Кравчук О. С., студент 3 курсу,
Житомирський державний технологічний університет

ВИКОРИСТАННЯ ПРИНЦИПІВ ІМПЕРАТИВНОГО ПРОГРАМУВАННЯ В АРХІТЕКТУРНИХ ШАБЛОНАХ

Використання предметно-орієнтованого підходу, під час проектування додатку, надає змогу використовувати індивідуальні типи архітектури в залежності від предметної області застосування та проблем, які повинен вирішувати додаток.

До поширених проблем, які виникають під час розробки додатків, можна віднести проблему сукупностей, що перетинаються, та проблему незбалансованих обов'язків в додатку. Проблема сукупностей, що перетинаються, полягає в необхідності представлення як єдиного цілого даних, які зберігаються в різних агрегатних коренях. Можливими наслідками проблеми є порушення інкапсуляції або обмеження можливостей варіантів представлення даних. Проблема незбалансованих обов'язків характерна для додатків, в яких існує значна невідповідність між кількістю запитів спрямованих на зчитування даних та кількістю запитів (команд) спрямованих на їх модифікацію або обробку.

Розділення обов'язків між виконанням запитів та команд - принцип імперативного програмування, винайдений Бертраном Мейером. Головна ідея принципу полягає в наступному: метод повинен бути командою, яка виконує якусь дію, або запитом, який повертає дані, але не одночасно. Принцип набув застосування в шаблоні проектування CQRS (command-query responsibility segregation). Наслідком використання принципу став розподіл моделі додатку на дві частини - domain (домен, або предметна область) та read model (модель для зчитування), які, відповідно, призначені для опрацювання запитів на модифікацію даних, або команд (commands), та запитів на зчитування даних (queries). Для синхронізації відповідних змін в моделях використовуються події (events) в середині самого додатку.

Таким чином, використання CQRS вирішує проблему сукупностей, що перетинаються, шляхом використання read model, в якій представлення даних можуть формуватися та оновлюватися без порушень інкапсуляції. Проблема незбалансованих обов'язків вирішується можливістю оптимізації роботи domain або read моделі в залежності від вимог додатку.

Розширенним використанням разом з CQRS є використання менеджера подій (Event Sourcing - ES). Додавання шаблону проектування ES до додатку надає змогу зберігати об'єкти domain не в вигляді кінцевого результату, а в вигляді списку всіх важливих подій які мали вплив на дані об'єкту.

Сумарне використання CQRS та ES забезпечує цілий ряд переваг, зокрема:

1. Легка можливість горизонтального масштабування додатку. Велика кількість складових компонентів шаблону та комунікація між ними типу передачі повідомлень (команд, запитів або подій) надає можливіть легкого розподілення системи між різними фізичними складовими.
2. Висока доступність додатку. Оскільки додаток розподілений на частини, кожна частина може продовжувати функціонувати в умовах відсутності або виходу з ладу будь-якої іншої частини.
3. Аудит. Збереження списку всіх подій надає змогу відновити стан об'єктів в будь-який момент часу.
4. Відсутність обмеження використання типів для баз даних. Дані можуть зберігатися як в реляційних базах даних, так і в базах даних типу NoSQL. Рекомендується використовувати реляційні бази даних в тих випадках, коли вимагаються елементи звітності від системи.

Використання CQRS вносить додаткову складність в архітектуру додатку. Завдяки своїй структурі, шаблон є складним для впровадження, проте надає системі гнучкості та можливості легкого розширювання функціоналу. Також, до недолів даного архітектурного рішення необхідно додати можливість ускладнення створення реентабельного та багатопоточного програмного забезпечення. Зазвичай, ця проблема виникає при використанні не багатонитево-безпечного шаблону для реалізації CQRS.

Використання шаблону CQRS надає значні переваги в вирішенні складних архітектурних пролем під час розробки високонавантажених та широкомасштабних додатків. Проте, як і більшість архітектурних рішень, CQRS призначений для вирішення визначених проблем. Шаблон рекомендується використовувати у випадках:

1. Необхідності опрацюванням додатку складної бізнес-логіки. CQRS змушує не змішувати логіку предметної області та інфраструктурних операцій.
2. Необхідності наявності умов легкого масштабування системи.
3. Розробки додатку в складі великої кількості розробників. Велика кількість складових компонентів та їх розділеність надає змогу розробникам виконувати розробку незалежно.