УДК 621.326
Шульга А.
Тернопільський національний технічний університет імені Івана Пулюя

ВИКОРИСТАННЯ ПАТЕРНУ MEDIATOR (MEDIATR) У WPF‑ ЗАСТОСУНКАХ ДЛЯ ОБЛІКУ ТА ПЛАНУВАННЯ ФІНАНСІВ

Науковий керівник: д.ф.-м.н., професор Петрик Михайло Романович

Shulha A.
Ternopil Ivan Puluj National Technical Univercity

USING THE MEDIATOR PATTERN (MEDIATR) IN WPF APPLICATIONS FOR FINANCE TRACKING AND PLANNING

Функції персонального фінансового обліку зазвичай швидко ростуть від простого введення витрат до аналітики (підсумки за період, розподіл за категоріями, відстеження лімітів), що збільшує кількість взаємодій між UI, правилами обробки даних і сховищем. Якщо ці взаємодії залишити «розсипаними» по ViewModel, проєкт стає складнішим у супроводі: кожне розширення функціональності створює нові залежності й точки можливої помилки.

У WPF‑ архітектурі на базі MVVM ViewModel часто виступає координатором дій користувача. Ризик полягає в тому, що ViewModel починає накопичувати не лише стан екрана, а й «важкі» обчислення, правила валідації та безпосередню роботу з БД. Наслідки типові: по‑ перше, складніше змінювати логіку, не зачіпаючи UI; по‑ друге, важче тестувати бізнес‑ сценарії без запуску інтерфейсу; по‑ третє, зростає кількість прихованих залежностей між екранами (наприклад, коли різні ViewModel напряму викликають одні й ті самі сервіси даних).

Патерн Mediator («Посередник») описує підхід, за якого взаємодії між об’єктами переносяться в один компонент‑ посередник, щоб зменшити хаотичні залежності між класами [2]. У .NET це зручно реалізувати через MediatR, який підтримує сценарій request/response (команди й запити), а також забезпечує пошук і виклик відповідного handler‑ а за типом запиту [1].

У WPF‑ застосунку архітектурний «ланцюжок» виглядає так: ViewModel → IMediator.Send(...) → IRequest<T> → IRequestHandler<TRequest,TResponse> [1]. Практична інтерпретація для задач обліку фінансів включає команди (зміна даних): AddExpenseCommand, DeleteExpenseCommand, UpdateBudgetLimitCommand — повертають, наприклад, Unit або ідентифікатор створеного запису та запити (читання/аналітика): GetMonthlyTotalsQuery, GetCategoryBreakdownQuery — повертають агреговані значення або DTO для графіків і звітів.

MediatR інтегрується через стандартний контейнер залежностей .NET: сервіси реєструються у IServiceCollection, а потім резолвляться через IServiceProvider. Microsoft підкреслює, що DI в .NET базується саме на реєстрації сервісів у колекції та подальшому отриманні їх через контейнер, що напряму підтримує ін’єкцію залежностей у конструктор [3]. На практиці це означає: ViewModel отримує IMediator, handler отримує FinanceDbContext та інші сервіси, а композиція проєкту залишається централізованою й прозорою.

Представлення сценаріїв обліку фінансів як окремих request/handler дає відчутний «організаційний» ефект: функціональність структурується за діями, а не за

екранами, і кожен сценарій має чітку точку входу та відповідальність. Патерн «Посередник» зменшує прямі залежності, а MediatR у .NET надає для цього готові контракти й механізм диспетчеризації запитів [2]. Для даних, що зберігаються в SQLite через EF Core, це додатково означає, що бізнес‑ логіка читається/тестується в одному місці (handler), а UI не «заражається» деталями збереження [4].

Література:

1. MediatR (офіційний репозиторій). GitHub. URL: https://Https://github.com/LuckyPennySoftware/MediatR.
2. Посередник (Mediator). Refactoring Guru. URL: https://refactoring.guru/uk/design-patterns/mediator.
3. Dependency injection in .NET. Microsoft Learn. URL: https://learn.microsoft.com/en-us/dotnet/core/extensions/dependency-injection/overview.
4. .NET dependency injection. Microsoft Learn. URL: https://learn.microsoft.com/en-us/dotnet/core/extensions/dependency-injection/overview.