Лабораторна робота № 3-4

Мета: Ознайомлення з основними принципами структурного моделювання та побудови класів. Закріплення практичних навичок побудови UML-діаграм класів та аналізу патернів проектування у програмній системі.

Хід роботи

Завдання №3. Побудова діаграми класів (Domain Model)

Завдання №3.1. Перелік основних сутностей

На основі функціональних вимог було визначено ключові класи, що представляють основні сутності системи:

- **User:** Базовий клас для всіх зареєстрованих учасників.
- **Admin:** Спеціалізований тип користувача (успадковує від User), що визначається атрибутом role = 'admin' та має розширені права.
- **Site:** Основний об'єкт, який створюють користувачі.
- **Template:** "Креслення", на основі якого створюється сайт та його початковий контент (default_block_content).
- **Product:** Сутність, що існує в межах сайту з шаблоном "Магазин".
- **Category:** Групує товари на сайті-магазині.
- **Tag:** Мітка для категоризації та пошуку сайтів.
- **SupportTicket:** Запит від користувача до адміністрації.
- **UserWarning:** Запис про порушення правил користувачем.
- **Favorite:** Асоціативний клас, що реалізує зв'язок "багато-до-багатьох" між User та Site.

Завдання №3.2. Ідентифікація атрибутів класів

Таблиця 3.1. Атрибути класів предметної області

Клас	Атрибут
User	+ id: Int {PK}
	+ username: String {unique}
	+ email: String {unique}
	# password_hash: String

	+ role: Enum [user, admin]
	+ status: Enum [active, suspended]
	+ avatar_url: String
	+ created_at: Timestamp
Administrator	(успадковує User)
Site	+ id: Int {PK}
	+ user_id: Int {FK > User}
	+ template_id: Int {FK > Template}
	+ site_path: String {unique}
	+ title: String
	+ logo_url: String
	+ status: Enum [draft, published, suspended]
	+ view_count: Int
	+ block_content: JSON
	+ created_at: Timestamp
Template	+ id: Int {PK}
	+ name: String {unique}
	+ description: Text
	+ thumbnail_url: String
	+ default_block_content: JSON
Product	+ id: Int {PK}
	+ site_id: Int {FK > Site}
	+ category_id: Int {FK > Category}
	+ name: String
	+ description: Text
	+ price: Decimal
	+ stock_quantity: Int
	+ image_url: String
Category	+ id: Int {PK}
	+ site_id: Int {FK > Site}
	+ name: String
Tag	+ id: Int {PK}
	+ name: String {unique}
SupportTicket	+ id: Int {PK}
	+ user_id: Int {FK > User}
	+ site_id: Int {FK > Site} (optional)
	+ subject: String
	+ status: Enum [new, in_progress, answered, closed]
	+ created_at: Timestamp
TicketReply	+ id: Int {PK}
	+ ticket_id: Int {FK > SupportTicket}
	+ user_id: Int {FK > User}
	+ body: Text
	+ created_at: Timestamp
UserWarning	+ id: Int {PK}
	+ user_id: Int {FK > User}
	+ admin_id: Int {FK > User}
	+ site_id: Int {FK > Site} (optional)
	+ reason_note: Text {optional}
	+ created_at: Timestamp

Завдання №3.3. Визначення методів класів

Таблиця 3.2. Основні методи класів предметної області

Клас	Метод (поведінка)
User	+ register(data): User
	+ login(email, password): String {повертає JWT}

	+ updateProfile(data): bool
	+ updateAvatar(file): bool
	+ getOwnedSites(): Site[]
	+ getFavoriteSites(): Site[]
	+ getTickets(): SupportTicket[]
Administrator	+ suspendSite(site, reason): bool
	+ restoreSite(site): bool
	+ deleteSite(site): bool
	+ replyToTicket(ticket, message): bool
	+ issueWarning(user, site): UserWarning
Site	+ updateSettings(title, status): bool
	+ updateTags(tag_ids): bool
	+ updateBlockContent(json): bool
	+ addProduct(data): Product
	+ addCategory(name): Category
	+ incrementViewCount(): void
Product	+ updateStock(quantity): bool
SupportTicket	+ addReply(user, body): TicketReply
	+ close(): bool

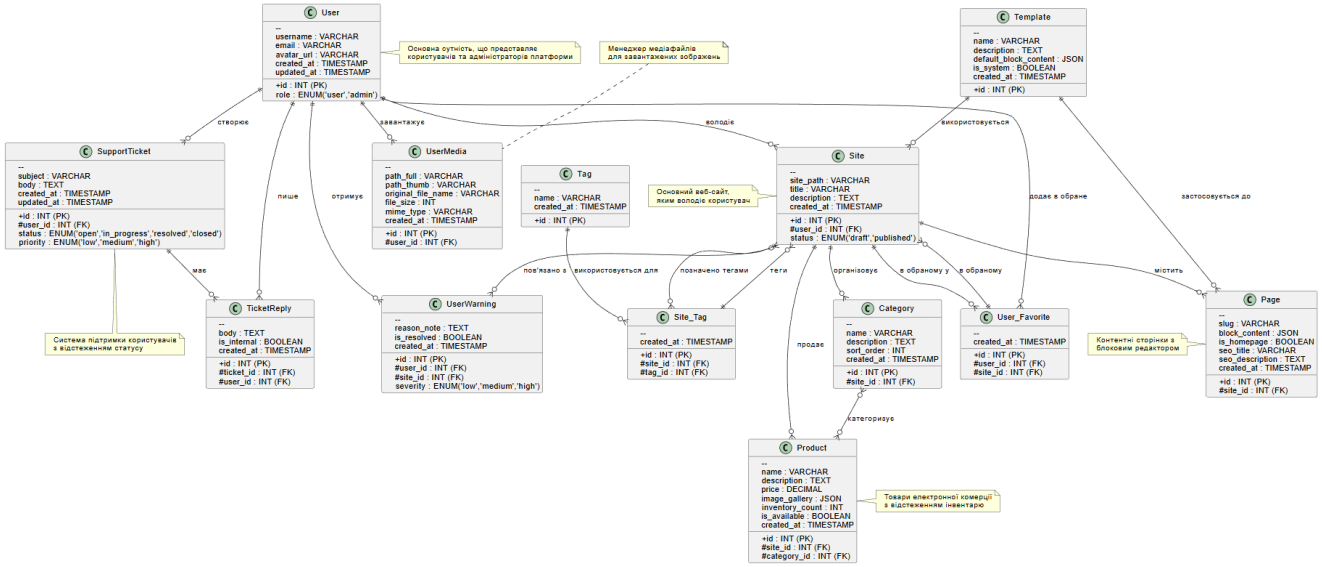

Рис. 3.1. Діаграма класів предметної області

Завдання №4. Побудова діаграми класів реалізації (Патерни проектування)

На відміну від діаграми предметної області, діаграма класів реалізації показує, як саме архітектура системи реалізує цю логіку за допомогою конкретних класів (контролерів, сервісів, компонентів) та патернів проектування.

Завдання №4.1. Аналіз патернів проектування

В архітектурі системи "Kendr" ідентифіковано декілька ключових патернів:

1. **Model-View-Controller (MVC)**: Цей патерн є основою серверної частини (backend).

1.1. **Model (Site)**: Клас, що відповідає за бізнес-логіку та взаємодію з базою даних.

1.2. **View (SiteRoutes)**: Шар, що визначає точки входу API (ендпоінти), які приймають HTTP-запити.

1.3. **Controller (SiteController)**: Отримує запити від View, взаємодіє з Model для обробки даних та повертає відповідь.

2. **Middleware**: Патерн, що дозволяє вбудовувати логіку в конвеєр обробки запиту. Клас VerifyToken є прикладом, що виконує аутентифікацію перед тим, як запит потрапить до контролера.
3. **Singleton (Одинак)**: Гарантує, що клас має лише один екземпляр.

   3.1. **Database**: Створює та експортує єдиний пул з'єднань (pool), який використовується у всій серверній частині.

   3.2. **ApiClient**: Створює єдиний, налаштований екземпляр axios для всіх HTTP-запитів з клієнтської частини.
4. **Object Pool (Пул об'єктів)**: Використовується класом Database. Замість створення нового з'єднання з БД для кожного запиту, пул керує набором готових з'єднань для пере-використання.
5. **Interceptor (Перехоплювач)**: Реалізовано в ApiClient. Дозволяє перехоплювати вихідні запити (або відповіді) для їх модифікації, наприклад, для автоматичного додавання JWT-токену авторизації.
6. **Route Guard (Захисник маршруту)**: Компонент AdminRouteGuard на клієнтській частині (frontend) перевіряє права користувача перед тим, як дозволити доступ до захищеної сторінки (AdminPage).
7. **Factory Method (Фабричний метод) / Lazy Loading (Ліниве завантаження)**: Компонент TemplateLoader діє як "фабрика", яка вирішує, який саме компонент-шаблон (SimpleBioTemplate чи ShopTemplate) потрібно завантажити. Використання React.lazy() реалізує патерн лінивого завантаження, підвантажуючи код шаблону лише за потреби.

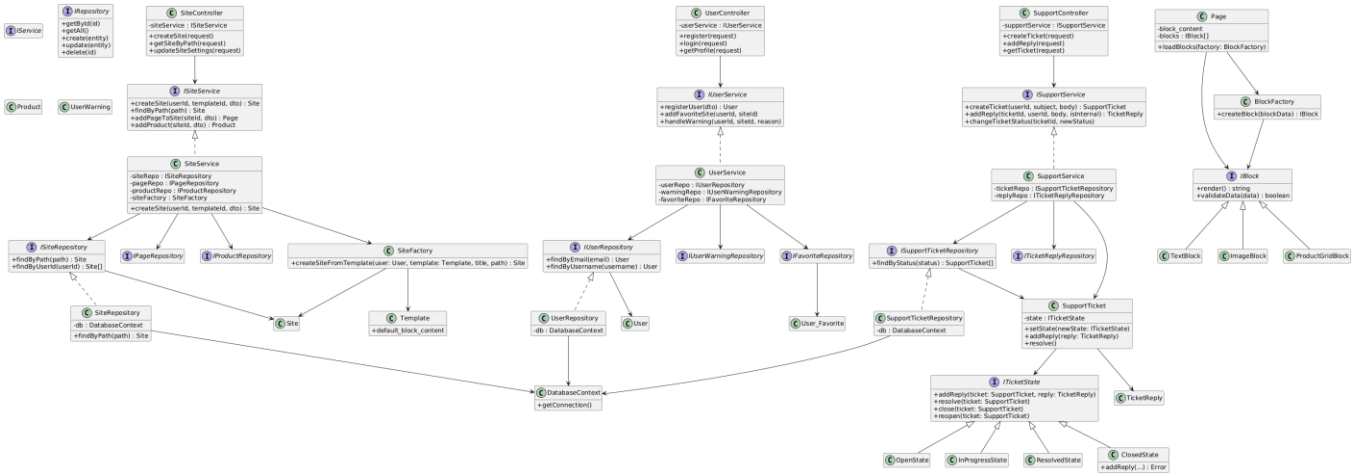

Рис. 4.1. Діаграма класів реалізації з патернами проектування

Висновок: в ході виконання лабораторної роботи було ознайомлено з основними принципами структурного моделювання та побудови класів. Закріплено практичні навички побудови UML-діаграм класів та аналізу патернів проектування у програмній системі.