Звіт
Лабораторна робота №1а
“Рефакторінг на моделювання з використанням UML”
Виконала студентка групи ІПС-22
Чергінець Юлія Дмитрівна

1. Опис об'єкта дослідження

В якості об'єкта для рефакторінгу та моделювання було обрано веб-застосунок **"Weather Vibes"**.

- **Тип:** Single Page Application (SPA).
- **Стек технологій:** JavaScript (ES6+), React 19, Tailwind CSS 4, Vitest (тестування).
- **Призначення:** Моніторинг погодних умов, перегляд прогнозу, ведення списку обраних міст та аналіз статистики.

Початковий код мав високу зв'язність (High Coupling) у класі WeatherService, який відповідав і за збереження даних, і за логіку генерації погоди, і за взаємодію з UI, що порушувало принципи SOLID.

2. Моделювання системи (UML)

Для отримання оцінки **"Відмінно"** було розроблено та реалізовано **13 типів діаграм UML 2.5**. Для моделювання використано підхід *Diagram as Code* (інструмент **Mermaid.js**).

Структурні діаграми (Structure Diagrams)

1. **Class Diagram:** Відображає статичну структуру класів (WeatherService, CityManager, WeatherFactory) та зв'язки між ними.
2. **Component Diagram:** Показує поділ системи на логічні компоненти (UI Components, Logic Layer, Data Layer).
3. **Deployment Diagram:** Описує розгортання застосунку (Client Browser, GitHub Pages Host).
4. **Object Diagram:** Демонструє знімок стану системи в конкретний момент часу (інстанси об'єктів).
5. **Package Diagram:** Відображає фізичну структуру папок та залежності між модулями (src, components, logic).
6. **Composite Structure Diagram:** Деталізує внутрішню структуру класу WeatherService (порти та внутрішні частини).

Поведінкові діаграми (Behavior Diagrams)

7. **Use Case Diagram:** Описує сценарії взаємодії користувача з системою (Перегляд погоди, Додавання міста).

8. **Sequence Diagram:** Деталізує послідовність викликів методів при оновленні погоди.
9. **Activity Diagram:** Алгоритм роботи методу оновлення даних.
10. **State Machine Diagram:** Життєвий цикл об'єкта WeatherCondition (Creating -> Validating -> Displayed).
11. **Communication Diagram:** Альтернатива діаграмі послідовності, акцентує увагу на зв'язках між об'єктами.
12. **Timing Diagram:** Зміна стану об'єкта WeatherData у часі (хронологія виконання запиту).
13. **Interaction Overview Diagram:** Високорівневий огляд взаємодії, що поєднує Activity та Sequence.

*Усі діаграми знаходяться в папці **/uml** репозиторію.*

3. Рефакторінг та Патерни проєктування

Для покращення архітектури було запропоновано та реалізовано наступні зміни:

3.1. Патерн Singleton (Одинак)

- **Проблема:** Список "Улюблених міст" скидався при перезавантаженні компонентів або створенні нових екземплярів сервісу.
- **Рішення:** Реалізовано клас CityManager як Singleton.
- **Результат:** Гарантується існування єдиного списку міст у пам'яті програми. Покращено управління станом (State Management).

3.2. Патерн Factory Method (Фабричний метод)

- **Проблема:** Клас WeatherService містив жорстку логіку створення типів погоди через if/else та пряме створення об'єктів (new SunnyCondition), що порушувало принцип Open/Closed.
- **Рішення:** Створено клас WeatherFactory та абстракцію WeatherCondition.
- **Результат:** WeatherService більше не залежить від конкретних класів погоди. Додавання нового типу погоди (наприклад, снігу) не вимагає зміни коду сервісу.

3.3. Архітектурні зміни (Layered Architecture)

- **Проблема:** Початковий код був монолітним, де логіка обробки даних перемішувалася з відображенням (UI).
- **Рішення:** Реалізовано чіткий поділ на компоненти (шари):
    1. **Presentation Layer:** React-компоненти (`/components`) — відповідають лише за відображення.
    2. **Business Logic Layer:** Сервіси та фабрики (`/logic`) — обробка даних.
    3. **Data Layer:** Менеджери даних (`/data`) — зберігання стану.
- **Результат:** Зменшено зв'язність (Coupling), що дозволяє змінювати логіку без впливу на інтерфейс.

4. Порівняльний аналіз метрик

Метрика	**До рефакторінгу (AS IS)**	**Після рефакторінгу (TO BE)**	**Коментар**
Кількість класів	5	7	Зросла через декомпозицію (Factory, Interface).
Coupling (Зв'язність)	Висока	Низька	Service відокремлено від логіки створення об'єктів.
Cyclomatic Complexity	Середня	Низька	Умовні конструкції винесено у Фабрику.

Тестопридатність	Низька	Висока	Можливість мокати фабрику та менеджер міст.

5. Аналіз дотримання принципів SOLID

1. **SRP (Single Responsibility):** Розділено відповідальність. CityManager керує містами, WeatherFactory створює погоду, WeatherService керує потоком даних.
2. **OCP (Open/Closed):** Система відкрита для розширення (нові типи погоди), але закрита для модифікації (код сервісу не змінюється).
3. **DIP (Dependency Inversion):** Високорівневий сервіс залежить від абстракцій, а не від конкретних реалізацій.

6. Тестування (Unit Tests)

Реалізовано набір модульних тестів з використанням фреймворку **Vitest**:

1. testWeather.js — перевірка базової логіки класів.
2. testRefactoring.js — перевірка коректності роботи Singleton (єдиний екземпляр) та Factory Method (правильні типи об'єктів).

Всі тести проходять успішно, що підтверджує: рефакторінг не порушив бізнес-логіку застосунку.

7. Документація та Глосарій

- **Глосарій:** Створено файл **glossary.md** з описом предметної області та технічних термінів.
- **JSDoc:** Код документовано згідно зі стандартом JSDoc.
- **GitHub Pages:** Згенеровано HTML-документацію (тема Docdash) та розгорнуто на GitHub Pages.

8. Висновки

В ході виконання лабораторної роботи було проведено аналіз та рефакторінг JavaScript-застосунку. Використання патернів **Singleton**

та **Factory Method** дозволило зменшити зв'язність коду та покращити його розширюваність. Створено повний набір UML-діаграм (13 типів), що повністю описують архітектуру системи. Налаштовано автоматичну генерацію документації та середовище тестування.