхідної і вихідної інформації:** Для контролю коректності вхідної інформації та захисту від помилок оператора:

- Перевірка коректності формату введення даних;
- Захист від SQL-ін'єкцій та кроссайтних атак;
- Використання шифрування для передавання конфіденційних даних.

**Час відновлення після відмови:** Час відновлення після відмови, не пов'язаною з роботою програми, повинен складатися із: часу перезапуску операційної системи; часу запуску сервера БД (підключення до сервера) запуску виконуваного файлу, часу повторного введення або зчитування даних.

**Умови експлуатації і збереження:** Програма використовується у багатокористувацькому середовищі з доступом до мережі інтернет, збереження забезпечується створенням резервних копій адміністраторами сайту.

**Вимоги до інформаційної і програмної сумісності:** Система призначена для роботи через сучасні браузери (Chrome, Firefox, Edge). Підтримує багатокористувацьке середовище з доступом до інтернету.

**Вимоги до складу і параметрів технічних засобів:** Система повинна підтримувати та виконувати вимоги на комп'ютері наступних технологій:

- React.JS (v18.2.0)
- Node.js (v16+)
- Базою даних MySQL (v8+)

**Вимоги до програмної документації:** Програмна документація повинна включати наступні відомості:

- Детальна документація API для інтеграції з іншими системами.
- Користувацька документація з описом основних функцій та використання системи. Під час оформлення пояснювальної записки до курсової роботи дані відомості містяться в 1 та 2 розділах курсової роботи.

### 1.2. Обґрунтування вибору засобів моделювання

У процесі вибору засобів для моделювання програмного забезпечення на мові UML, було проведено порівняльний аналіз п'яти популярних інструментів: Visual Paradigm, PlantUML, Draw.io, Lucidchart, та StarUML.

**Visual Paradigm:**

- Переваги:
  - Має надзвичайно багатий функціонал для моделювання (UML, BPMN, ERD, SysML).
  - Має потужні інструменти для кодогенерації та зворотного проектування, зокрема з DDL схем.
  - Інтегровані інструменти для керування проектами та спільної роботи.
- Недоліки:
  - Вимагає оплати за повний функціонал.
  - Інтерфейс перевантажений та складний для новачків.

**PlantUML:**

- Переваги:
  - Підхід "Diagram as Code", що дозволяє писати діаграми у текстовому редакторі.
  - Дуже швидке створення та редагування діаграм.

- Безкоштовний, відкритий код та має плагіни для всіх сучасних IDE.

- Недоліки:
  - Автоматичне розташування елементів може створювати неідеальні візуальні макети.
  - Вимагає вивчення власного синтаксису.
  - Відсутні функції CASE-інструментів, такі як кодогенерація з DDL.

**Draw.io :**

- Переваги:
  - Безкоштовний, інтуїтивно зрозумілий та працює у браузері.
  - Пропонує готові шаблони та підтримує багато форматів (PNG, SVG, PDF).
  - Добре інтегрується з хмарними сховищами.
- Недоліки:
  - Дозволяє просто малювати діаграми, бо не є CASE-інструментом.
  - Відсутність можливості автоматичної генерації коду або створення логічної моделі.

**Lucidchart:**

- Переваги:
  - Зручний для командної роботи в реальному часі.
  - Легко інтегрується з багатьма сервісами.
- Недоліки:
  - Базовий функціонал у безкоштовній версії суттєво обмежений.
  - Більше орієнтований на бізнес-діаграми та блок-схеми, ніж на UML-моделювання.

**StarUML:**

- Переваги:
  - Надає зручний та інтуїтивно зрозумілий інтерфейс для створення діаграм UML.

- Має багато плагінів для генерації коду (Java, C# та ін.).

- Недоліки:
  - Відсутність автоматичного вирівнювання та допомоги у розміщенні компонентів.
  - Платна модель використання.

**Висновок щодо вибору:**

Було обрано пару PlantUML та Visual Paradigm, перший буде використовуватись для побудови діаграм в розділах 2 та 3, другий для кодогенерації, необхідної в розділі 3.

### 1.3. Аналіз вимог до програмного продукту

Для забезпечення ефективного та зручного створення веб-ресурсів на платформі «Kendr» визначено високорівневі вимоги, які повинна задовольняти система. Ці вимоги спрямовані на створення безпечного, функціонального та інтуїтивно зрозумілого середовища для взаємодії між різними ролями користувачів.

Повний каталог вимог наведено у Додатку А, глосарій термінів - у Додатку Б. Аналіз вимог користувачів дозволив визначити варіанти використання платформи. У системі «Kendr» розрізняють трьох ключових акторів: Гість (неавторизований), Користувач (зареєстрований) та Адміністратор. Короткий опис акторів представлено в табл. 1.1.

Таблиця 1.1. Виявлення акторів

| Актор | Опис ролі |
|---|---|
| **Гість** | Має доступ до перегляду публічних сайтів, створених на платформі. Може зареєструватися або увійти в систему для отримання розширених можливостей. |
| **Користувач** | Зареєстрований учасник системи. Може створювати нові сайти на основі шаблонів, керувати власними сайтами (редагувати контент і налаштування), звертатися до служби підтримки. |
| **Адміністратор** | Має повний доступ до панелі адміністрування. Керує користувачами, модерує сайти (призупиняє, видає попередження, видаляє) та обробляє звернення у службу підтримки. |

На основі аналізу акторів та функціональних вимог (Додаток А) було виявлено ключові варіанти використання, що описують основні бізнес-процеси системи.

Таблиця 1.2. Виявлення варіантів використання

| **Основний актор** | **Найменування варіанту використання** | **Опис / Формулювання** |
|---|---|---|
| Гість | Реєстрація | Дозволяє створити новий обліковий запис у системі для подальшої авторизації та користування функціоналом. |
| | Авторизація | Дозволяє ввести облікові дані та увійти до системи під власним акаунтом. |
| Користувач | Створення сайту | Дозволяє обрати шаблон і створити новий веб-ресурс на платформі. |
| | Керування сайтом | Дозволяє редагувати контент (через редактор блоків), змінювати налаштування, публікувати або видаляти власний сайт. |
| | Додавання товару до кошика | Дозволяє користувачу сайту-магазину додати обраний товар для подальшої покупки. |
| | Звернення до підтримки | Дозволяє створювати нові запити (тікети) до служби підтримки та відстежувати їх статус. |
| Адміністратор | Призупинення сайту | Дозволяє тимчасово заблокувати сайт користувача у випадку порушення правил, із можливістю видати попередження. |
| | Відповідь на звернення | Дозволяє переглядати запити користувачів до служби підтримки та надавати офіційні відповіді. |

На основі виявлених акторів та варіантів використання побудовано діаграму варіантів використання (Рис. 1.1), що візуалізує загальну архітектуру взаємодії в системі «Kendr».

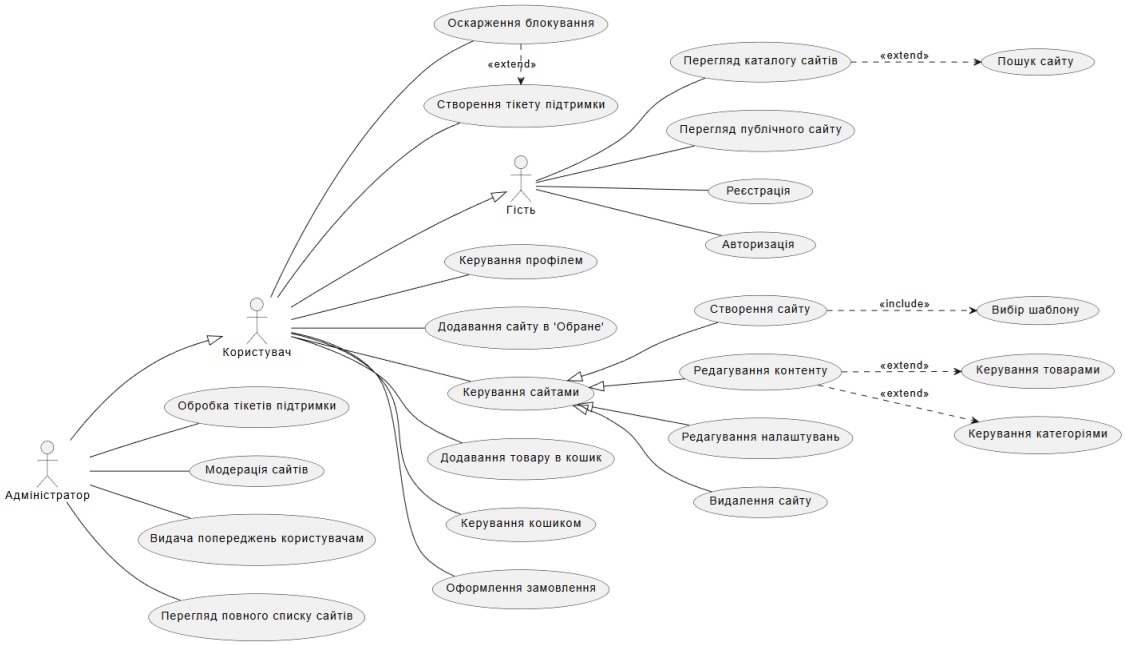


Рис. 1.1. Діаграма варіантів використання системи

## Висновки до першого розділу

У першому розділі було проаналізовано вимоги до програмного продукту «Kendr» та, виходячи з них, було сформовано технічне завдання.

Для візуального моделювання було обрано пару із PlantUML для швидкого створення діаграм та Visual Paradigm для кодогенерації.

На основі технічного завдання було виявлено трьох основних акторів: Гість, Користувач та Адміністратор. Для кожного з них було визначено ключові варіанти використання, які описують їхню основну взаємодію з системою. Ці моделі є основою для подальшого, більш детального моделювання поведінки системи.

# РОЗДІЛ 2. РОЗРОБКА МОДЕЛІ ПРОГРАМНОГО КОМПЛЕКСУ НА ЛОГІЧНОМУ РІВНІ

## 2.1. Алгоритм роботи та стани програмної системи

Діаграми активності, разом з діаграмами варіантів використання, вважаються діаграмами поведінки, оскільки вони описують те, що має відбуватися в системі, яка моделюється.

Діаграма активності допомагає моделювати послідовності дій у бізнес-процесах. Вона показує, як методи або дії взаємодіють між собою. Ці послідовності можуть бути різними галузями обробки даних або діями, які відбуваються паралельно. Діаграми активності схожі на блок-схеми для алгоритмів. Вони представлені у вигляді графу, де дії - це вузли, а зв’язки між ними - переходи.

Кожний стан на діаграмі активності відповідає виконанню деякої елементарної операції, а перехід в наступний стан виконується тільки після завершення цієї операції. Для візуального розділення відповідальності між різними акторами (наприклад, Користувач) та системними компонентами (наприклад, Frontend та Backend) використовуються "доріжки" (Swimlanes).

Функції діаграм активності у даному проекті:

- **Демонстрація логіки алгоритмів:** Візуалізація складних умовних потоків (наприклад, процес модерації).
- **Ілюстрація бізнес-процесів:** Показ взаємодії між користувачем та різними рівнями системи (клієнт, сервер).
- **Роз’яснення сценаріїв використання:** Деталізація ключових прецедентів, визначених у Розділі 1.3.

Для системи «Kendr» було розроблено п'ять ключових діаграм активності, що описують основні процеси.

**Процес створення сайту:**

Діаграма активності на Рис. 2.1 показує процес створення нового сайту користувачем. Це ключовий варіант використання, що демонструє взаємодію трьох сторін: Користувача, Frontend та Backend.

Процес починається з ініціативи користувача. На доріжці Система (Frontend) відбувається валідація на клієнті - перевірка, чи заповнені поля "назва" та "унікальний шлях". Це розвантажує сервер від обробки завідомо некоректних запитів.

Ключовим кроком на Backend є перевірка унікальності адреси сайту (site_path). Це критична бізнес-вимога, оскільки два сайти не можуть існувати за однаковою URL-адресою. Якщо шлях унікальний, сервер виконує дві важливі дії: (1) знаходить обраний шаблон в БД, щоб отримати default_block_content (JSON-структуру блоків за замовчуванням), та (2) створює новий запис у таблиці sites, копіюючи туди цей контент. Це гарантує, що користувач отримає не порожній сайт, а готовий до редагування ресурс.

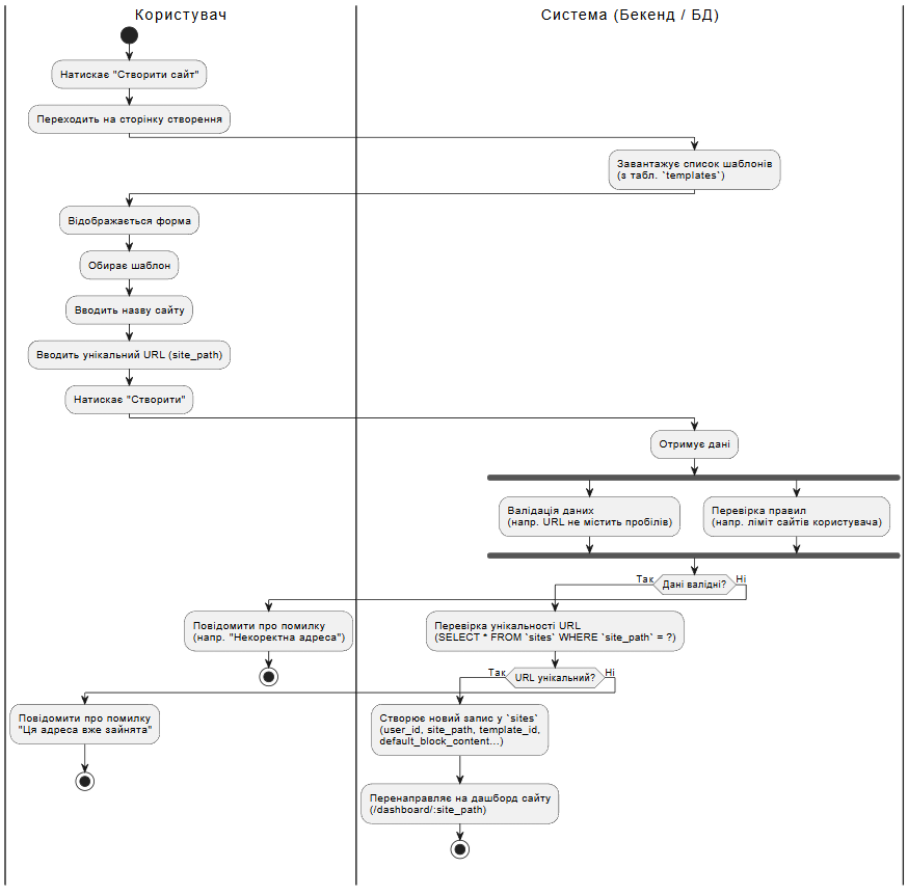


Рис. 2.1. Діаграма активності "Створення сайту користувачем"

**Процес редагування контенту:**

На Рис. 2.2 показано роботу блокового редактора. Цей процес навмисно оптимізовано для високої продуктивності та хорошого користувацького досвіду (UX).

Ключове архітектурне рішення полягає в тому, що всі операції з контентом (додавання, видалення, зміна порядку блоків) відбуваються **виключно на Frontend**. Система (Frontend) маніпулює локальним масивом JSON об'єктів. Це дозволяє інтерфейсу реагувати миттєво, без затримок на мережеві запити.

Система (Backend) залучається лише один раз - коли користувач натискає кнопку "Зберегти". У цей момент Frontend надсилає фінальний JSON масив, який Backend просто приймає та повністю перезаписує (UPDATE) у полі block_content для відповідної сторінки в БД. Такий підхід значно знижує навантаження на сервер та базу даних.

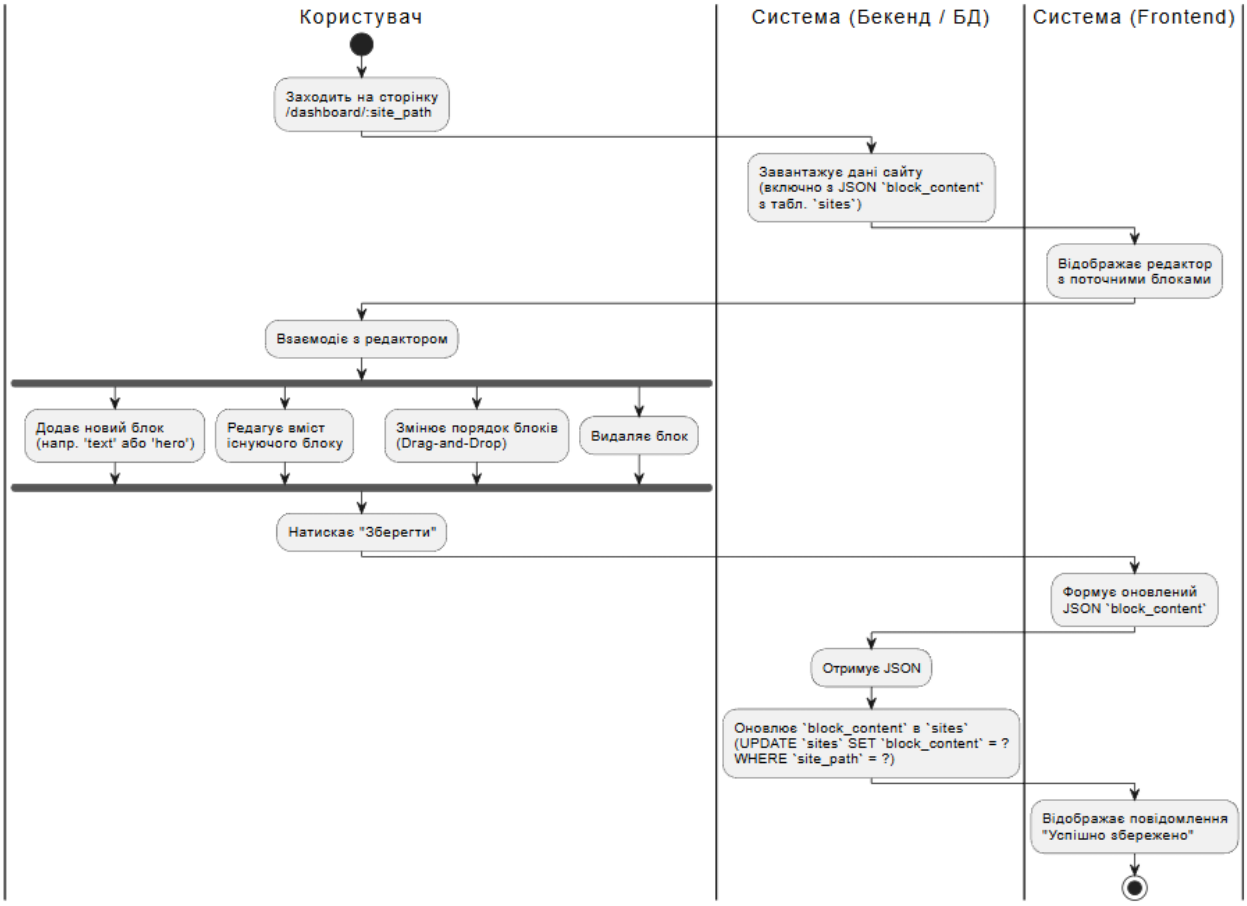


Рис. 2.2. Діаграма активності "Редагування контенту (Блоковий редактор)"

**Процес взаємодії з магазином:**

На Рис. 2.3 деталізовано логіку додавання товару до кошика. Цей процес також відбувається повністю на Frontend для миттєвого відгуку.

Діаграма ілюструє кілька важливих правил:

- **Тільки для авторизованих:** Гість (неавторизований користувач) не може додати товар, його буде перенаправлено на сторінку входу.
- **Власник не може купувати:** Система перевіряє, чи не є поточний користувач власником сайту. Це логічне обмеження, щоб власники не могли "накручувати" покупки на власному сайті.
- **Перевірка наявності:** Ключова перевірка stock_quantity > 0 гарантує, що користувач не може додати до кошика товар, якого немає на складі.

Якщо всі перевірки пройдено, кошик оновлюється у localStorage. Це забезпечує збереження кошика між сесіями користувача, навіть якщо він закриє браузер.

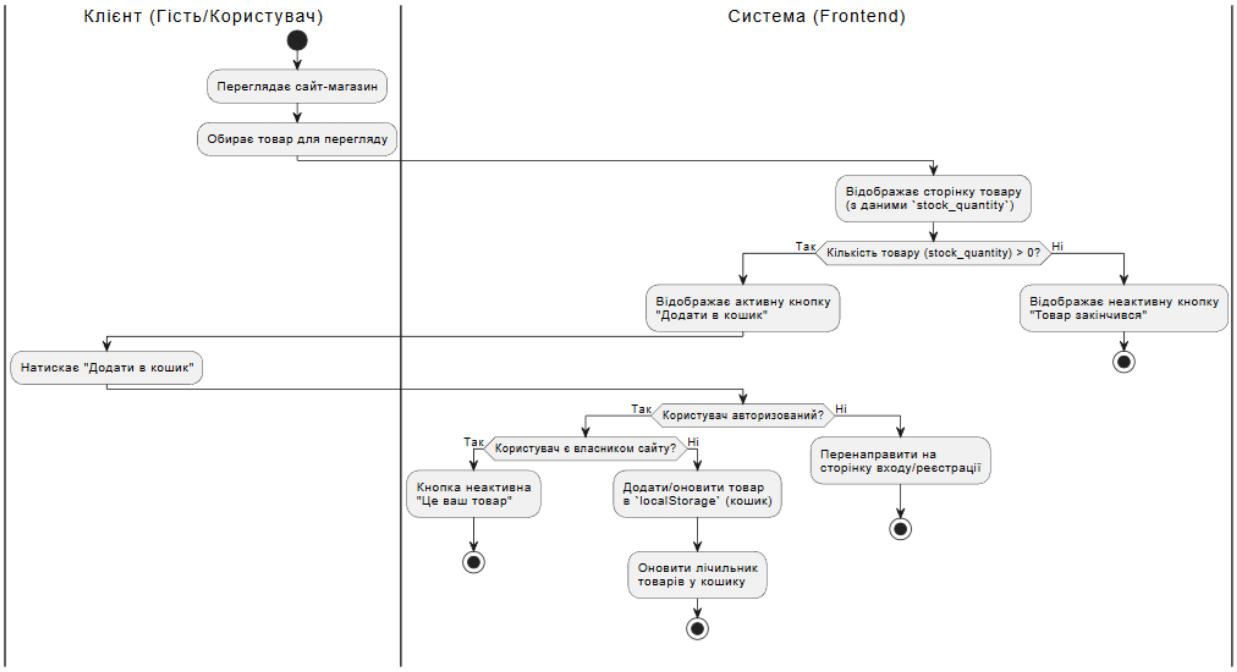


Рис. 2.3. Діаграма активності "Взаємодія Клієнта з сайтом-магазином"

**Процес призупинення сайту:**

На Рис. 2.4 показано складний процес модерації, ініційований Адміністратором. Це яскравий приклад того, як одна дія користувача запускає ланцюжок пов'язаних бізнес-правил на Backend.

Коли Адміністратор призупиняє сайт, Backend виконує не одну, а цілу серію операцій:

- Змінює статус сайту на suspended.
- Встановлює дату автоматичного видалення (deletion_scheduled _for=NOW()+3d), реалізуючи "м'яке видалення".
- Створює запис про порушення у user_warnings.
- Умовна логіка: Перевіряє загальну кількість попереджень у користувача. Якщо ліміт (напр., 3) перевищено, Backend автоматично запускає процедуру повного видалення акаунту користувача та всіх його даних.

Ця діаграма показує, як система автоматизує політику модерації платформи.

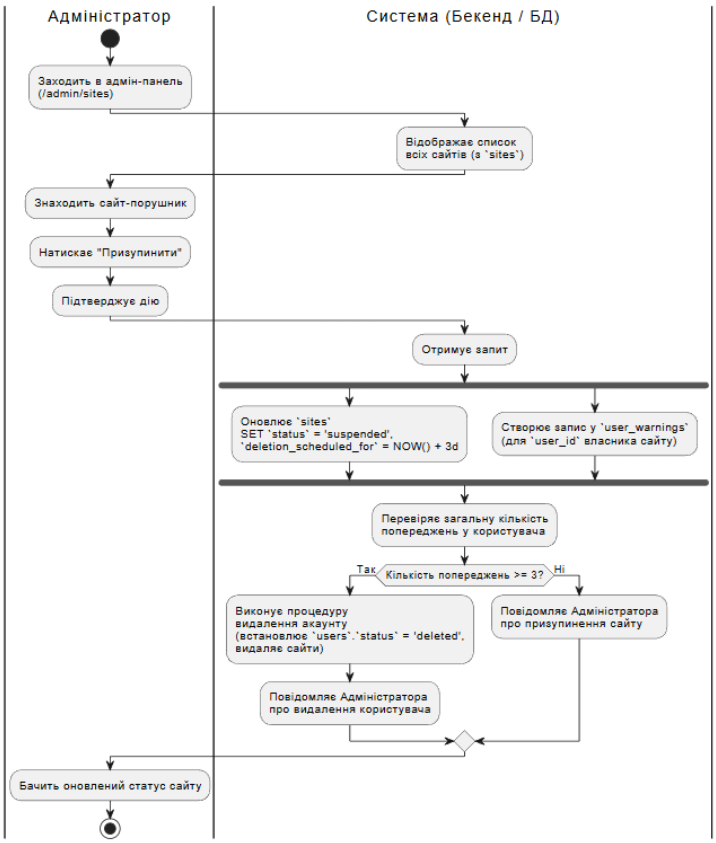


Рис. 2.4. Діаграма активності "Призупинення сайту Адміністратором"

**Процес обробки тікету підтримки:**

На Рис. 2.5 показано повний життєвий цикл звернення до служби підтримки. Ця діаграма чітко ілюструє, як "доріжки" (Swimlanes) показують передачу відповідальності.

Процес є циклічним (while) і триває, доки тікет не отримає статус closed.

- Користувач ініціює процес (створює тікет).
- Відповідальність переходить до Адміністратора. Він бачить тікет (status='open'), відповідає на нього, і Backend змінює статус на answered.
- Відповідальність повертається до Користувача. Він отримує сповіщення і переглядає відповідь.
- Тут 2 розгалуження:
  - **(Так)** Якщо проблема вирішена, Користувач закриває тікет, і цикл завершується.
  - **(Ні)** Якщо проблема не вирішена, Користувач додає коментар, Backend змінює статус на in_progress (щоб тікет знову з'явився у черзі адміністратора), і цикл while починається знову.

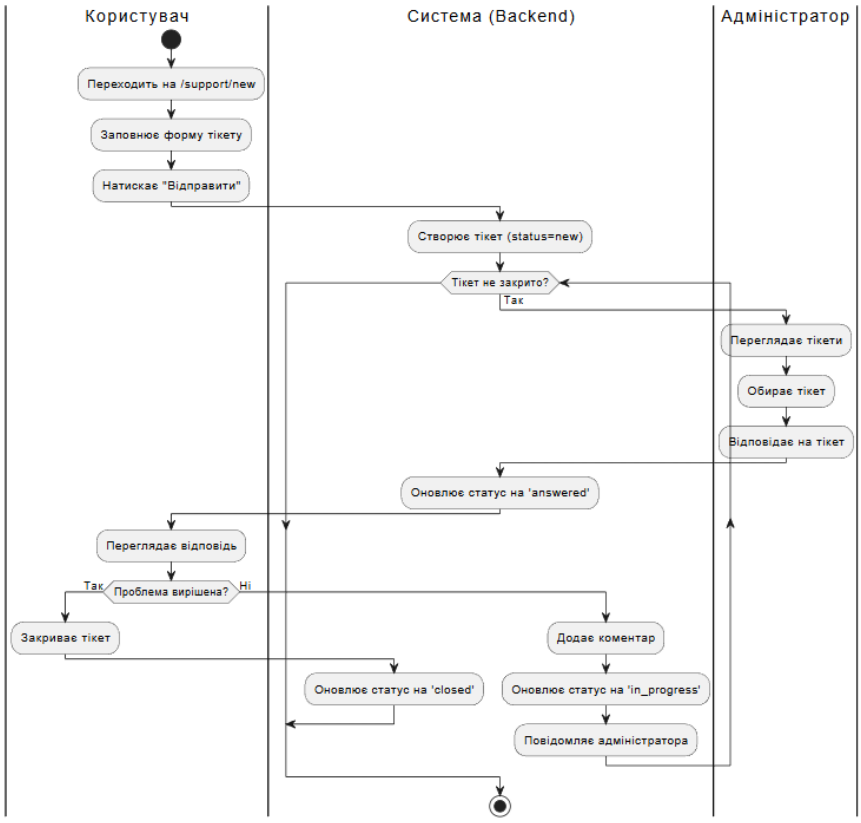


Рис. 2.5. Діаграма активності "Обробка тікету підтримки"

## 2.2. Об’єктно-орієнтована модель системи

Тепер, коли вже визначені основні функції (Розділ 1.3) та алгоритми роботи (Розділ 2.1) веб-платформи «Kendr», можна створити об’єктно-

орієнтовану модель системи. Нижче наведено опис основних сутностей, їх зв’язків і використаних патернів проектування.

**Основні сутності системи (Доменна модель):**

- **User** - користувач системи, який може реєструватися, володіти сайтами, керувати медіафайлами та звертатися до підтримки. Має роль (user або admin).
- **Site** - ключова сутність; веб-сайт, який створює User. Має унікальний шлях (site_path), статус та набір сторінок.
- **Template** - "креслення" для Site, що визначає початковий набір сторінок та контенту (default_block_content).
- **Page** - окрема сторінка в межах сайту. Містить JSON поле block_content, що реалізує блоковий редактор.
- **Product** - товар, що належить до Site з шаблоном "Магазин". Має ціну, опис, залишки на складі.
- **Category** - категорія, що групує товари в межах одного Site.
- **Tag** - глобальна мітка (тег) для категоризації Site у загальному каталозі.
- **SupportTicket** - звернення (тікет) до служби підтримки, яке створює User.
- **TicketReply** - відповідь (коментар) в межах одного SupportTicket.
- **UserWarning** - попередження, яке Адміністратор видає User.
- **UserMedia** - завантажений користувачем файл (аватар, логотип).
- **User_Favorite** та **Site_Tag** - асоціативні сутності для реалізації зв'язків "багато-до-багатьох".

Між сутностями використовуються різні типи зв'язків.

- **Агрегація (Aggregation):** Використовується між User та Site (User "володіє" Site). Це зв'язок "один-до-багатьох", але Site може існувати в базі даних, навіть якщо User буде (теоретично) видалений (хоча в нашій системі стоїть каскадне видалення).

- **Композиція (Composition):** Більш суворий варіант. Site "складається" зі Page. SupportTicket "складається" з TicketReply. Це означає, що при видаленні Site, всі пов'язані з ним Page також видаляються (життєвий цикл залежить від контейнера).
- **Асоціація (Association):** Звичайний зв'язок між незалежними сутностями, наприклад, між Site та Template (багато сайтів *використовують* один шаблон).

**Перерахування (Enumeration)**

На діаграмі класів також було виділено кілька перерахувань (ENUM) для зручного опису типів та статусів:

- **User.role** - визначає ролі: user, admin.
- **Site.status** - визначає стан сайту: draft, published, suspended.
- **SupportTicket.status** - визначає стан тікету: open, in_progress, resolved, closed.
- **UserWarning.severity** - визначає серйозність попередження: low, medium, high.

На Рис. 2.6 показано діаграму класів предметної області (Domain Model), що ілюструє ці сутності та їхні зв'язки.

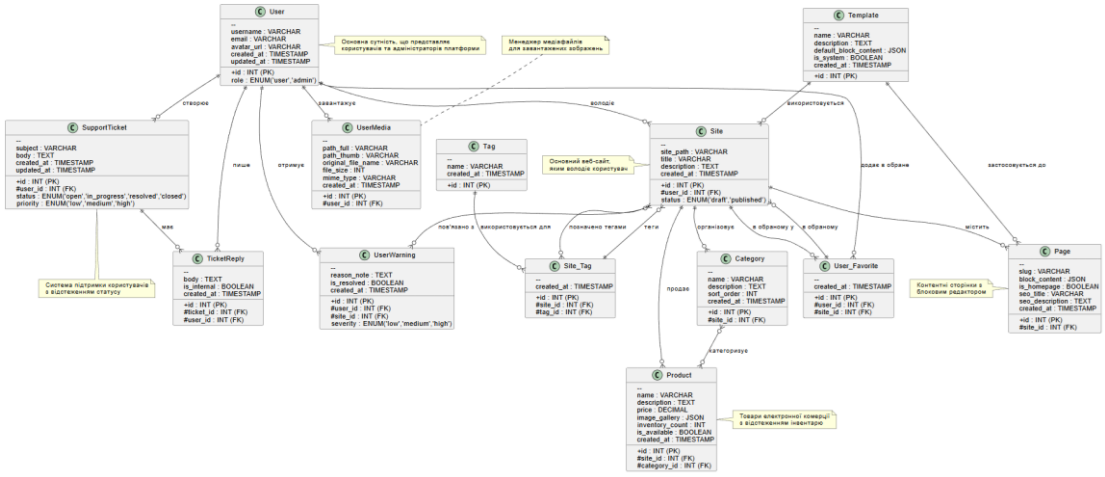


Рис. 2.6. Діаграма класів без використання паттернівпаттернів

**Патерни проектування:**

На додаток до діаграми предметної області (Рис. 2.6), яка показує *що* система зберігає, було розроблено діаграму класів реалізації (Рис. 2.7), яка

показує *як* система побудована. Вона демонструє ключові патерни проектування, використані в архітектурі «Kendr».

- **Model-View-Controller (MVC):** Це основний архітектурний патерн, використаний на **Backend**. Структура чітко розділяє обов'язки:
  - **Model (Модель):** Класи, що відповідають за дані та бізнес-логіку (напр., User, Site).
  - **Controller (Контролер):** Класи, що приймають HTTP-запити, взаємодіють з моделями та відправляють відповідь (напр., SiteController, authController).
  - **View (Представлення):** Роль "Представлення" у даній архітектурі виконує Frontend (React) додаток, який споживає JSON дані, що надсилаються контролерами.
- **Singleton (Одинак):** Цей патерн використовується у двох ключових місцях:
  - **Database (Підключення до БД):** Файл db.js створює один пул з'єднань (pool) з MySQL і експортує його. Всі інші частини Backend (моделі, контролери) імпортують і використовують цей єдиний екземпляр пулу, запобігаючи створенню нових з'єднань на кожен запит.
  - **ApiClient:** На Frontend створюється і конфігурується один екземпляр axios (apiClient), який використовується для всіх HTTP-запитів.
- **Object Pool (Пул об'єктів):** Тісно пов'язаний з Singleton. Клас Database використовує mysql.createPool(). Це класичний приклад патерну "Пул об'єктів", який керує набором готових з'єднань з БД для пере-використання.
- **Interceptor (Перехоплювач):** Реалізовано в ApiClient на Frontend. apiClient.interceptors.request.use(...) дозволяє "перехоплювати" вихідні запити для їх модифікації - у даному випадку, для автоматичного додавання JWT-токену авторизації у заголовок кожного запиту.
- **Route Guard (Захисник маршруту):** Компонент AdminRouteGuard на Frontend. Це компонент вищого порядку, який перевіряє права доступу

користувача (чи role === 'admin') перед тим, як дозволити доступ до захищених маршрутів (наприклад, /admin).

- **Factory Method (Фабричний метод) / Lazy Loading (Лініве завантаження):** Компонент TemplateLoader на Frontend діє як "фабрика", яка вирішує, який саме компонент-шаблон (SimpleBioTemplate чи ShopTemplate) потрібно завантажити. Одночасно, використання React.lazy() реалізує патерн "Лініве завантаження", підвантажуючи код шаблону лише за потреби.

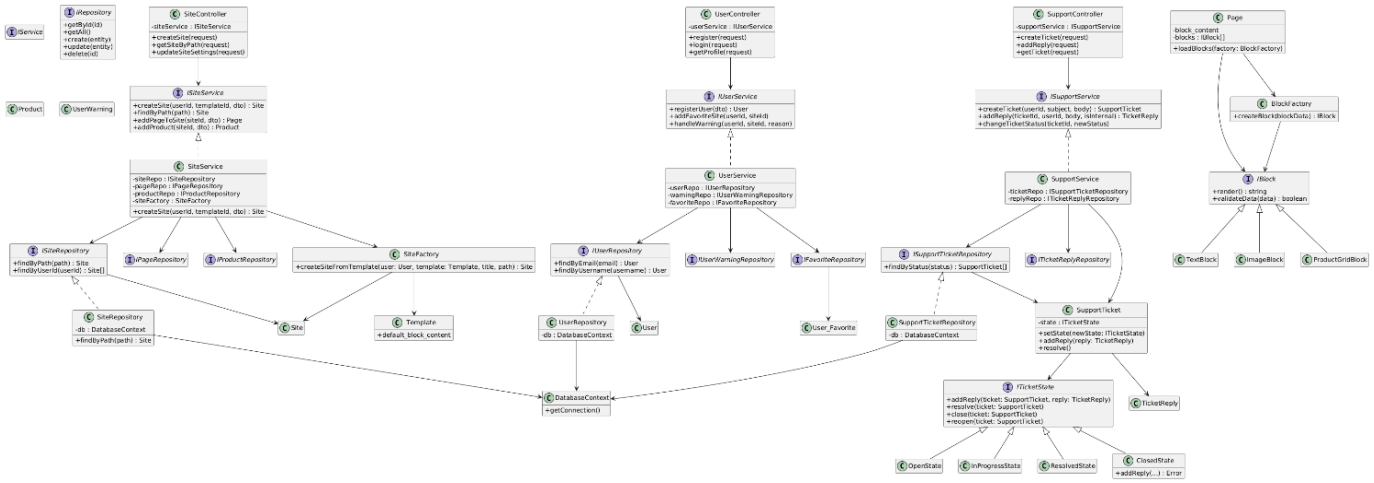

Рис. 2.7. Діаграма класів реалізації з патернами проектування

## 2.3. Комунікації і послідовність взаємодії об’єктів системи

Діаграми кооперації (комунікації) та послідовності є ключовими інструментами для моделювання динамічної поведінки системи. Якщо діаграми класів (Розділ 2.2) показують статичну структуру, то ці діаграми демонструють, як екземпляри цих класів (об'єкти) обмінюються повідомленнями (викликають методи) у часі для виконання конкретного завдання або варіанту використання.

**Діаграма кооперації (комунікації):**

Діаграма кооперації фокусується на структурних зв'язках між об'єктами, що беруть участь у сценарії. Вона показує, хто з ким взаємодіє, а вкладена нумерація повідомлень ілюструє послідовність, хоч і менш явно, ніж діаграма послідовності.

На Рис. 2.8 представлено діаграму кооперації для одного з найскладніших сценаріїв системи

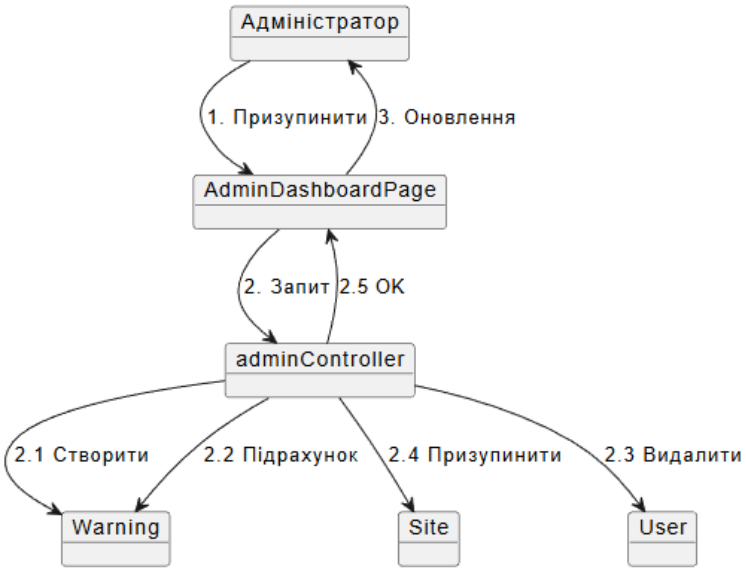


Рис. 2.8. Діаграма кооперації "Призупинення сайту"

Діаграма (Рис. 2.8) ілюструє потік керування:

- Процес ініціює :Адміністратор, надсилаючи повідомлення “Призупинити до :AdminDashboardPage”.
- Інтерфейс передає запит “Запит” до :adminController на сервері.
- Контролер стає центром оркестрації та послідовно виконує низку вкладених операцій:
  - Створити: Викликає модель :Warning для створення запису про попередження.
  - Підрахунок: Викликає :Warning для перевірки загальної кількості попереджень у користувача.
  - Видалити: Умовний виклик до моделі :User (якщо кількість попереджень перевищує ліміт).
  - Призупинити: Виклик до моделі :Site для зміни її статусу.
- Після завершення серверних операцій, :Controller повертає відповідь 2.5 OK до :Dashboard.
- В кінці, :Dashboard інформує :Адміністратор про результат “Оновлення”, оновивши інтерфейс.

**Діаграми послідовності (Sequence Diagrams)**

Діаграми послідовності роблять акцент на **часовій послідовності** викликів. Вони детально показують "лінії життя" (lifelines) об'єктів та точний порядок обміну повідомленнями згори донизу. Нижче наведено три ключові діаграми послідовності для системи «Kendr».

**Сценарій 1: Призупинення сайту Адміністратором**

На Рис. 2.9 показано той самий процес, що й на діаграмі кооперації, але з чітким фокусом на часі та умовній логіці. Ця діаграма моделює критично важливий процес модерації.

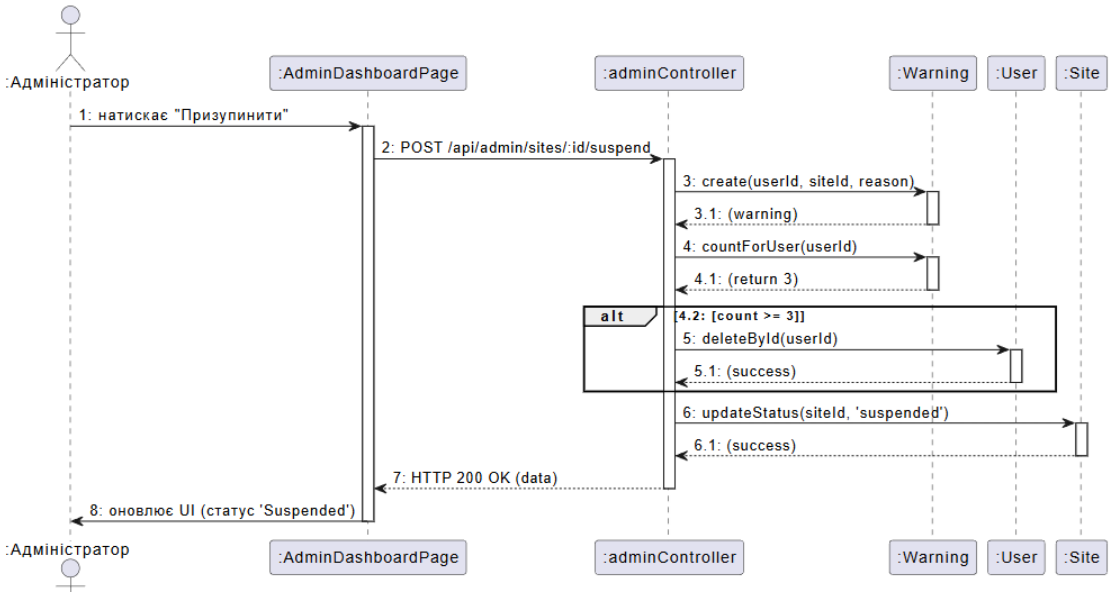


Рис. 2.9. Діаграма послідовності "Призупинення сайту"

Процес (Рис. 2.9) ініціюється :Адміністратором на клієнті (крок 1). :AdminDashboardPage надсилає захищений API-запит (який проходить через middleware, тут не показаний) до :adminController (крок 2).

Контролер є центром бізнес-логіки. Він не просто змінює статус сайту, а виконує повний процес модерації:

- **Створення попередження (крок 3):** Спершу створюється запис у :WarningModel, фіксуючи факт порушення.
- **Перевірка лічильника (крок 4):** Негайно виконується запит до тієї ж моделі :WarningModel для підрахунку загальної кількості активних попереджень у цього користувача.

- **Умовна логіка (блок alt, крок 4.2):** Це найважливіша частина. **Тільки якщо** умова [count >= 3] виконується, контролер ініціює додатковий, більш серйозний процес - повне видалення користувача через виклик :UserModel (крок 5).
- **Виконання основного завдання (крок 6):** Незалежно від блоку alt, контролер виконує головне завдання - призупиняє сайт через :SiteModel.

Така централізація логіки в контролері забезпечує послідовне та автоматичне застосування правил платформи, звільняючи адміністратора від необхідності виконувати ці кроки вручну.

**Сценарій 2: Створення сайту Користувачем**

На Рис. 2.10 показано основний "щасливий шлях" (happy path) створення нового сайту, що є ключовою функцією платформи «Kendr».

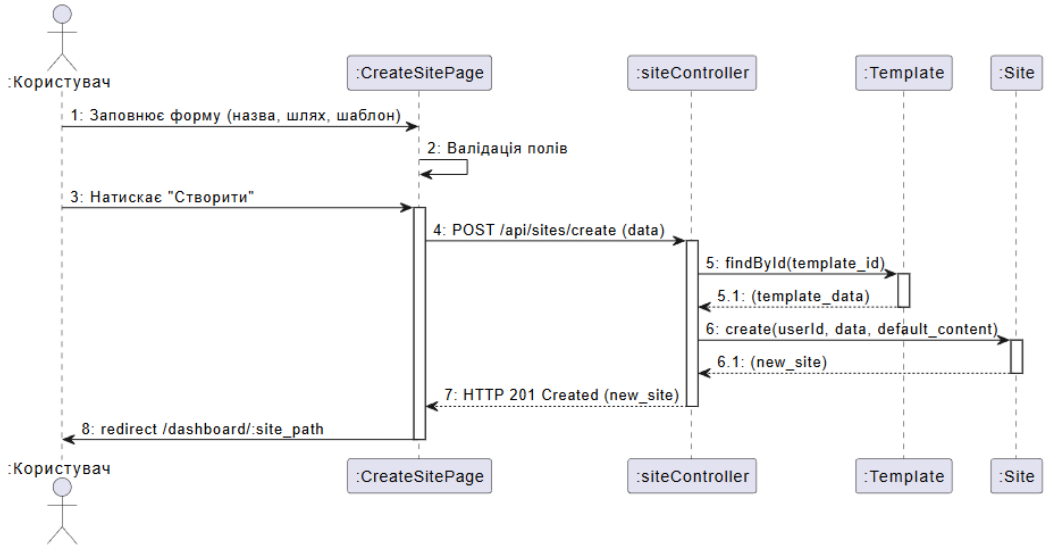


Рис. 2.10. Діаграма послідовності "Створення сайту"

Процес (Рис. 2.10) демонструє чіткий поділ відповідальності. :CreateSitePage (UI) відповідає за збір даних та клієнтську валідацію.

Серверний :siteController (крок 4) оркеструє процес створення. Ключовою особливістю системи «Kendr» є використання шаблонів. Тому контролер спершу звертається до :TemplateModel (крок 5), щоб отримати default_block_content - той JSON, що описує початкову структуру блоків для нового сайту.

Лише отримавши ці дані (крок 5.1), контролер викликає :SiteModel (крок 6), передаючи їй дані користувача (userId), дані з форми (data) та контент шаблону (default_content). Це гарантує, що жоден сайт не може бути створений без контенту за замовчуванням. Успішна відповідь HTTP 201 Created (крок 7) підтверджує створення ресурсу, а клієнт виконує перенаправлення (крок 8), забезпечуючи гарний користувацький досвід.

**Сценарій 3: Створення тікету підтримки**

На Рис. 2.11 показано процес створення запиту до служби підтримки, що є базовим сценарієм CRUD (Create, Read, Update, Delete).

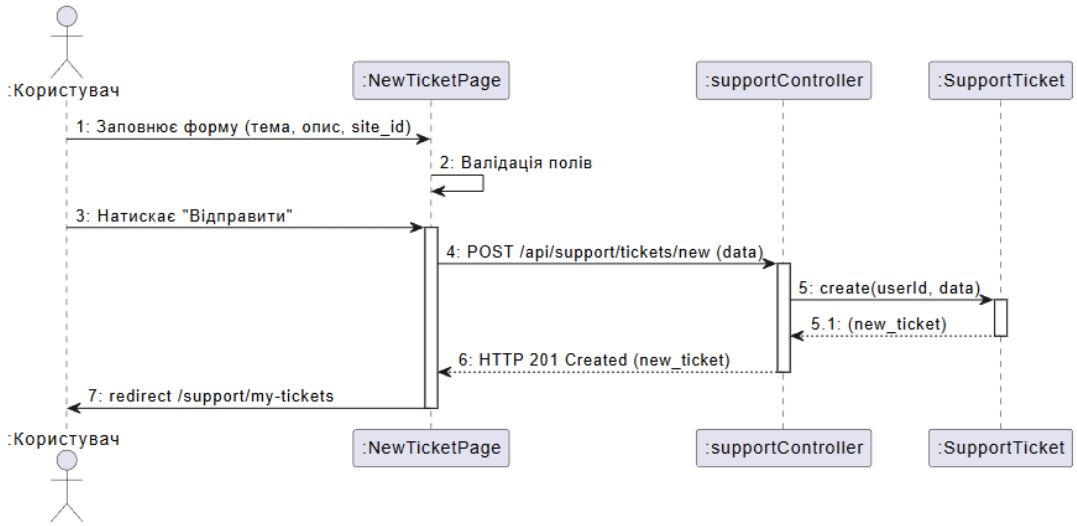


Рис. 2.11. Діаграма послідовності "Створення тікету підтримки"

Цей процес (Рис. 2.11) є найпростішим з трьох, оскільки він ілюструє прямий потік створення даних. Після валідації на клієнті (кроки 1-3), :supportController (крок 4) викликає лише одну модель :SupportTicket (крок 5) для створення нового запису в базі даних.

Ця діаграма показує початкову фазу процесу підтримки (створення), тоді як повний життєвий цикл (відповідь, закриття) деталізовано у діаграмі активності Рис. 2.5. Успішна відповідь HTTP 201 (крок 6) та перенаправлення (крок 7) є важливою частиною потоку, що дає користувачеві чіткий зворотний зв'язок про те, що його запит прийнято в роботу.

## Висновки до другого розділу

У другому розділі було розроблено повну логічну модель системи «Kendr», що детально ілюструє її поведінку, структуру та взаємодію компонентів.

Було деталізовано поведінку системи (через діаграми активності), її статичну структуру (через діаграми класів та патернів), а також динамічну взаємодію об'єктів (через діаграми кооперації та послідовності).

Ці моделі надають повне уявлення про логічну архітектуру системи «Kendr» та слугують основою для фізичного моделювання та реалізації, описаних у наступному розділі.

# РОЗДІЛ 3. ФІЗИЧНА МОДЕЛЬ ТА ПРОТОТИП ПРОГРАМНОГО КОМПЛЕКСУ

## 3.1. Взаємодія компонентів системи

При розробці веб-платформи «Kendr» було застосовано класичну багаторівневу архітектуру (N-Tier), яка чітко розділяє систему на логічні та фізичні компоненти. Ключовим принципом такої архітектури є розділення відповідальності, що забезпечує високу гнучкість, масштабованість та простоту супроводу коду.

Архітектура системи «Kendr» поділена на три основні логічні рівні, які ми детально моделюємо у діаграмі компонентів (див. Рис. 3.1 у наступному підрозділі):

- **Рівень представлення (Frontend / Клієнт):** Це клієнтська частина, що виконується у веб-браузері користувача. Вона реалізована за допомогою бібліотеки **React**. Цей рівень відповідає виключно за користувацький інтерфейс (UI), керування станом на боці клієнта та взаємодію з користувачем. Він не містить бізнес-логіки, а лише надсилає HTTP-запити до серверної частини (Backend) та відображає отримані дані.
- **Рівень бізнес-логіки (Backend / Сервер додатку):** Це серверна частина, реалізована на платформі Node.js з використанням фреймворку Express.js. Цей рівень є "мозком" системи і, у свою чергу, використовує архітектурний патерн **MVC (Model-View-Controller)**:
  - **Маршрути (Routes):** Виступають як вхідна точка API. Вони приймають HTTP-запити від клієнта (React) і передають їх на обробку відповідним контролерам.
  - **Проміжне ПЗ (Middleware):** Компоненти, що виконують наскрізні завдання, такі як автентифікація (перевірка JWT-токену), валідація вхідних даних та обробка помилок.
  - **Контролери (Controllers):** Оркеструють обробку запиту. Вони викликають необхідні моделі (сервіси) для виконання бізнес-логіки та повертають клієнту відповідь (зазвичай у форматі JSON).

- **Моделі (Models):** Рівень, що відповідає за безпосередню бізнес-логіку та взаємодію з базою даних. Інкапсулює всі SQL-запити.

• **Рівень даних (Persistence / База даних):** Фізичне сховище даних. У даному проекті це реляційна система керування базами даних (СУБД) **MySQL**. Сервер додатку (Backend) є єдиним компонентом, який має прямий доступ до цього рівня, що забезпечує високий рівень безпеки та цілісності даних.

Такий поділ гарантує, що залежності спрямовані лише в одному напрямку (Frontend залежить від Backend, Backend залежить від DB), що дозволяє незалежно модифікувати та масштабувати кожен компонент системи.

### 3.2. Архітектура програмного комплексу та його розгортання

Діаграма розгортання призначена для моделювання фізичної архітектури системи. За допомогою цієї діаграми можна оцінити необхідну апаратну/серверну конфігурацію, на якій працюватимуть окремі компоненти (артефакти) системи, і описати їх взаємодію між собою.

Вузол (node) - це фізичний або віртуальний компонент системи (сервер, пристрій), який має певні обчислювальні ресурси. На діаграмі також зображені відносини між вузлами у вигляді фізичних з’єднань, які дозволяють обмінюватися даними.

Діаграми розгортання та компонентів (див. Рис. 3.1) тісно пов'язані. Діаграма розгортання показує, як саме логічні компоненти "розгортаються" на фізичних вузлах.

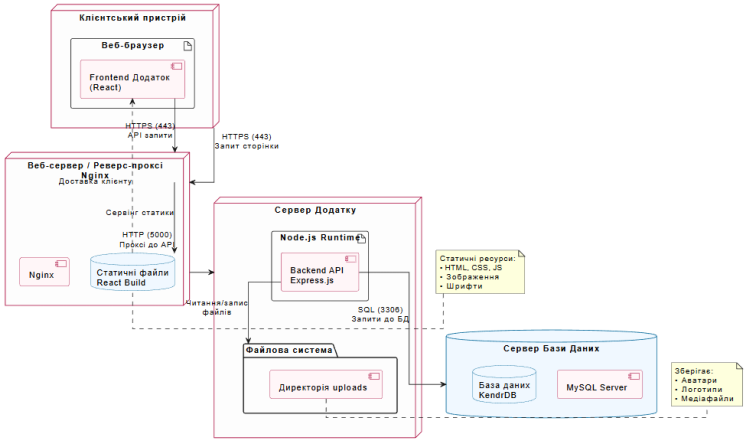


Рис. 3.1. Діаграма розгортання

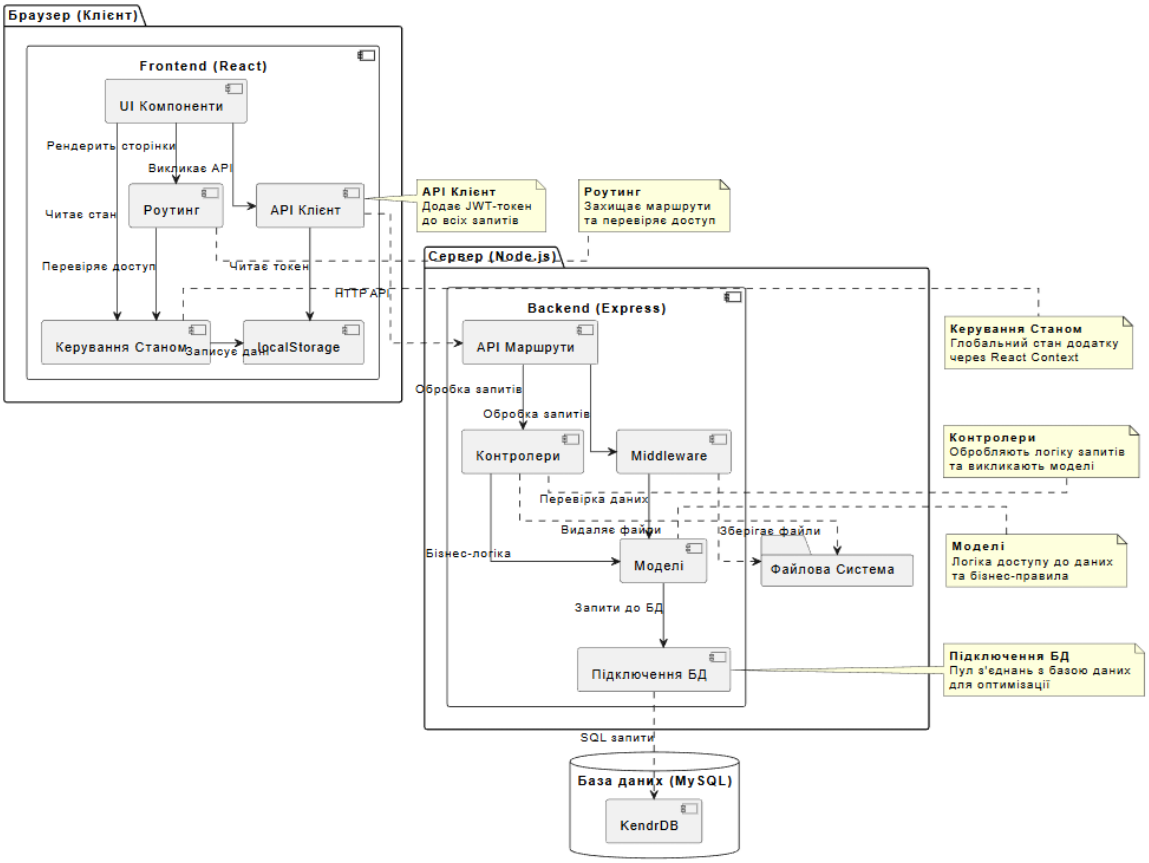


Рис. 3.2. Діаграма компонентів

Система складається з чотирьох основних вузлів (див. Рис. 3.1):

- **Клієнтський пристрій (Client Devices):**
  - Це будь-який пристрій користувача (ПК, смартфон, планшет), на якому запущено веб-браузер.
  - На цьому вузлі виконується артефакт Frontend Додаток (React), який завантажується з веб-сервера. Він відповідає за весь користувацький інтерфейс.
- **Веб-сервер / Реверс-проксі (Nginx):**
  - Це сервер, що є "обличчям" системи. Він слухає HTTPS запити (порт 443).
  - На ньому розгорнуто компонент Nginx, який виконує дві функції:
    - **Сервінг статики:** Миттєво віддає готові, зібрані файли Frontend Додатку (HTML, CSS, JS) з вузла Статичні файли (React Build).

- **Проксіювання API:** Всі запити, що починаються з /api/, він безпечно перенаправляє (проксіює) на Сервер Додатку, приховуючи його реальну адресу.

- **Сервер Додатку (Application Server):**
  - Це вузол, де виконується основна бізнес-логіка.
  - На ньому розгорнуто артефакт Node.js Runtime, який, у свою чергу, містить компонент Backend API (Express.js).
  - Цей вузол має доступ до локальної Файлової системи для збереження та віддачі завантажених файлів (аватари, логотипи) з директорії uploads.
- **Сервер Бази Даних (Database Server):**
  - Окремий, ізольований вузол, призначений виключно для зберігання даних.
  - На ньому розгорнуто компонент MySQL Server, який керує базою даних KendrDB.
  - Доступ до цього вузла (через порт 3306) має **тільки** Сервер Додатку.

**Архітектурна взаємодія системи** Взаємодія у системі «Kendr» починається з Клієнтського пристрою, коли користувач відкриває адресу сайту у браузері. Запит надходить на Веб-сервер (Nginx). Nginx миттєво віддає клієнту зібраний Frontend Додаток (React).

Після завантаження, React-додаток виконує API-запити (наприклад, для входу або отримання списку сайтів) на ту саму адресу (на /api/...). Nginx "ловить" ці запити і перенаправляє їх на Сервер Додатку (Node.js). Node.js обробляє запит, виконує бізнес-логіку (наприклад, перевіряє пароль) і, за потреби, звертається до Сервера Бази Даних (MySQL) для отримання або запису даних. MySQL повертає дані Node.js, Node.js формує JSON-відповідь і повертає її Nginx, а Nginx повертає її клієнту.

Така архітектура з реверс-проксі забезпечує високу продуктивність (статика віддається миттєво) та безпеку (клієнт ніколи не має прямого доступу до API сервера чи бази даних).

### 3.3. Генерування програмного коду для прототипу програмного комплексу

Процес кодогенерації у Visual Paradigm починається з його встановлення та налаштування. Для цього необхідно завантажити програму з офіційного сайту, встановити її та запустити. Далі створюється новий проєкт або відкривається вже існуючий, у якому планується виконувати моделювання та генерацію коду.

Перед початком роботи важливо переконатись, що у Visual Paradigm активовано відповідне розширення для обраної мови програмування, наприклад, для Java. Це налаштування можна здійснити через меню програми, обравши пункт "Application Options" і відкривши "Technology Selection".

Процес кодогенерації в даній роботі є демонстрацією повного циклу MDE (Model-Driven Engineering) та реалізується за допомогою CASE-інструменту Visual Paradigm.

Замість ручного створення UML-моделі з нуля, що є трудомістким процесом, ми використаємо функцію зворотного проектування (Reverse Engineering). Цей підхід дозволяє автоматично згенерувати UML-діаграму на основі вже існуючої бази даних.

Процес складається з наступних кроків:

1. У Visual Paradigm ми запускаємо інструмент Tools > DB > Reverse Database....
2. Обираємо цільову базу даних MySQL та запускаємо процес.

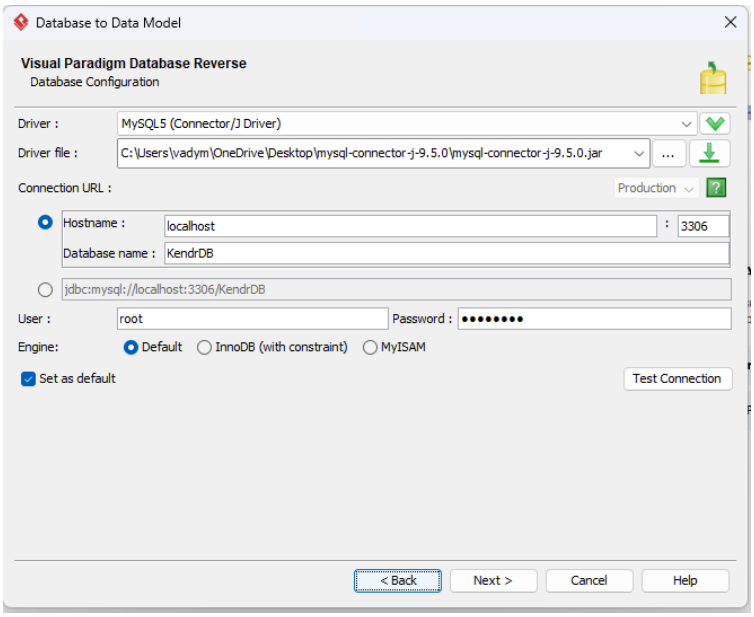


Рис. 3.3. Налаштування з'єднання для зворотного проектування бази

Програма аналізує SQL-файл і автоматично створює в "Model Explorer" повну логічну UML-модель, що точно відповідає структурі бази даних. Після цього ми візуалізуємо її, створивши нову діаграму класів та перетягнувши на неї згенеровані класи.

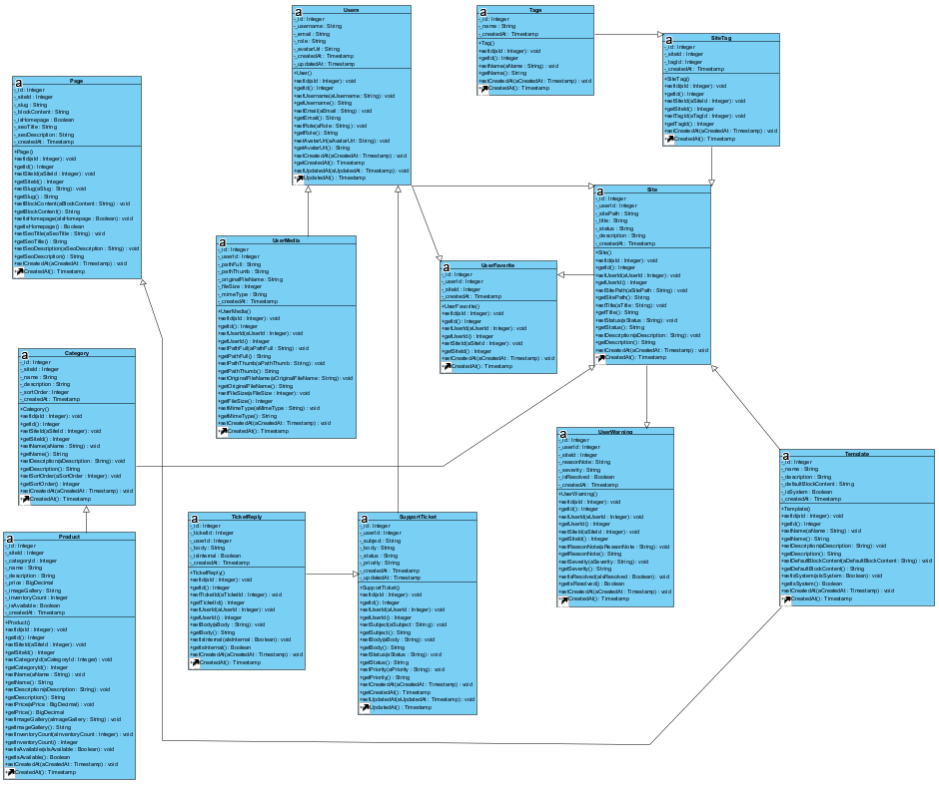

Рис. 3.4. Діаграма класів, автоматично згенерована

Тепер, коли ми маємо готову UML-модель, ми можемо використати її для прямого проектування (Forward Engineering) - генерації коду.

Налаштування процесу генерації здійснюється через контекстне меню (обравши потрібні класи або пакет) Code > Instant Generator.... У діалоговому вікні (Рис. 3.4) вибирається цільова мова програмування (у нашому випадку Java, для демонстрації академічного принципу) та каталог для збереження файлів.

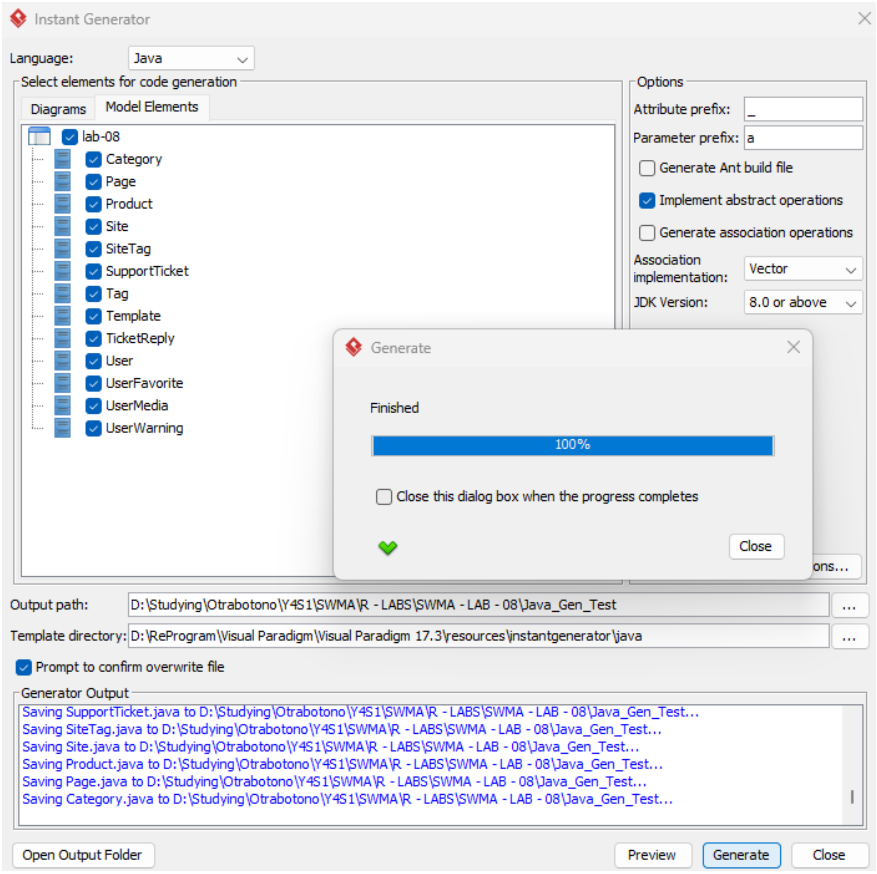


Рис. 3.5. Налаштування генератора коду (Instant Generator) для Java

Після запуску процесу генерації, Visual Paradigm автоматично створює файли коду (.java) для кожного обраного класу з UML-моделі, включаючи атрибути (поля) та асоціації (зв'язки з іншими класами).

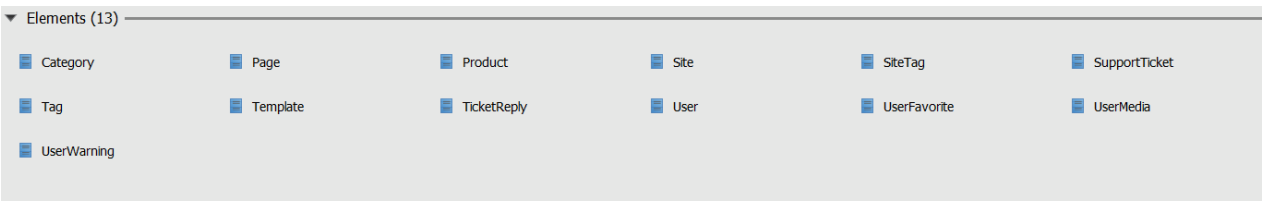


Рис. 3.6. Файлова структура згенерованого коду

Лістинг згенерованого файлу User.java:

```
public class User {
    private Integer _id;
    private String _username;
    private String _email;
    private String _role;
    private String _avatarUrl;
    private Timestamp _createdAt;
    private Timestamp _updatedAt;
    public User() {
        throw new UnsupportedOperationException();
    }
    public void setId(Integer aId) {
        this._id = aId;
    }
    public Integer getId() {
        return this._id;
    }
    public void setUsername(String aUsername) {
        this._username = aUsername;
    }
    public String getUsername() {
        return this._username;
    }
    public void setEmail(String aEmail) {
        this._email = aEmail;
    }
    public String getEmail() {
        return this._email;
    }
    public void setRole(String aRole) {
        this._role = aRole;
    }
    public String getRole() {
        return this._role;
    }
    public void setAvatarUrl(String aAvatarUrl) {
        this._avatarUrl = aAvatarUrl;
    }
    public String getAvatarUrl() {
        return this._avatarUrl;
    }
    public void setCreatedAt(Timestamp aCreatedAt) {
        this._createdAt = aCreatedAt;
    }
    public Timestamp getCreatedAt() {
        return this._createdAt;
    }
    public void setUpdatedAt(Timestamp aUpdatedAt) {
        this._updatedAt = aUpdatedAt;
    }
    public Timestamp getUpdatedAt() {
        return this._updatedAt;
    }
}
```

Лістинг згенерованого файлу Site.java:

```
public class Site {
    private Integer _id;
    private Integer _userId;
    private String _sitePath;
    private String _title;
    private String _status;
    private String _description;
    private Timestamp _createdAt;
    public Site() {
```

```
        throw new UnsupportedOperationException();
    }
    public void setId(Integer aId) {
        this._id = aId;
    }
    public Integer getId() {
        return this._id;
    }
    public void setUserId(Integer aUserId) {
        this._userId = aUserId;
    }
    public Integer getUserId() {
        return this._userId;
    }
    public void setSitePath(String aSitePath) {
        this._sitePath = aSitePath;
    }
    public String getSitePath() {
        return this._sitePath;
    }
    public void setTitle(String aTitle) {
        this._title = aTitle;
    }
    public String getTitle() {
        return this._title;
    }
    public void setStatus(String aStatus) {
        this._status = aStatus;
    }
    public String getStatus() {
        return this._status;
    }
    public void setDescription(String aDescription) {
        this._description = aDescription;
    }
    public String getDescription() {
        return this._description;
    }
    public void setCreatedAt(Timestamp aCreatedAt) {
        this._createdAt = aCreatedAt;
    }
    public Timestamp getCreatedAt() {
        return this._createdAt;
    }
}
```

Після генерації коду, можна відкрити ці файли-каркаси у своєму IDE та наповнити їх реальною бізнес-логікою.

## Висновки до третього розділу

У третьому розділі було детально розглянуто фізичну модель системи «Kendr» та продемонстровано процес генерації прототипу програмного комплексу.

Побудовано діаграму компонентів (Рис. 3.2), що візуалізує логічний поділ системи на три основні рівні: Frontend (React), Backend (Node.js/Express) та Рівень даних (MySQL). Також було розроблено діаграму розгортання (Рис.

3.1), яка описує фізичну архітектуру системи, включаючи Клієнтський пристрій, Веб-сервер (Nginx), Сервер Додатку (Node.js) та Сервер Бази Даних.

Наприкінці розділу було описано та продемонстровано ключовий цикл MDE (Model-Driven Engineering) за допомогою CASE-інструменту Visual Paradigm. Ми виконали реверс-інжиніринг, згенерувавши UML-модель з DDL-схеми, а потім, на основі цієї моделі, виконали пряме проектування, згенерувавши код на мові Java. Це доводить гнучкість UML-моделювання та його здатність підтримувати синхронізацію між проектом та кодом.

# ВИСНОВКИ

В ході виконання курсової роботи було проведено повний цикл моделювання та аналізу програмного забезпечення для веб-платформи «Kendr».

На основі аналізу вимог до програмного продукту (викладених у Додатку А) було створено технічне завдання, яке визначає основні бізнес-цілі та функціональні вимоги. Для розробки системи було виявлено трьох основних акторів (Гість, Користувач, Адміністратор) та для кожного з них було визначено варіанти використання, які описують їх взаємодію з системою (Рис. 1.1).

Модель логічного рівня (Розділ 2) ілюструє поведінку системи, її структуру та взаємодію між її компонентами. Для її розробки було створено:

- Діаграми активності
- Діаграми класів
- Діаграми взаємодії

Фізична модель системи (Розділ 3) описує компонентну та фізичну архітектуру. Вона складається з чотирьох основних компонентів, відображених на діаграмі компонентів та діаграмі розгортання : Клієнтський пристрій (React), Веб-сервер (Nginx), Сервер Додатку (Node.js) та Сервер Бази Даних (MySQL).

Прототип програмного комплексу було продемонстровано на основі фізичної моделі системи. Процес кодогенерації було успішно здійснено за допомогою CASE-інструменту Visual Paradigm. Було продемонстровано повний цикл MDE (Model-Driven Engineering): спершу виконано реверс-інжиніринг для отримання UML-моделі з існуючої DDL-схеми (.sql), а потім - пряме проектування для генерації "каркасів" коду (Java) на основі отриманої моделі.

## ДЖЕРЕЛА ІНФОРМАЦІЇ


1. Офіційний сайт Visual Paradigm [Електронний ресурс] - Режим доступу до ресурсу: https://www.visual-paradigm.com/.
2. Документація Visual Paradigm [Електронний ресурс] - Режим доступу до ресурсу: https://circle.visual-paradigm.com/docs/.
3. UML Tutorials від Visual Paradigm [Електронний ресурс] - Режим доступу до ресурсу: https://www.visual-paradigm.com/tutorials/.
4. UML Resource Page від OMG [Електронний ресурс] - Режим доступу до ресурсу: https://www.omg.org/spec/UML/.
5. Oracle Java Tutorials [Електронний ресурс] - Режим доступу до ресурсу: https://docs.oracle.com/javase/tutorial/.
6. Stack Overflow - Тег Visual Paradigm [Електронний ресурс] - Режим доступу до ресурсу: https://stackoverflow.com/questions/tagged/visual-paradigm.
7. UML Class Diagram Tutorial [Електронний ресурс] - Режим доступу до ресурсу: https://www.youtube.com/watch?v=UI6lqHOVHic.
8. Udemy: Courses on Visual Paradigm and UML [Електронний ресурс] - Режим доступу до ресурсу: https://www.udemy.com/.
9. Coursera: Object-Oriented Design and Programming [Електронний ресурс] - Режим доступу до ресурсу: https://www.coursera.org/.
10. Visual Paradigm Community Forum [Електронний ресурс] - Режим доступу до ресурсу: https://forums.visual-paradigm.com/.
11. CodeProject [Електронний ресурс] - Режим доступу до ресурсу: https://www.codeproject.com/.

# ДОДАТКИ

Додаток А

## Каталог вимог

**Бізнес вимоги:**

- **Основні цілі:** Проєкт створюється з метою надати простий та інтуїтивно зрозумілий інструмент для швидкого створення міні-сайтів (візитівок, портфоліо, невеликих магазинів) для малого бізнесу, фрілансерів та приватних осіб без навичок програмування.
- **Представлення проєкту:** Проєкт буде реалізовано у вигляді платформи (сайту), що дозволяє користувачам реєструватися, створювати сайти на основі готових шаблонів та керувати ними.

Вимоги користувачів:

У системі існує кілька ролей з різними потребами.

**Вимоги користувачів-власників сайтів:**

- Можливість створення та редагування особистого профілю (ім'я, аватар, пароль).
- Можливість створювати сайти, обираючи один із запропонованих шаблонів.
- Можливість керувати власними сайтами: редагувати контент (тексти, товари, категорії), змінювати налаштування (назва, статус, теги) та видаляти сайти.
- Можливість переглядати список своїх сайтів, включно з чернетками та заблокованими.
- Можливість додавати сайти інших користувачів до списку "Обране".
- Можливість звернутися до служби підтримки через систему тікетів та оскаржити блокування сайту.

**Вимоги відвідувачів (клієнтів) сайтів:**

- Можливість переглядати загальний каталог опублікованих сайтів з функцією пошуку.
- Можливість вільно переглядати сторінки створених сайтів та їх контент.

- Для сайтів-магазинів: можливість додавати товари в кошик, змінювати їх кількість та оформлювати замовлення.

**Вимоги персоналу веб-додатку (Адміністратори):**

- Можливість перегляду повного списку всіх сайтів на платформі, незалежно від їх статусу.
- Можливість модерувати сайти: призупиняти їх роботу (suspended) з автоматичним видаленням через певний час, переводити в чернетки (draft) або відновлювати (published).
- Можливість видавати користувачам попередження за порушення правил.
- Можливість переглядати звернення користувачів у службу підтримки та відповідати на них.

**Функціональні вимоги:**

- **Автентифікація та авторизація:** В системі повинна бути реалізована можливість реєстрації, входу та виходу з акаунту. Система повинна розрізняти ролі: user (користувач) та admin (адміністратор).
- **Керування сайтами:** Система повинна дозволяти користувачам створювати сайти на основі шаблонів, редагувати їх контент та налаштування, а також видаляти їх.
- **Керування контентом магазину:** Для шаблону "Магазин" система повинна надавати інструменти для створення категорій та керування товарами (додавання, редагування, видалення, облік залишків).
- **Модерація:** Система повинна надавати адміністраторам інструменти для призупинення та видалення сайтів, а також для видачі попереджень користувачам.
- **Система підтримки:** Система повинна дозволяти користувачам створювати звернення (тікети), а адміністраторам - відповідати на них.
- **Збереження даних:** Система повинна надійно зберігати всю інформацію (профілі, сайти, товари) у реляційній базі даних MySQL.

**Нефункціональні вимоги:**

- **Сприйняття:**
  - Час, необхідний для навчання звичайних користувачів базовим функціям (створення та редагування сайту) - не більше 1 години.
  - Час відгуку системи для типових задач (завантаження сторінки, збереження даних) - не більше 5 секунд.
  - Інтерфейс має бути інтуїтивно зрозумілим, адаптивним та мати відповідні підказки.
- **Надійність:**
  - Доступність - час планового обслуговування системи не повинен перевищувати 4% від загального часу роботи.
  - Середній час безвідмовної роботи 20 робочих днів.
- **Продуктивність:**
  - Система повинна підтримувати мінімум 50 одночасно працюючих користувачів, пов’язаних із загальною базою даних.
- **Можливість експлуатації:**
  - Масштабування - система повинна мати архітектуру, що дозволяє збільшувати потужності (продуктивність) зі збільшенням кількості користувачів та сайтів без значного погіршення її роботи.

Додаток Б

## Глосарій

**Адміністратор (Admin)** - користувач із розширеною роллю admin, який має доступ до панелі управління всією платформою. Він може керувати всіма Сайтами, змінювати їх Статус (включно з блокуванням) та обробляти Звернення до підтримки.

**Блок (Block)** - основна структурна одиниця контенту на Сторінці. Кожна сторінка складається з масиву блоків (наприклад, 'hero', 'text', 'catalog_grid'), які зберігаються у форматі JSON.

**Звернення до підтримки (Support Ticket)** - механізм зворотного зв'язку, що дозволяє авторизованому Користувачу створити запит (тікет) до Адміністратора для вирішення проблем або оскарження призупинення Сайту.

**Категорія (Category)** - сутність, що належить Сайту та використовується для групування Товарів. Власник сайту може створювати, редагувати та видаляти категорії.

**Користувач (User)** - обліковий запис, зареєстрований у системі. Характеризується унікальним username та email, має аватар, роль ('user' або 'admin') та може створювати власні Сайти.

**Медіатека (Media Library)** - персональне сховище файлів Користувача, де зберігаються завантажені зображення. Для кожного файлу система автоматично створює повну версію та оптимізовану мініатюру (path_full, path_thumb).

**Обліковий запис (Account)** - див. Користувач.

**Панель управління сайтом (Site Dashboard)** - інтерфейс, доступний власнику Сайту, що надає інструменти для керування його вмістом: редагування Сторінок, керування Товарами та Категоріями, а також зміна загальних налаштувань сайту.

**Попередження (Warning)** - санкція, що застосовується Адміністратором до Користувача, зазвичай одночасно з призупиненням Сайту. Накопичення

трьох активних попереджень призводить до автоматичного видалення Облікового запису Користувача та всіх його даних.

**Призупинений сайт (Suspended Site)** - Сайт, який отримав статус suspended від Адміністратора. Доступ до такого сайту обмежується для всіх, окрім власника та Адміністратора. Призупинення активує таймер автоматичного видалення сайту.

**Редактор блоків (Block Editor)** - візуальний інструмент (BlockEditor.jsx) в Панелі управління сайтом, що дозволяє власнику додавати, видаляти, переміщувати та налаштовувати Блоки контенту на Сторінці.

**Сайт (Site)** - ключова сутність платформи, яку створює Користувач. Являє собою набір Сторінок, доступних за унікальним шляхом (site_path). Кожен сайт має заголовок, логотип, тему оформлення та статус.

**Статус сайту (Site Status)** - поточний стан Сайту, який визначає його видимість. Може бути draft (чернетка, видима лише власнику), published (опубліковано, видно всім) або suspended (призупинено Адміністратором).

**Сторінка (Page)** - окрема веб-сторінка в межах Сайту. Характеризується назвою (name) та шляхом (slug), який має бути унікальним в межах одного сайту. Одна зі сторінок обов'язково позначається як головна (is_homepage = 1).

**Товар (Product)** - елемент каталогу, що належить конкретному Сайту та використовується для функцій електронної комерції. Має назву, опис, ціну, кількість на складі (stock_quantity) та може належати до Категорії.

**Шаблон (Template)** - набір початкових налаштувань та Блоків, який Користувач обирає під час створення нового Сайту. Визначає початковий вигляд та функціональність.