

АНАЛІЗ РОЗРОБКИ ДОМЕННОЇ ЧАСТИНИ В ПАРАДИГМІ ООП

Дерменжи К. В.

Національний університет «Одеська юридична академія»

Україна, Одеса

dermenzhi.kostya@gmail.com

Анотація. Програмні системи мають базуватися на добре розробленій моделі домену. Проєкти зі складною бізнес-логікою потребують від розробника розуміння процесів, що потребують автоматизації. В роботі проаналізовано різні доменні моделі.

Ключові слова: доменна модель; активна модель; анемічна модель домену; багата модель домену.

Вступна частина. Розробка доменної частини застосунку є важливою задачею для розробника, адже саме ця частина відображає предметну область і дані, з якими працює застосунок. При цьому далеко не кожен застосунок потребує доменної частини. Коли йдеться про прості й нескладні завдання, створення окремого шару буде зайвим.

Мета роботи – аналіз популярних доменних моделей.

Основна частина. Під час розробки програмного забезпечення поняття логіки домену або доменного рівня використовують для опису бізнес-логіки, тобто моделювання взаємодії бізнес-об'єктів програми. Є різні погляди на розробку доменної моделі.

Активна модель (Active Record) використовується для невеликих і середніх застосунків, завдяки своїй простоті та інтеграції. Вона використовується у багатьох фреймворках як частина архітектури [1]. Суть побудови цієї моделі полягає в тому, що вона містить як дані, так і елементи бізнес-логіки. Маючи доступ до бази даних, вона працює з нею особисто. Така модель не має чітких вимог до інкапсуляції. Перевагами моделі є спрощення роботи з базою даних та ORM (Object-Relational Mapping), а недоліками — перенасичення моделі, складність підтримки проєкту при зростанні кодової бази, порушення принципу єдиної відповідальності [2].

Анемічна модель домену (Anemic Domain Model) добре проявляє себе в невеликих і середніх застосунках, але дещо програє у великих. Суть її в тому, що вона не має жодної бізнес-логіки та ігнорує інкапсуляцію. Тобто анемічна модель є набором даних, якими вільно може користуватися або змінювати клієнтський код. Вся бізнес-логіка та взаємодія з базою даних застосунку при такому підході віддається в руки інших частин застосунку [3]. Перевагами є простота, хороша інтеграція з різними ORM та висока швидкість розробки. Недоліками є порушення принципів ООП, погана стабільність і безпека, особливо у великих проєктах, де ймовірність припуститись помилки через відсутність інкапсуляції є великою.

Багата модель домену (Rich Domain Model) є протилежністю анемічної моделі. По-перше, багаті моделі містять бізнес-логіку, по-друге, вони інкапсульовані та керують своїм станом і поведінкою, абстрагуючись від бази даних і всього, що не стосується предметної області [4]. Переважно такі моделі використовують у контексті предметно-орієнтованого програмування, в якому наріжним каменем застосунку є саме доменна частина, а інші частини застосунку орієнтовані на неї. У свою чергу, багаті моделі поділяють на три види: агрегатні корені, сутності та значущі об'єкти [5].

Значущий об'єкт (Aggregate Root) — проста, переважно незмінна модель, яка відображає певне значення з предметної області та інкапсулює логіку його створення і взаємодії. Він не має унікального ідентифікатора і здебільшого є частиною сутності або агрегатного кореня.

Сутність (Entity) — комплексна модель, яка переважно складається з значущих об'єктів і має унікальний ідентифікатор. Вона може містити в собі більш складну логіку, ніж значущі об'єкти, і є частиною агрегатного кореня.

Агрегатний корінь (Value Objects) – основна сутність в агрегаті, що є вхідною точкою для доступу та зміни сутностей, що належать до цього агрегату. Агрегат сам по собі є сукупністю пов'язаних сутностей та значимих об'єктів, які розглядаються як єдине ціле для управління та цілісності даних. Тобто всі зміни сутностей можуть відбуватися лише через корінь.

Рішення про те, які сутності повинні бути призначені агрегатними коренями, є одним із найскладніших і найважливіших. Агрегатним коренем варто призначати ту сутність, без якої всі частини цього агрегату не мають особливого сенсу і не можуть існувати окремо від неї. Для наочності можна навести приклад бізнесу з бронювання залів. Припустимо, що у залу є певне обладнання, наприклад: проєктор, інтернет тощо. У такому випадку модель залу міститиме колекцію обладнання. Виходячи з предметної області в контексті цього бізнесу обладнання не має сенсу без залу і не може існувати окремо від нього, отже, зал є агрегатним коренем відносно сутності обладнання.

Перевагами багатой моделі є: висока стабільність, гарна підтримуваність і масштабованість, що особливо важливо для великих проєктів. Недоліками є: складність цього підходу, адже він вимагає хороших знань самої концепції та предметної області; уповільнена швидкість розробки, особливо на етапі проєктування, що робить цей підхід не найкращим вибором для невеликого проєкту або у разі, коли швидкість розробки є критично важливою.

Висновки. Розробка доменної частини застосунку є важливим етапом, який часто впливає на архітектуру застосунку. Проведений аналіз показав, що для масштабних, великих проєктів кращим кандидатом є багата модель, оскільки вона здатна забезпечити високу стабільність, безпеку та масштабованість. Для малих та середніх проєктів краще надати перевагу анемічній або активній моделі, залежно від обставин. Якщо потрібно спростити роботу з базою даних, варто вибрати активну модель, а якщо важливою є швидкість, наприклад, для створення мінімально життєздатного продукту, краще вибрати анемічну модель.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

- [1] Smith, K. (2023). *How to Produce a Rich Domain Model with Active Record*. <https://kevinsmith.io/how-to-produce-a-rich-domain-model-with-active-record/>
- [2] Hanselman, S., Shifflet, D. (2021). *Implementing a Domain with POCO*. <https://blog.ndepend.com/implementing-a-domain-with-poco-plain-old-clr-objects/>
- [3] Schenk, M. (2023). *Anemic Domain Model vs. Rich Domain Model*. <https://medium.com/@inzuael/anemic-domain-model-vs-rich-domain-model-78752b46098f>
- [4] Fowler, M. (2020). *Domain Driven Design*. <https://martinfowler.com/bliki/DomainDrivenDesign.html>
- [5] Paulovich, I. (2018). *Rich Domain Model with DDD/TDD (Reviewed)*. <https://paulovich.net/rich-domain-model-with-ddd-tdd-reviewed/>

Dermenzhi Konstantin

Analysis of the development of the domain part in the oop paradigm

Abstract. Software systems should be based on a well-developed domain model. Projects with complex business logic require the developer to understand processes that require automation. Different domain models are analyzed in the work..

Keywords: domain model; active record; anemic domain model; rich domain model.