УДК 658.89:004.423
DOI https://doi.org/10.32782/tnv-tech.2025.1.19

DOMAIN DRIVEN DESIGN ЯК ОСНОВА ДЛЯ МОДЕЛЮВАННЯ МІКРОСЕРВІСІВ В ЕЛЕКТРОННІЙ КОМЕРЦІЇ

Сікора Р. В. *– аспірант кафедри цифрової економіки та системного аналізу*
Державного торговельно-економічного університету
ORCID ID: 0009-0002-7570-9826

У сучасному світі електронної комерції, де конкуренція постійно зростає, ефективне моделювання програмних систем є ключовим фактором успіху. Одним з найперспективніших підходів до вирішення цієї проблеми є Domain Driven Design (DDD), який дозволяє розробникам створювати програмні системи, що точно відображають бізнес-логіку та легко адаптуються до змін ринку.

DDD пропонує розробникам зосередитися на предметній області бізнесу, створюючи спільну мову (Ubiquitous Language) між розробниками та експертами з предметної області. Це дозволяє уникнути непорозумінь та забезпечує точне відображення бізнес-вимог у коді.

DDD включає в себе вісім ключових концепцій: Domain Driven Design, Business Entities, Model Boundaries, Aggregation, Entities vs. Value Objects, Operational Modeling, Layering the architecture, Build the domain model.

Одним з ключових понять DDD є агрегати – групи об'єктів, що обробляються як єдине ціле. Агрегати забезпечують інкапсуляцію та цілісність даних, що є особливо важливим при розробці мікросервісів.

Мікросервіси – це архітектурний стиль, що передбачає розробку програмних систем як набору невеликих, незалежних сервісів. DDD ідеально підходить для моделювання мікросервісів, оскільки дозволяє розробникам розбити складну систему на невеликі, керовані частини, кожна з яких відповідає за певну бізнес-функцію.

Застосування DDD в електронній комерції може значно покращити якість програмного забезпечення, зменшити час розробки та підвищити задоволеність клієнтів. Завдяки DDD, компанії можуть створювати більш гнучкі та адаптивні системи, що дозволяє їм швидко реагувати на зміни ринку та потреби клієнтів.

В статті розглянуто основні принципи DDD та їх застосування для моделювання мікросервісів в електронній комерції. Показано, як DDD може допомогти розробникам створювати більш ефективні та масштабовані системи, що відповідають потребам бізнесу.

Ключові слова: *Domain Driven Design, DDD, мікросервіси, електронна комерція, моделювання.*

Sikora R. V. Domain Driven Design as a basis for microservice modeling in e-commerce

In today's world of e-commerce, where competition is constantly increasing, effective software system modeling is a key success factor. One of the most promising approaches to solving this problem is Domain Driven Design (DDD), which allows developers to create software systems that accurately reflect business logic and easily adapt to market changes.

DDD encourages developers to focus on the business domain by creating a common language (Ubiquitous Language) between developers and domain experts. This helps to avoid misunderstandings and ensures that business requirements are accurately reflected in the code.

DDD includes eight key concepts: Domain Driven Design, Business Entities, Model Boundaries, Aggregation, Entities vs. Value Objects, Operational Modeling, Layering the architecture, Build the domain model.

One of the key concepts of DDD is aggregates – groups of objects that are treated as a single unit. Aggregates provide encapsulation and data integrity, which is especially important when developing microservices.

Microservices is an architectural style that involves developing software systems as a set of small, independent services. DDD is ideal for modeling microservices because it allows developers to break down a complex system into small, manageable parts, each of which is responsible for a specific business function.

Applying DDD in e-commerce can significantly improve software quality, reduce development time, and increase customer satisfaction. With DDD, companies can create more flexible and adaptive systems, allowing them to respond quickly to market changes and customer needs.

This article discusses the basic principles of DDD and their application for modeling microservices in e-commerce. It shows how DDD can help developers create more efficient and scalable systems that meet business needs.

Key words: *Domain Driven Design, DDD, microservices, e-commerce, modeling.*

Вступ. У сучасному динамічному середовищі електронної комерції, бізнеси стикаються з постійно зростаючим тиском швидко адаптуватися до мінливих ринкових тенденцій та очікувань клієнтів. Традиційні монолітні архітектури часто не встигають за цими вимогами, що призводить до тривалих циклів розробки, вузьких місць масштабованості та труднощів у впровадженні нових функцій. Це стимулювало появу мікросервісів, архітектурного стилю, який розбиває додатки на невеликі, незалежні сервіси, що можуть розроблятися, розгортатися та масштабуватися автономно. Однак ефективне моделювання та впровадження мікросервісів створює власний набір проблем. Саме тут Domain-Driven Design (DDD) виступає як потужний підхід до керування проектуванням та розробкою систем електронної комерції на основі мікросервісів.

DDD наголошує на глибокому розумінні бізнес-домену та узгоджує дизайн програмного забезпечення з основними бізнес-концепціями та процесами. Сприяючи співпраці між розробниками та експертами з предметної області, DDD встановлює універсальну мову, яка долає розрив між технічною реалізацією та бізнес-логікою. Це спільне розуміння гарантує, що програмне забезпечення точно відображає бізнес-потреби та може легко адаптуватися до змін у домені.

Ця стаття заглиблюється в ключові концепції DDD та ілюструє, як їх можна використовувати для ефективного моделювання мікросервісів у контексті електронної комерції (див. рис. 1). Ми розглянемо переваги прийняття DDD та надамо практичні рекомендації щодо його впровадження.

Ключові концепції Domain-Driven Design

DDD обертається навколо кількох основних концепцій, які керують процесом проектування та розробки. Ці концепції забезпечують основу для розуміння бізнес-домену, моделювання його складнощів та перетворення їх у надійну архітектуру програмного забезпечення.

1. Domain-Driven Design

Domain-Driven Design (DDD) – це не просто набір інструментів чи шаблонів, це скоріше філософія, спосіб мислення про розробку програмного забезпечення, який ставить бізнес-домен в центр уваги. DDD закликає до глибокого занурення в предметну область, до розуміння її складнощів та нюансів. Це означає, що структура та поведінка програмного забезпечення повинні бути максимально наближені до реальних процесів та концепцій, що існують в бізнес-домені.

DDD вимагає тісної співпраці між розробниками та експертами з предметної області. Розробники повинні не просто писати код, а й розуміти, як цей код відображає бізнес-процеси. Експерти з предметної області, в свою чергу, повинні брати активну участь у процесі розробки, ділячись своїми знаннями та досвідом. Така співпраця дозволяє створити програмне забезпечення, яке є не просто набором функцій, а й точним відображенням бізнес-реальності. DDD допомагає розробникам створювати програмне забезпечення, яке є більш гнучким, масштабованим та легким в обслуговуванні. Це досягається шляхом розбиття складної системи на менші, більш керовані частини, кожна з яких відповідає за певну бізнес-функцію.

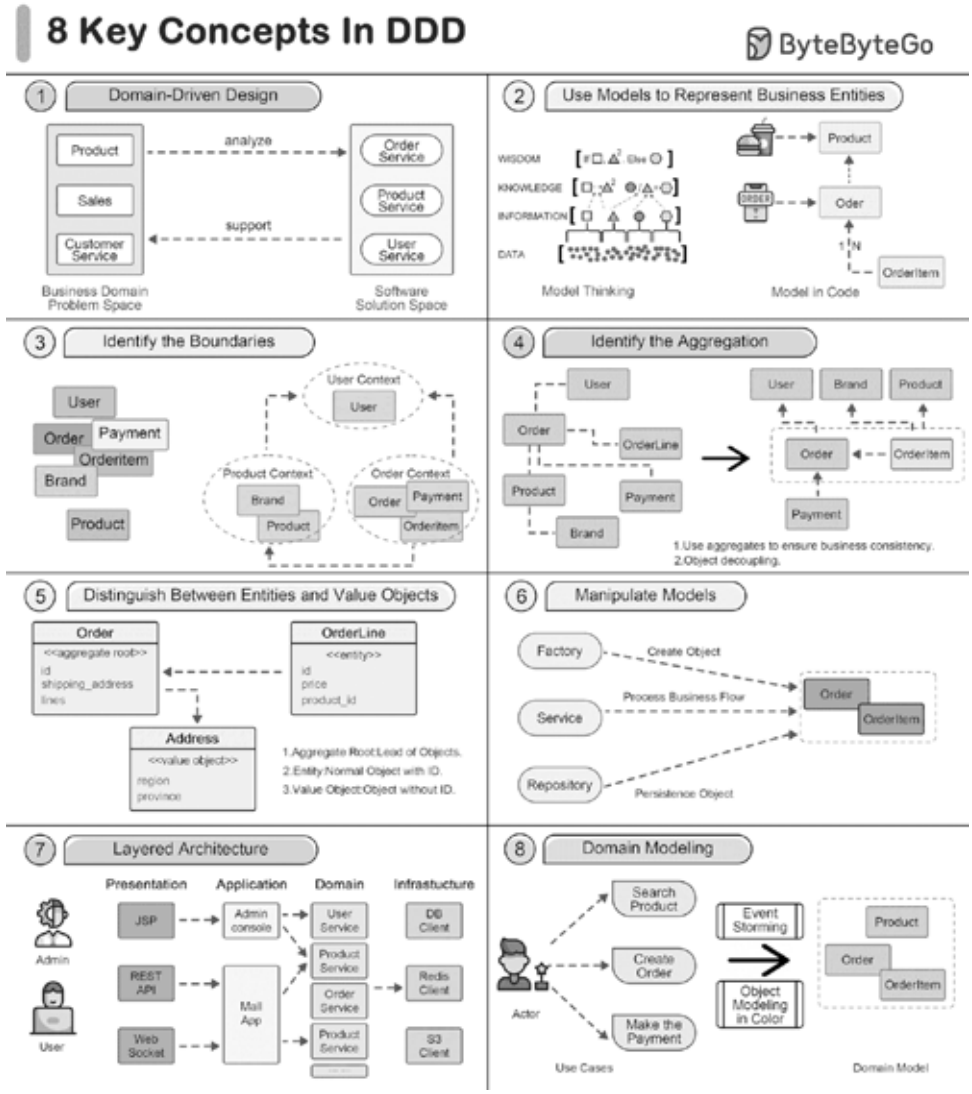

Рис. 1. 8 ключових понять у DDD

Джерело: https://blog.bytebytego.com

Як зазначає Ерік Еванс у своїй основоположній книзі: «Програмне забезпечення, яке не враховує особливості домену, приречене на невдачу.» [1] Він підкреслює, що успіх проекту залежить від того, наскільки добре розробники розуміють бізнес-домен та наскільки точно вони відображають його в коді.

2. Універсальна мова (Ubiquitous Language)

Універсальна мова – це один з ключових елементів DDD, який служить мостом між розробниками та експертами з предметної області. Це не просто набір термінів, це спільний словник, який використовується для опису бізнес-домену. Універсальна мова повинна бути зрозумілою як для розробників, так і для

бізнес-експертів. Вона повинна бути точною, лаконічною та відображати бізнес-реальність.

Універсальна мова повинна використовуватися в усіх аспектах розробки програмного забезпечення, від документації до коду. Вона повинна бути присутня в назвах класів, методів, змінних, в коментарях до коду, в діаграмах та в будь-яких інших артефактах проекту. Створення універсальної мови – це ітеративний процес, який вимагає постійної співпраці між розробниками та експертами з предметної області.

Універсальна мова допомагає уникнути непорозумінь та двозначності, що може призвести до помилок в програмному забезпеченні. Вона також сприяє кращому розумінню бізнес-домену розробниками, що дозволяє їм створювати більш якісне та ефективне програмне забезпечення.

3. Сутності (Entities) та Об'єкти-значення (Value Objects)

DDD пропонує чітке розмежування між двома основними типами об'єктів домену: сутностями та об'єктами-значеннями. Сутності – це об'єкти, які мають унікальну ідентичність, яка зберігається протягом усього їх життєвого циклу. Ідентичність сутності не залежить від її атрибутів. Наприклад, два клієнти з однаковим ім'ям та прізвищем все одно будуть різними сутностями, оскільки вони мають різні ідентифікатори.

Об'єкти-значення, навпаки, визначаються виключно їх атрибутами. Вони не мають власної ідентичності. Наприклад, дві адреси з однаковими значеннями вулиці, будинку та квартири будуть вважатися одним і тим же об'єктом-значенням.

Розрізнення між сутностями та об'єктами-значеннями є важливим аспектом DDD, оскільки воно допомагає розробникам створювати більш точні та ефективні моделі домену. Правильне визначення сутностей та об'єктів-значень дозволяє уникнути дублювання даних та забезпечує цілісність даних.

4. Агрегати (Aggregates)

Агрегати – це один з найважливіших будівельних блоків DDD. Агрегат – це кластер пов'язаних сутностей та об'єктів-значень, які розглядаються як єдине ціле. Агрегат має кореневу сутність (aggregate root), яка є єдиною точкою доступу до агрегату.

Агрегати забезпечують інкапсуляцію та цілісність даних, гарантуючи, що зміни в одному об'єкті агрегату не порушують узгодженість інших об'єктів. Наприклад, «Замовлення» може бути агрегатом, що містить сутності «Товар» та «Клієнт», а також об'єкти-значення, такі як «Адреса доставки» та «Дата замовлення».

Агрегати допомагають розробникам створювати більш модульні та легкі в обслуговуванні системи. Вони також спрощують управління транзакціями та забезпечують узгодженість даних.

5. Сервіси (Services)

Сервіси в DDD – це об'єкти, які виконують певну бізнес-логіку, але не є частиною жодної сутності чи об'єкта-значення. Сервіси зазвичай представляють операції, які охоплюють кілька об'єктів домену. Наприклад, сервіс «Оплата» може обробляти транзакції між клієнтом, замовленням та платіжною системою.

Сервіси допомагають розробникам розділяти бізнес-логіку від об'єктів домену, що робить систему більш гнучкою та легкою в обслуговуванні. Вони також сприяють повторному використанню коду та спрощують тестування.

6. Репозиторії (Repositories)

Репозиторії в DDD – це механізми, які забезпечують абстракцію для доступу до даних. Вони дозволяють об'єктам домену взаємодіяти з базою даних без знання

деталей її реалізації. Репозиторії приховують складність доступу до даних, надаючи простий та зрозумілий інтерфейс для роботи з об'єктами домену.

Репозиторії спрощують тестування, дозволяючи легко замінити реальну базу даних на тестову. Вони також дозволяють легко змінювати спосіб зберігання даних без впливу на бізнес-логіку.

7. Фабрики (Factories)

Фабрики в DDD – це спеціальні об'єкти, які відповідають за створення складних об'єктів домену. Вони інкапсулюють логіку ініціалізації та налаштування об'єктів, забезпечуючи їх узгодженість та правильність створення.

Фабрики допомагають уникнути дублювання коду та спрощують створення об'єктів домену. Вони також сприяють розділенню відповідальності та покращують читабельність коду.

8. Шари архітектури (Layered Architecture)

DDD рекомендує розділяти додаток на шари, кожен з яких має свою чітко визначену відповідальність. Найпоширеніша модель шарів в DDD включає:

Шар представлення (Presentation Layer): Відповідає за взаємодію з користувачем, відображення даних та отримання вхідних даних.

Шар додатків (Application Layer): Координує роботу об'єктів домену та забезпечує виконання бізнес-логіки.

Шар домену (Domain Layer): Містить основну бізнес-логіку та об'єкти домену.

Шар інфраструктури (Infrastructure Layer): Забезпечує доступ до даних, взаємодію з зовнішніми системами та інші технічні функції.

Розділення додатку на шари покращує організацію коду, розділяє відповідальність та спрощує тестування. Воно також сприяє повторному використанню коду та дозволяє легко змінювати реалізацію окремих шарів без впливу на інші шари.

DDD та мікросервіси в електронній комерції

DDD та мікросервіси – це дві концепції, які чудово доповнюють одна одну, особливо в динамічному світі електронної комерції. DDD забезпечує основу для розуміння та моделювання складної бізнес-логіки, тоді як мікросервіси пропонують гнучку та масштабовану архітектуру для реалізації цієї логіки.

DDD ідеально підходить для моделювання мікросервісів, оскільки дозволяє розробникам розбити складну систему на невеликі, керовані частини, кожна з яких відповідає за певну бізнес-функцію. Це узгоджується з основними принципами мікросервісної архітектури: автономність, незалежність та спеціалізація.

Як зазначає Сем Ньюмен: «Однією з ключових переваг мікросервісів є те, що вони дозволяють розробникам розбивати великі, складні додатки на менші, більш керовані частини. Це робить розробку, тестування та розгортання програмного забезпечення набагато простішим» [2].

DDD допомагає визначити ці «менші, більш керовані частини» шляхом виявлення обмежених контекстів (Bounded Contexts) в бізнес-домені. Кожен обмежений контекст представляє собою окрему область бізнесу з власною моделлю домену та універсальною мовою. Це дозволяє розробникам створювати мікросервіси, які є автономними та незалежними, що спрощує їх розробку, тестування та розгортання.

Переваги використання DDD для моделювання мікросервісів в електронній комерції:

- **Краще розуміння бізнес-вимог:** DDD сприяє тісній співпраці між розробниками та експертами з предметної області, що забезпечує глибоке розуміння

бізнес-вимог та їх точне відображення в програмному забезпеченні. Це особливо важливо в електронній комерції, де бізнес-процеси можуть бути досить складними та динамічними.

- **Гнучкість та масштабованість:** Мікросервіси, змодельовані з використанням DDD, є невеликими, незалежними та легко масштабованими, що дозволяє системі адаптуватися до змін у навантаженні та бізнес-потреб. В електронній комерції, де трафік може значно коливатися, масштабованість є критично важливою.
- **Легкість у розробці та розгортанні:** Незалежність мікросервісів спрощує їх розробку, тестування та розгортання, що прискорює цикл розробки та дозволяє швидше реагувати на зміни ринку. В електронній комерції, де швидкість виходу на ринок є ключовим фактором успіху, це є значною перевагою.
- **Підвищена надійність та відмовостійкість:** Ізоляція мікросервісів обмежує вплив збоїв, підвищуючи надійність та відмовостійкість системи. В електронній комерції, де простої можуть призвести до значних фінансових втрат, надійність є надзвичайно важливою.

Приклади застосування DDD в електронній комерції:

- **Моделювання каталогу товарів:** DDD може бути використаний для моделювання каталогу товарів, враховуючи різні атрибути товарів, категорії, фільтри та пошук.
- **Моделювання процесу оформлення замовлення:** DDD може бути використаний для моделювання процесу оформлення замовлення, включаючи кошик, оплату, доставку та обробку замовлення.
- **Моделювання системи рекомендацій:** DDD може бути використаний для моделювання системи рекомендацій, яка аналізує поведінку користувачів та пропонує їм релевантні товари.

Висновки. Domain-Driven Design (DDD) є не просто методологією розробки програмного забезпечення, а скоріше філософією, яка ставить бізнес-домен в центр уваги. DDD закликає до глибокого розуміння предметної області, до виявлення її складнощів та нюансів, та до відображення цих знань в коді. Це підхід, який вимагає тісної співпраці між розробниками та експертами з предметної області, що дозволяє створити спільне розуміння бізнес-процесів та термінології.

В сучасному світі електронної комерції, де конкуренція постійно зростає, а вимоги до програмного забезпечення стають все більш складними, DDD є незамінним інструментом для створення гнучких, масштабованих та надійних систем. DDD дозволяє розробникам розбити складну систему на менші, більш керовані частини, кожна з яких відповідає за певну бізнес-функцію. Це спрощує розробку, тестування та розгортання програмного забезпечення, а також дозволяє швидше реагувати на зміни ринку.

Мікросервіси, в свою чергу, є архітектурним стилем, який ідеально підходить для реалізації DDD. Мікросервіси – це невеликі, незалежні сервіси, які можуть розроблятися, розгортатися та масштабуватися автономно. Це дозволяє створювати більш гнучкі та масштабовані системи, які можуть легко адаптуватися до змін у бізнес-потребах.

Поєднання DDD та мікросервісів є потужним інструментом для створення сучасних систем електронної комерції. DDD забезпечує основу для розуміння та моделювання складної бізнес-логіки, тоді як мікросервіси пропонують гнучку та масштабовану архітектуру для реалізації цієї логіки.

В цілому, DDD та мікросервіси є важливими інструментами для будь-якої компанії, яка займається електронною комерцією. Вони дозволяють створювати сучасні, гнучкі та масштабовані системи, які можуть допомогти компанії досягти успіху в конкурентному середовищі.

СПИСОК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ:

1. Еванс, Е. Проектування, орієнтоване на домен: боротьба з складністю в серці програмного забезпечення. Addison-Wesley, 2003. 560 с.
2. Ньюмен, С. Мікросервіси: проектування тонкозернистої системи. O’Reilly Media, 2015. 272 с.

REFERENCES:

1. Evans, E. (2003) Domain-Driven Design: Tackling Complexity in the Heart of Software. Addison-Wesley, 560 p.
2. Newman, S. (2015) Building Microservices: Designing Fine-Grained Systems. O’Reilly Media, 272 p.