Звіт

Лабораторна робота №1а

“Моделювання з використанням UML ”

Виконала студентка групи ІПС-21

Сенечко Дана Володимирівна

Design/Implementation Modeling:

1. Обрано існуючий навчальний проект для ознайомлення з MERN stack (MongoDB, Express, React, NodeJS) із використанням Vite (Chakra UI): Link

2. В початковому проєкті зовсім не було реалізовано unit tests, тому я додала їх для backend частини (***включно з перевіркою роботи бази даних***) з використанням Jest (+babel).

3. Реалізовані типи UML діаграм: Use Case, Class, Activity, Object, Component, Deployment та Communication. Також присутній глосарій проекту та документація (JSDOC ***з налаштованими Github Pages/Actions***)

4. Запропоновані зміни та покращення: додати вхід на сайт з певними правами доступу для різних користувачів (використання JWT); створити сторінку товару, додати опис товару (необов'язковий); можливість додавати товари в кошик для зареєстрованих користувачів та переглядати кошик з доданими товарами (або без них :)). Звичайні користувачі (як зареєстровані, так і не зареєстровані) матимуть можливість тільки переглядати сайт, в той час як користувачі з правом доступу admin матимуть доступ до всіх можливостей сайту, таких як додавання, редагування та видалення товарів. Також відповідно додати unit tests для нововведень. З легких стилістичних змін також є зміна кольорової гами сайту з синього кольору на рожевий :)

5. Запропоновані зміни було реалізовано: Link (комміти можна глянути тут, я мусила перенести проект в окремий репозиторій для налаштування документації).

6. Нова версія програми зберігає всю попередню функціональність, в чому можна впевнитись за допомогою реалізованих раніше unit tests. Нові функції також працюють коректно.

7. Час виконання функцій не змінився (принаймні не суттєво), хоча обсяг коду і став значно більшим.

8. Проект має непогану структуру та дотримується багатьох принципів якісного проєктування. Основні компоненти розділені за відповідальністю, реалізована авторизація та аутентифікація, наявне тестування.

Основи ООП

- **Інкапсуляція** - моделі інкапсулюють дані та поведінку

(наприклад, userSchema.methods.matchPassword інкапсулює логіку порівняння паролів). Контролери інкапсулюють бізнес-логіку роботи з моделями. Пароль користувача позначений як select: false, що забезпечує додаткову інкапсуляцію чутливих даних. У ProductCard.jsx стан та логіка, пов'язана з редагуванням продукту, інкапсульовані всередині компонента.

- **Наслідування** - прямого наслідування класів не спостерігається, але це типово для Node.js/Express проектів, які часто використовують функціональний підхід або композицію замість класичного наслідування, проте можна зазначити, що використовується наслідування MongoDB Schema через Mongoose. Композиція також активно застосовується через поєднання компонентів Chakra UI та власних компонентів.
- **Поліморфізм** - у класичному розумінні обмежений, але middleware authorize демонструє елементи поліморфізму, приймаючи різні ролі як параметри.

Більш загальні

- **DRY** (Don't Repeat Yourself) - винесення логіки авторизації в middleware, що запобігає дублюванню, обробка помилок у кожному контролері. Проте присутнє деяке дублювання в frontend частині та тестових файлах.
- **KISS** (Keep It Simple, Stupid) - контролери мають зрозумілу відповідальність, чіткий маршрутизатор з простими URL-шляхами, прості й зрозумілі моделі даних.
- **YAGNI** (You Aren't Gonna Need It) - код не містить надлишкової функціональності, реалізовані лише необхідні функції.
- **SOLID**.

Більш конкретні об’єктно-орієнтовані

- **Cohesion** - контролери згруповані за функціональністю (auth.controller.js, product.controller.js), як і маршрути (user.route.js, product.route.js).
- **Law of Demeter** - функції в контролерах зазвичай взаємодіють з "найближчими сусідами" і не звертаються до внутрішніх деталей об'єктів.
- **Principle of Least Astonishment** - API-інтерфейси працюють передбачувано, з очікуваними URL-шляхами, використовуються стандартні HTTP-коди (200, 201, 400, 401, 404, 500) відповідно до їх призначення.

Патерни

В проаналізованому коді можна виділити наступні класичні GoF патерни:

- **Behavioral Patterns (Поведінкові патерни)**

1. **Chain of Responsibility** - через Express middleware. Запити проходять через ланцюжок обробників (middleware), кожен з яких має можливість обробити запит або передати його далі. Це є класичною реалізація даного патерну, де кожен обробник вирішує, чи обробляти запит або передати його далі по ланцюжку.

- **Structural Patterns (Структурні патерни)**

2. **Decorator** - через middleware для розширення функціональності маршрутів. Наприклад: `router.post("/", protect, authorize("admin"), createProduct);` Ці middleware "декорують" основний обробник маршруту (createProduct), додаючи до нього додаткову функціональність (перевірку аутентифікації та авторизації) без зміни його коду.
3. **Facade** - через контролери. Контролери представляють собою фасад, який спрощує доступ до складної підсистеми (взаємодія з базою даних, бізнес-логіка, обробка помилок).
4. **Proxy** - через middleware для контролю доступу. Middleware protect та authorize діють як проксі, які контролюють доступ до захищених маршрутів, тобто middleware контролює доступ до оригінального об'єкта (обробника маршруту).

- **Creational Patterns (Патерни створення)**

5. **Factory Method** - через створення моделей Mongoose. Наприклад: `const User = mongoose.model("User", userSchema);` - фабричний метод, який створює конкретні екземпляри моделей на основі заданої схеми.
6. **Singleton** - через підключення до бази даних. Наприклад: функція `connectDB`.
7. **Builder** - частково через Mongoose Schema. Схема поступово будується з різних полів та опцій, що нагадує патерн Builder.

У висновку можна сказати, що backend частина чудово дотримується принципів архітектури та проектування, проте зі сторони frontend ще присутні деякі проблеми. В backend частині проекту основні компоненти

розділені за відповідальністю, реалізована авторизація та аутентифікація, наявне тестування. З деяких проблем frontend можна виділити високий рівень зв'язування між компонентами та сховищами, відсутність чіткого розділення відповідальності та дублювання коду.