Національний технічний університет України
«Київський політехнічний інститут імені Ігоря Сікорського»
Факультет інформатики та обчислювальної техніки
Кафедра інформаційних систем та технологій

Лабораторна робота №7

з дисципліни <<Технології розробки програмного забезпечення>>

Тема <<ШАБЛОН «MEDIATOR», «FACADE», «BRIDGE», «TEMPLATE METHOD»>>

Виконав:
студент ІА-23
Содолиснький Вадим

Перевірив:
Мягкий М. Ю.

Київ 2024

Тема: ШАБЛОН «MEDIATOR», «FACADE», «BRIDGE», «TEMPLATE METHOD»

Хід роботи

Патерн Міст (Bridge)

Патерн "Міст" (Bridge) — це структурний патерн проєктування, який розділяє абстракцію та її реалізацію так, щоб вони могли змінюватися незалежно одна від одної. Це досягається за допомогою створення мосту між абстракцією та реалізацією через окремий інтерфейс.

Переваги

Уникає експоненційного зростання кількості класів, яке виникає через комбінації варіантів абстракції та реалізації. Замість створення класів для кожної комбінації, достатньо розділити їх логіку на окремі модулі.

Проблема

Результат відповіді від сервера - це об'єкти різних класів і щоб їх обробити потрібно окремі класи, що викликає велику кількість класів, які виконують одну функцію.

Рішення

Створено один клас ServerResponse, який містить поле з об'єктом відповіді. Таким чином замість створення великої кількість класів створюється один, який зберігає об'єкт в залежності від відповіді сервера.

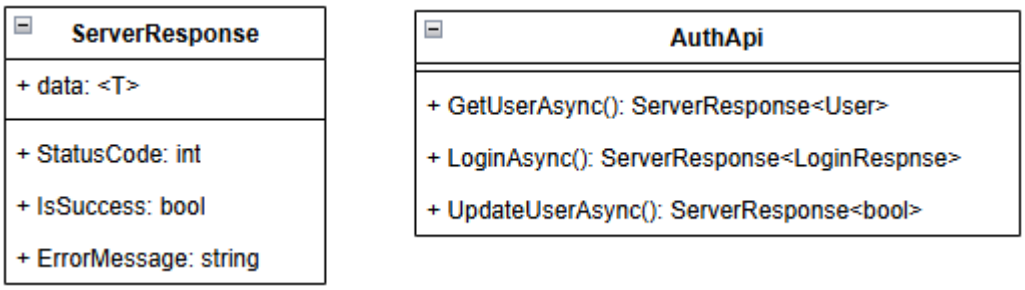

Рис. 1 – Реалізація патерну Bridge

```
public class ServerResponse<T>
{
    59 references
    public T? Data { get; set; }
    6 references
    public int StatusCode { get; set; }
    86 references
    public bool IsSuccess { get; set; }
    10 references
    public string? ErrorMessage { get; set; }

    19 references
    public ServerResponse(HttpResponseMessage response)
    {
        if(Data is not bool) Data = response.Content.ReadFromJsonAsync<T>().Result;
        StatusCode = ((int)response.StatusCode);
        IsSuccess = response.IsSuccessStatusCode;
        ErrorMessage = response.ReasonPhrase;
    }

    22 references
    public ServerResponse(T? data, int statusCode, bool isSuccess, string? errorMessage)
    {
        Data = data;
        StatusCode = statusCode;
        IsSuccess = isSuccess;
        ErrorMessage = errorMessage;
    }
}
```

Рис. 2 – Клас ServerResponse

```
public async Task<ServerResponse<User>> GetUserAsync(string token)
{
    _httpClient.DefaultRequestHeaders.Authorization = new AuthenticationHeaderValue("Bearer", token);
    try
    {
        var response = await _httpClient.GetAsync(_url + "/get");
        return new ServerResponse<User>(response);
    }
    catch (Exception e)
    {
        return new ServerResponse<User>(null, 500, false, e.Message);
    }
}
```

Рис. 3 – Застосуваванння класу ServerResponse для обробки відповіді сервера

Висновок: я використав патерн ServerResponse для уніфікації обробки відповіді від сервера.