МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

Харківський національний університет радіоелектроніки

Центр післядипломної освіти

Кафедра Програмної інженерії

КВАЛІФІКАЦІЙНА РОБОТА

Пояснювальна записка

. другий (магістерський) .
(рівень вищої освіти)

Дослідження продуктівності фреймворків при створенні інтернет-магазинів

Виконав:

студент групи ІПЗздм-21-1

Барченко П. В.
(прізвище, ініціали)

Спеціальність 121 – Інженерія програмного

забезпечення

Тип програми Освітньо-наукова

Керівник доц., к.т.н, Мазурова О. О.
(посада, прізвище, ініціали)

Допускається до
захисту

З.В. Дудар

Зав. кафедри

2023 р.

Харківський національний університет радіоелектроніки

Факультет Центр післядипломної освіти

Кафедра Програмної інженерії

Рівень вищої освіти другий (магістерський)

Спеціальність 121 – Інженерія програмного забезпечення

(код і повна назва спеціальності)

Тип програми освітньо-наукова програма

Освітня програма Інженерія програмного забезпечення

ЗАТВЕРДЖУЮ:

Зав. кафедри _____

(підпис)

«_____»_____ 2023 р.

ЗАВДАННЯ

НА КВАЛІФІКАЦІЙНУ РОБОТУ

студенту Барченко Павло Васильович

(прізвище, ім’я, по батькові)

1. Тема роботи Дослідження продуктівності фреймворків при створенні інтернет-магазинів.

затверджена наказом по університету від 03.04.2023 р. № 83Стз

2. Термін подання студентом роботи до екзаменаційної комісії 20.05.2023 р.

3. Вихідні дані до роботи дослідження продуктівності фреймворків при створенні інтернет-магазинів у вигляді програмної системи (фреймворку) для клієнської імлементації, проаналізувати отримані результати.

4. Перелік питань, що потрібно опрацювати в роботі вступний опис питання, аналіз предметної галузі, збір даних, побудова моделі, побудова архітектури модулів, аналіз результатів у висновках.

КАЛЕНДАРНИЙ ПЛАН

№	Назва етапів роботи	Терміни виконання етапів роботи	Примітка
1	Аналіз проблемної галузі	23.01.2023 – 30.01.2023	Виконано
2	Постановка задачі дослідження	01.02.2023	Виконано
3	Аналіз фреймворків	05.02.2023 – 23.02.2023	Виконано
4	Планування експериментів	23.02.2023 – 30.02.2023	
5	Проектування програмного забезпечення	20.02.2023 – 14.03.2023	Виконано
6	Розробка програмного забезпечення	15.03.2023 – 30.03.2023	Виконано
7	Проведення дослідження продуктивності фреймворків	30.03.2023 – 18.04.2023	Виконано
8	Підготовка пояснювальної записки	20.04.2023 – 25.04.2023	Виконано
9	Підготовка презентації та доповіді	27.04.2023 – 01.05.2023	Виконано
10	Перевірка на плагіат	15.05.2023	Виконано
11	Перевірка на нормоконтроль	16.05.2023	Виконано
12	Занесення диплома в електронний архів	19.05.2023	Виконано
13	Попередній захист	20.05.23	Виконано
14	Допуск до захисту у зав. кафедри	21.05.2023	Виконано

Дата видачі завдання «23» січня 2023 р.

Студент _____ Барченко П. В.
(підпис)

Керівник роботи _____ доц., к.т.н, Мазурова О. О.

РЕФЕРАТ / ABSTRACT

Кваліфікаційна робота магістра містить: 77 стр., 22 рис., 21 джерел.

ВЕБ-СИСТЕМА, МЕТРИКА, РІВЕНЬ ЗАДОВОЛЕНОСТІ КЛІЄНТА, DOCKER, JAVA, POSTMAN, POSTGRESQL, REACT, SPRING.

Об’єкт розробки – програмна система для створення інтернет-магазинів.

Мета розробки – дослідити продуктивність фреймворків та спростити процес розробки e-Commerce рішень задовольнити клієнта та бізнес сучасним архітектурним рішенням.

Метод рішення – середовище розробки Intellij Idea, платформа Java Spring, мови програмування Java, фреймворк Spring, СУБД PostgreSql, контейнеризація Docker, хмарне рішення AWS.

В результаті проведено дослідження методів оптимізації інтеграції між сервісами та розроблено веб-систему, що допомагає спростити процес розробки e-Commerce рішень.

WEB SYSTEM, METRIC, CUSTOMER SATISFACTION, DOCKER, JAVA, POSTMAN, SPRING, POSTGRESQL, REACT.

The object of expansion is a software system for creating online stores.

Development meta – to increase the productivity of frameworks and to simplify the process of e-Commerce development is a solution to satisfy the client and business with modern architectural solutions.

Solution method – Intellij Idea development framework, Java Spring platform, Java mobile programming, Spring framework, PostgreSql DBMS, Docker containerization, AWS smart solution.

As a result, further methods were carried out to optimize the integration between services and split the web system, which helps to simplify the process of splitting the e-Commerce solution.

Умови публікації пояснювальної записки

Я, Барченко Павло Васильович, студент гр. ІПЗздм-21-1, здобувач вищої освіти на другому (магістерському) рівні кафедри «Програмна інженерія», заявляю: моя кваліфікаційна робота на тему «Дослідження продуктівності фреймворків при створенні інтернет-магазинів», що буде представлена в екзаменаційну комісію для публічного захисту, виконана самостійно, в ній не містяться елементи плагіату і вона може бути опублікована в електронному архіві відкритого доступу ElAr KhNURE. Всі запозичення з друкованих та електронних джерел мають відповідні посилання.

Я ознайомлений з діючим положенням «Про протидію академічному плагіатув ХНУРЕ», згідно з яким виявлення плагіату є підставою для відмови в допуску кваліфікаційної роботи до захисту та застосування дисциплінарних заходів.

ЗМІСТ

Вступ 8
1 Аналіз проблемної області та постановка задачі 10
1.1 Аналіз проблемної області розробки 10
1.2 Аналіз аналогів 16
1.3 Постановка задачі 20
2 Опис прийнятих проектних рішень 22
2.1 Аналіз та визначення основних вимог до фреймворків 22
2.2 Аналіз існуючіх технології реалізації фреймворків 24
2.3 Аналіз моделювання предметної області та розробка схеми бази даних 35
2.4 Проектування обраних архітектур 39
2.5 Планування експериментального дослідження 43
3 Опис програмної реалізації 46
3.1 Вибір засобів програмної реалізації 46
3.2 Опис фізичної моделі БД 47
3.3 Опис реалізації модулів фреймворму 51
3.3.1 Реалізація модуля tds.core 51
3.3.2 Реалізація модуля cache.core 53
4 Результати експериментального дослідження 55
4.1 Результати дослідження продуктивности фреймворків 55
4.2 Опис рекомендацій 56
Висновки 58
Перелік джерел посилання 60
Додаток А Перелік джерел посилання за науковими напрямами керівника та науковців кафедри програмної інженерії 62
Додаток Б Результати перевірки роботи на академічну доброчесність 63
Додаток В Слайди презентації 64

Додаток Г Апробація результатів 72

Додаток Д Звіт результатів перевірки на унікальність тексту 76

Додаток Е Експертний висновок результатів перевірки кваліфікаційної роботи на відповідність оформлення вимогам ДСТУ 3008: 2015 77

ВСТУП

Сучасні сайти, додатки, сервіси стають все більш складними, динамічними, багатофункціональними. Вони повинні активно взаємодіяти з користувачем, бути здатними підходити під мінливе середовище, що обумовлює постійний розвиток методів їх розробки, підтримувати. Прагнучи відповідати зростаючим потребам ринку IT-технологій, фахівці розробили новий інструмент – веб-фреймворк. Він представляє собою каркас, платформу для створення веб-продуктів нового покоління, їх ефективної підтримки. Він призначений для складних, масштабних проектів, дозволяє реалізувати нестандартні рішення.

Більш детально відповідаючи на питання, що такий фреймворк варто відзначити, що він не формує для продукту, що розробляється жорсткі рамки. Він надає базові модулі, на основі яких створюється ефективний сайт із широкими можливостями модернізації, розширення функціоналу для приєднання додаткових додатків у майбутньому.

Стандартна архітектурна фреймворка ґрунтується на поділі трьох шарів:

- модель – відповідає за формування структури, правил бізнес-логіки;
- виявлення – основна функція – графічне відображення даних;
- контролер – реалізує зв’язок з користувачем, перетворюючи отриману від нього інформацію в команди для попередніх двох шарів.

Крім цього інструменту, розробка веб-сайтів може виконуватися IT-фахівцем самостійно з написанням коду з нуля або із залученням готової CMS. Перший варіант занадто витратний за час, вимагає маси зусиль у ході реалізації, подальшого тестування. Використання CMS дозволяє швидко створювати інтернет-додатки, проте накладає на їх функціонування певні обмеження, тобто вийти за рамки «конструктора» не буде. Такі способи розробки більше підходять для простих, статичних проектів.

Метою даної роботи є дослідження побудови архітектури фреймворку для e-commers рішень, а також оптимізації інтеграцій між модулями ПС. Під час виконання кваліфікаційної роботи був проведений аналіз проблемної області

продуктивності сервісних інтеграцій на основі публікацій світових та вітчизняних фахівців в проблемній області (див. додаток А).

Кваліфікаційна робота пройшла успішну перевірку на академічну доброчесність (див. додаток Б).

На основі плану проведення дослідження та його результатів було розроблено презентацію (див. додаток В). За результатами роботи були створені тези доповіді IV Міжнародна науково-практична конференції молодих вчених та студентів «Інженерія програмного забезпечення і передові інформаційні технології (SoftTech-2023)» присвяченої 125-й річниці КПІ ім. Ігоря Сікорського (див. додаток Г).

Також, кваліфікаційна робота перевірена на відповідність вимогам оформлення (див. додаток Д).

1 АНАЛІЗ ПРОБЛЕМНОЇ ОБЛАСТІ ТА ПОСТАНОВКА ЗАДАЧІ

1.1 Аналіз проблемної області дослідження

Фреймворки – це програмні продукти, які спрощують створення та підтримку технічно складних чи навантажених проектів. Фреймворк, зазвичай, містить лише базові програмні модулі, проте специфічні для проекту компоненти реалізуються розробником з їхньої основі. Тим самим досягається не тільки висока швидкість розробки, а й велика продуктивність та надійність рішень.[1]

Веб-фреймворк – це платформа для створення сайтів та веб-додатків, що полегшує розробку та об'єднання різних компонентів великого програмного проекту. За рахунок широких можливостей у реалізації бізнес-логіки та високої продуктивності ця платформа особливо добре підходить для створення складних сайтів, бізнес-додатків та веб-сервісів.

Розглянемо економічну ефективність та доцільність використання фреймворків.

З погляду бізнесу розробка на фреймворку майже завжди економічно ефективніша і якісніша за результатом, ніж написання проекту чистою мовою програмування без використання будь-яких платформ. Розробка без використання платформи може бути правильним рішенням лише у двох випадках – або проект дуже простий і не потребує подальшого розвитку, або дуже навантажений і потребує дуже низькорівневої оптимізації (наприклад, веб-сервіси з десятками тисяч звернень на секунду). У всіх інших випадках розробка на програмній платформі швидше та якісніше.

Якщо порівнювати фреймворки з іншими класами платформ – SaaS, CMS або CMF – то фреймворки значно ефективніше використовувати у проектах зі складною бізнес-логікою та високими вимогами до швидкості роботи, надійності та безпеки. Але в простих і типових проектах без значних вимог швидкість та вартість розробки на фреймворку буде вищою, ніж на SaaS чи CMS.[2]

Однією з головних переваг використання фреймворків і те, що фреймворк визначає уніфіковану структуру для побудованих з його додатків. Тому програми на фреймворках значно простіше супроводжувати і доопрацьовувати, оскільки

стандартизована структура організації компонентів зрозуміла всім розробникам на цій платформі і не потрібно довго розбиратися в архітектурі, щоб зрозуміти принцип роботи програми або знайти місце реалізації того чи іншого функціоналу. Більшість фреймворків для розробки веб-додатків використовує парадигму MVC (модель-представлення-контролер) – тобто дуже у багатьох фреймворках ідентичний підхід до організації компонентів програми, і це ще більше спрощує розуміння архітектури програми навіть на незнайомому розробнику фреймворку.[1]

Проектування архітектури ПЗ при розробці на фреймворку теж дуже спрощується – у методологіях фреймворків зазвичай закладено найкращі практики програмної інженерії і просто дотримуючись цих правил можна уникнути багатьох проблем та помилок у проектуванні. По суті, фреймворк – це безліч конкретних та абстрактних класів, пов'язаних між собою та впорядкованих згідно з методологією фреймворку. Конкретні класи зазвичай реалізують взаємні відносини між класами, а абстрактні класи є точки розширення, в яких закладений у фреймворк базовий функціонал може бути використаний «як є» або адаптований під завдання конкретного додатка. Для забезпечення розширення можливостей більшості фреймворків використовуються техніки об'єктно-орієнтованого програмування: наприклад, частини програми можуть успадковуватися від базових класів фреймворку або окремі модулі можуть бути підключені як домішки.

Екосистеми веб-фреймворків також багаті на готові для реалізації багатьох функціональних можливостей. Розробникам під час роботи над типовими завданнями не треба «винаходити велосипеди», оскільки вони можуть скористатися вже створеною спільнотою реалізацією. А це не тільки скорочує витрати часу і грошей, а й дозволяє досягти вищої стабільності рішення – компонент, який використовується і допрацьовується тисячами інших розробників, зазвичай більш якісно реалізований і краще протестований на всіляких сценаріях, ніж рішення, яке може в адекватні терміни розробити один розробник. чи навіть невелика команда.[2]

Бібліотека – це найпростіший компонент архітектури програмного забезпечення. Програмна бібліотека може бути використана просто як набір підсистем близької функціональності, не впливаючи на архітектуру основного програмного продукту та не накладаючи на неї жодних обмежень.

Фреймворк ж непросто дає розробнику потрібний функціонал, а ще й диктує правила побудови архітектури докладання, задаючи на початковому етапі розробки поведінка за умовчанням, формуючи каркас, який потрібно буде розширювати і змінювати відповідно до зазначених вимог. Фреймворк також може включати допоміжні програми, бібліотеки коду, мову сценаріїв та інше програмне забезпечення, що полегшує розробку та об'єднання різних компонентів великого програмного проекту.[3]

Перед веб-розробниками часто стоїть вибір між коробковими CMS та фреймворками для реалізації проекту. У кожного з підходів є свої плюси та мінуси, нижче ми розглянемо переваги та недоліки розробки на фреймворках.

Розробка на фреймворку (на відміну від самописних рішень) дозволяє досягти простоти супроводжуваності проекту.

Можлива (і відносно проста) реалізація будь-яких бізнес-процесів, а не лише тих, що спочатку закладені в систему. Також проекти на базі фреймворків легко масштабуються та модернізуються.

Рішення на фреймворках, як правило, працюють значно швидше і витримують більше навантаження, ніж CMS та самописні системи. Саме тому багато популярних інтернет-магазинів працюють не на коробкових CMS, а на фреймворках. За рівнем безпеки рішення на фреймворках значно перевершують самописні системи і можна порівняти з CMS (зазвичай, сайти на фреймворках навіть безпечніше).[4]

Терміни розробки типового функціоналу на фреймворках більші, ніж під час використання CMS. Фреймворки містять лише базові компоненти бізнес-логіки рівня програми, тому багато функцій реалізуються індивідуально.

Для розробки на фреймворку потрібне розуміння бізнес-процесів, які потрібно реалізувати. Наприклад, якщо в CMS вже є якийсь процес обробки замовлень, то фреймворки такого не надають.[5]

Для розробки інтернет-магазинів існує кілька популярних фреймворків, які надають потрібні інструменти та функціонал для створення вдалого електронного комерційного рішення.

Один з найпопулярніших фреймворків для інтернет-магазинів – це WooCommerce. Цей фреймворк, розроблений на базі WordPress, забезпечує широкі можливості для створення та керування інтернет-магазином. WooCommerce має вбудовану підтримку для роботи з базою даних, що дозволяє зберігати та керувати інформацією про продукти, замовлення, клієнтів та інше. Крім того, він має багатий вибір розширень і плагінів, що дозволяють розширити його функціональність.

Інший популярний фреймворк – Magento, який відомий своєю потужністю та розширюваністю. Magento надає розширений функціонал для створення та управління інтернет-магазинами. Він має вбудовані засоби для роботи з базою даних, що дозволяють зберігати великі обсяги інформації про продукти, замовлення, користувачів та інші аспекти електронної комерції. Magento також надає можливість розширення функціоналу за допомогою різноманітних модулів та розширень.

Крім того, варто зазначити Shopify – хмарний фреймворк для створення інтернет-магазинів. Він пропонує простий у використанні інтерфейс та багатий функціонал для створення та управління магазином. Shopify має вбудований механізм для роботи з базою даних.

Інтернет-магазини використовують БД, то такий фреймворм повинен містити відповідний функціонал. Отримання точної бази даних для інтернет-магазину може залежати від конкретних потреб та вимог вашого проекту.[5]

Бази даних (БД) бувають різних типів, і вибір конкретного типу залежить від потреб та вимог вашого проекту. Основні типи БД включають:

– реляційні бази даних: Це найпоширеніший тип БД, у якому дані зберігаються у вигляді таблиць зі зв'язками між ними. Використовуються SQL (Structured Query Language) для запитів та маніпуляцій із даними. Приклади реляційних баз даних включають MySQL, Oracle, Microsoft SQL Server;
– нереляційні бази даних (NoSQL): Ці БД використовують інші моделі для зберігання та організації даних, відмінні від традиційних таблиць. Вони ефективні для масштабованих, розподілених систем та обробки великих обсягів неструктурованих даних. Приклади NoSQL-БД включають MongoDB, Cassandra, Redis;
– ієрархічні бази даних: У цьому типі БД дані організовані у вигляді деревоподібної структури, де кожен елемент має батьківський та дочірній вузли. Цей тип БД добре підходить для роботи з ієрархічно організованими даними, такими як файлові системи;
– мережеві бази даних: Цей тип БД використовує мережеву модель для зберігання даних, де кожен запис може мати багато батьківських та дочірніх вузлів. Використовується у розподілених обчисленнях та географічно розподілених системах.

Етапи розробки бази даних включають такі кроки:

– перший крок - збір вимог до бази даних. Це включає зрозуміння потреб бізнесу, визначення основних сутностей, атрибутів та зв'язків між ними;
– аналіз та проектування: На цьому етапі визначається структура бази даних. Включає розробку схеми бази даних, вибір типу БД та моделей даних, таких як реляційна, нереляційна або ієрархічна. Також визначаються правила цілісності даних та інші обмеження;
– створення БД: Після проектування розпочинається фаза реалізації. Це включає створення фізичної бази даних з використанням вібраної технології, налаштування параметрів та створення таблиць, індексів, відносин із відповідно до проекту;

- завантаження даних: Після створення бази даних потрібно завантажити початкові дані. Це може бути ручне завантаження даних або імпорт існуючих даних з інших джерел;
- тестування та оптимізація: Після завантаження даних слід провести тестування бази даних для перевірки її працездатності, цілісності та швидкодії. При необхідності можуть бути внесені зміни та оптимізації, які покращують продуктивність та ефективність бази даних;
- впровадження та підтримка: Після успішного тестування база даних готова до впровадження у реальній середовищі. Надалі потрібно забезпечити постійну підтримку та обслуговування бази даних, включаючи резервне копіювання, моніторинг та інше.[5]

Фреймворки – це ПЗ, а відповідно потребують під час своєї розробки виконання етапів розробки архітектури. Етапи розробки архітектури фреймворків можуть варіюватися в залежності від конкретного проекту та методології розробки. Однак, загальні етапи включають наступні кроки[5]:

- визначення вимог: першим етапом є збір та аналіз вимог до фреймворку. Це включає визначення функціональних та нефункціональних вимог, потреб користувачів та цілей фреймворку;
- проектування архітектури: на цьому етапі визначається загальна структура та організація фреймворку. Включає у себе розробку концептуальної архітектури, ідентифікацію основних компонентів та їх взаємозв'язків;
- вибір технологій: після проектування архітектури необхідно вибрати технології та інструменти, які будуть використовуватись для реалізації фреймворку. Це можуть бути мови програмування, бібліотеки, фреймворки третьої сторони та інші технології;
- реалізація компонентів: на цьому етапі розробляються окремі компоненти фреймворку, такі як роутер, контролери, моделі, перегляди тощо. Кожен компонент повинен бути розроблений згідно з архітектурою та добрими практиками;

– інтеграція та тестування: після реалізації компонентів вони повинні бути інтегровані разом та піддані тестуванню. Це включає функціональне та модульне тестування, а також перевірку відповідності вимогам;
– документування: створення документації.

1.2 Аналіз аналогів

Під час аналізу ринку серед існуючих рішень було знайдено наступні системи:

- Hybris;
- Oracle ATG;
- Magento.

SAP Hybris є продукцією сімейства німецької фірми Hybris, яка є програмним забезпеченням для e-commerce, marketing, sales, maintenance and product content management. SAP Hybris забезпечує рішення, що сприяють будь-якій організації зменшення вартості, обмеження часу, зменшення складності, і потребує невеликої уваги до того, щоб досягти вищої вартості експертів.

Hybris був введений в 1997 році в Zug, шведська і останній був придбаний SAP SE on August 1, 2013. SAP ha integred його власний внутрішній backend system – SAP CRM і SAP ERP з Hybris solution, будь-яка компанія, що має SAP ERP або SAP може бути надійно migrate до SAP Hybris solution. SAP Hybris Commerce Accelerator є Omni e-commerce channel solution with storefront templates and tools, які розраховують на амортизацію customer experience. Hybris Product focuses on following main areas[6]:

- Commerce;
- Marketing;
- Revenue (Billing);
- Sales;
- Service;
- Hybris as a Service (YaaS).

Маркетинг потрібен для того, щоб зрозуміти поведінку клієнтів у режимі реального часу та надати їм те, що вони хочуть і коли вони хочуть.

Дохід (Біллінг) – це рішення дає компанії можливість працювати у складних партнерських екосистемах, перепродувати продукти та ділитися доходами. Він включає продукти для управління замовленнями по підписці, дохід у хмарі, адаптивний контроль якості, управління фінансами клієнтів, консолідований білінг, виставлення рахунків і багато іншого.[7]

Хмара SAP Hybris для продажу отримує інформацію про клієнтів із серверної системи, передає її групі зовнішніх клієнтів та дозволяє їм зрозуміти цільових клієнтів та дізнатися, як використовувати кожну можливість нового продажу. Це забезпечує інформацію для менеджера з продажу на місцях та на мобільному пристрої в руках менеджера з продажу. Він включає продукт для роздрібного виконання, управління ефективністю продажів і автоматизації продажів.

SAP Hybris for Service дозволяє компанії надавати своїм клієнтам винятковий досвід обслуговування та, отже, чудовий досвід взаємодії з клієнтами. Використовуючи Hybris Service організації можуть надавати своїм клієнтам правильний сервіс на потрібному каналі. Він включає продукт для Omni Channel Call Center, Проактивне обслуговування на місцях і Комплексне самообслуговування.[7]

Hybris як послуга (YaaS) – це одна з найбільш розвинених екосистем мікро-послуг, яка дозволяє компанії розробляти користувацькі програми на існуючій платформі. YaaS дозволяє компаніям повторно збирати та адаптувати існуючі сервіси для створення інтерфейсів користувача без необхідності їх розробки з нуля.

SAP Hybris надає продукти для електронної комерції, маркетингу, продажу та обслуговування, доходів та міжфункціональних рішень, і вони покликані допомогти організаціям створювати цінні взаємодії зі своїми клієнтами та підтримувати різні типи галузей.[7]

Hybris включає портфель продуктів для наступних можливостей:

- товари для торгівлі;
- продукти для маркетингу;
- продукти для продажу;
- товари для сервісу;
- продукти для виставлення рахунків;
- крос-функціональне рішення.

До переваг можна віднести:

- динамическое профилирование клиентов;
- сегментация и управление кампаниями;
- коммерческий маркетинг;
- управление лояльностью;
- управление маркетинговыми ресурсами;
- маркетинговый анализ;
- управление маркетингом;
- атрибуция клиентов;
- архитектура и технология.

Недоліки має наступні:

- неможливість роботи з фідбеками;
- одночасний аналіз тільки за одною метрикою;
- дорога вартість підписки.

Більшість CMS (система керування контентом) орієнтовані на інтернет-магазини для роздрібної торгівлі. Сучасна електронна комерція пропонує можливості і для сектора B2B, наприклад, як ATG Oracle – ефективне рішення для великого бізнесу. Розробники постаралися врахувати всі потреби великих компаній, створивши безліч додаткових інструментів для роботи онлайн.[8]

Magento є одним з найпопулярніших і найпотужніших відкритих сорсових фреймворків для створення електронної комерції. Ось більш детальна інформація про Magento:

Функціонал: Magento надає широкий функціонал для створення та управління електронними магазинами. Він включає можливості для налаштування

каталогу товарів, управління замовленнями та оплатами, організації доставки, управління складами, промоційними акціями, скидками та купонами.

Розширюваність: Magento має гнучку архітектуру, яка дозволяє розробникам розширювати його функціонал за допомогою модулів. Є безліч безкоштовних та платних модулів Magento, які дозволяють додавати нові функції та інтегруватися з різноманітними сервісами.

Теми та дизайн: Magento надає можливості для налаштування зовнішнього вигляду магазину за допомогою тем. Є багато безкоштовних та платних тем Magento, які дозволяють створювати професійний та привабливий дизайн магазину.

Маркетинг та аналітика: Magento має вбудовані інструменти для проведення маркетингових кампаній, таких як акції, скидки, купоні, промо-коди та програми лояльності. Також доступні засоби аналітики, що дозволяють власникам магазинів отримувати статистику про продаж, відвідуваність та поведінку покупців.

Масштабованість та продуктивність: Magento може ефективно працювати з великими обсягами товарів та високим навантаженням.[8]

Переваги Magento:

- широкий функціонал;
- легко розширювати функціонал магазину за допомогою модулів;
- підтримує масштабованість;
- активну спільноту розробників.

Недоліки Magento:

- складність встановлення та налаштування.

1.3 Постановка задачі

Враховуючи недоліки та переваги вищенаведених систем, метою роботи є дослідження найпопулярніших фреймворків та створення власного фреймворку, який зможе надавати такі можливість як:

- інтуїтивний і зручний інтерфейс: Фреймворк може надати простий і зрозумілий інтерфейс для адміністраторів OPS та звичайних користувачів. Це дозволить легко навчати персонал, зменшити помилки та прискорити робочі процеси;
- модульність та розширюваність: Фреймворк може бути розроблений з урахуванням модульної структури, що дозволить додавати нові функціональні можливості та розширювати систему електронної комерції у майбутньому. Наприклад, додавання нових методів оплати або інтеграція з зовнішніми сервісами доставки;
- безпека та авторизація: Фреймворк може включати механізми для захисту даних, аутентифікації та авторизації. Це важливо для забезпечення безпеки користувачів та адміністраторів і запобігання несанкціонованому доступу до системи;
- аналітика та звіти: Фреймворк може мати вбудовані засоби аналітики та звітності, які дозволять відстежувати ключові метрики про продажі, відвідуваність та поведінку користувачів. Це надасть адміністраторам цінну інформацію для прийняття рішень щодо вдосконалення системи;
- тестування продуктивності: Фреймворк може включати засоби для тестування продуктивності системи електр.

Для досягнення цієї мети необхідно виконати наступні завдання:

- провести аналіз предметної області роботи інтернет-магазинів та визначити основні вимоги до фреймворків, що використовуються для їх створення;
- проаналізувати існуючі технології реалізації відповідних фреймворків та обрати відповідні для реалізації власного програмного рішення;
- провести моделювання предметної області та розробити схему бази даних;

- розробити архітектуру власного фреймворки;
- провести планування дослідження (а саме, обрати метрики для дослідження продуктивності використання фреймворків, обрати 2.1 Аналіз та визначення основних вимог до фреймворківпрограмні рішення, що будуть досліджуватися, сформулювати умови експериментів навантаження, тощо);
- провести дослідження продуктивності отриманих програмних рішень та сформулювати рекомендації.

Описані завдання мають бути виконані у формі веб-додатка. База даних має бути спроектована з дотриманням наступних вимог [9]:

- база має бути цілісною;
- дані мають бути атомарні;
- простота оновлення даних;
- збір статистичної інформації.

Дана система повинна бути реалізована з використанням клієнт-серверної архітектури. Продукт має бути легко супроводжувальним. Його архітектура має бути багато модульна. Це дозволить легко розширювати функціонал фреймверку.

2 ОПИС ПРИЙНЯТИХ ПРОЕКТНИХ РІШЕНЬ

2.1 Аналіз та визначення основних вимог до фреймворків

E-Commerce – це сфера економіки, в якій реклама, просування, торгівельні та фінансові угоди здійснюються безпосередньо в Інтернеті. Тобто коли ви щось купуєте або продаєте в Мережі, це і є e-Commerce.

Варто зрозуміти, що сучасні системи електронної комерції – це вже не просто сайти, які допомагають щось купити. Тепер атака на рітейлера йде з різних боків – від інтернет-магазину, мобільної та соціальної комерції, тіла маркетингу. Впровадження багатоканальної торгівлі дає можливість охопити різні сегменти та значно більше споживачів. При цьому потрібно використовувати багато форматів розкручування, врахувати особливості та переваги різних груп покупців.[6]

Інший важливий момент – покупця необхідно не лише залучити, а й утримати. Це вказує на необхідність підключення системи керування відносинами з клієнтами та лояльністю. [9]

Також знадобляться програми, що дозволяють персоналізувати послуги та пропозиції під конкретного покупця.

З погляду власників бізнесу, виробників та постачальників електронна комерція – це просування та надання своїх товарів чи послуг через інтернет. А з погляду покупців (клієнтів) – перегляд, вибір, порівняння та замовлення.

Для проведення аналізу та визначення основних вимог до фреймворків, використованих у системах електронної комерції, можна розглянути наступні аспекти[9]:

- функціональність: Фреймворк повинен мати широкий спектр функціональних можливостей, які відповідають потребам електронної комерції. Це можуть бути такі функції, як управління товарами, каталоги, кошики покупок, системи оплати та доставки, адміністрування замовлень та зворотного зв'язку з користувачами;
- масштабованість: Фреймворк повинен бути готовим до масштабування, щоб забезпечити рост системи електронної комерції зі збільшенням обсягів транзакцій та кількості користувачів. Це може включати

горизонтальне та вертикальне масштабування, кешування, розподілені системи та інші стратегії;

- безпека: Фреймворк повинен забезпечувати захист від потенційних загроз безпеці, таких як злам користувача, крадіжка даних, шахрайство з оплатою та інші атаки. Це може включати механізми шифрування, автентифікації, авторизації, контролю доступу та моніторингу безпеки;
- інтеграція з зовнішніми сервісами: Фреймворк повинен мати можливість легко інтегруватися з іншими сервісами, такими як платіжні шлюзи, системи доставки, соціальні медіа, аналітичні інструменти та інші. Це дозволить розширити функціональність та покращити взаємодію з користувачами;
- підтримка мобільних пристроїв: З урахуванням сучасних підходів у UX/UI дизайні.

На основі аналізу предметної області та аналізу існуючих систем були виділені наступні функції[9]:

- керування продуктами: Дозволяє додавати, редагувати та удаляти продукти з каталогу, встановлювати ціни, вказувати варіації товарів (колір, розмір тощо), керувати залишками на складі.
- корзина покупок та оформлення замовлення: Дозволяє покупцям додавати товари до кошика, здійснювати оплату, вибирати метод доставки, заповнювати дані для оформлення замовлення.
- управління замовленнями: Надає можливість перегляду та відстеження статусу замовлення, управління доставкою, оновлення інформації про замовлення.
- інтеграція з платіжними шлюзами: Забезпечує можливість приймання онлайн-платежів за допомогою різних платіжних систем, таких як PayPal тощо.
- управління користувачами: Дозволяє реєструвати користувачів, керувати їх профілями, надавати доступ до особистих кабінетів, зберігати історію замовлень.

- аналітика та звітність: Забезпечує збір та аналіз даних про продаж, відвідуваність, конверсію, дозволяє формувати звіти та статистику для прийняття управлінських рішень.

2.2 Аналіз існуючих технології реалізації фреймворків

Рівень розвитку сучасних технологій такий високий, що дозволяє побудувати ІС будь-якого масштабу, складності та функціональності. Однак, з огляду на вимоги бізнесу, засновані на показниках різних бізнес-оцінок, виникають додаткові складно і, вирішення яких зводиться до забезпечення раціонального підходу до процесу проектування, реалізації й подальшої експлуатації інформаційних систем. Виходячи з цього, можна однозначно вважати обрану архітектуру одним з основних показників ефективності створюваної інформаційної системи, а, отже, і успішності бізнесу.[10]

Визначити поняття "архітектура інформаційної системи" можна безліччю способів. Це пов'язано:

- з відсутністю загальноприйнятого визначення самої інформаційної системи. З огляду на складність структури, достатнім способом описати її можливо тільки при консолідації декількох точок зору, що в кожному конкретному випадку може приводити до різних результатів;
- з різноманіттям трактувань самого терміну "архітектура".

У результаті, архітектуру інформаційної системи можна описати як концепцію, що визначає модель, структуру, виконувані функції й взаємозв’язок компонентів інформаційної системи. Процедура вибору архітектури для проектованої інформаційної системи, у ринкових умовах, зводиться до визначення вартості володіння нею.[10]

Вартість володіння інформаційною системою складається із планових витрат і вартості ризиків. Планові витрати містять у собі вартість технічного обслуговування, модернізації, зарплату обслуговуючого персоналу й т.д.

Сукупна вартість ризиків визначається з вартості всіх типів ризиків, їхніх ймовірностей і матрицею відповідності між ними. Сама ж матриця відповідності

визначається обраною архітектурою інформаційної системи. Можна виділити найбільш важливі типи ризиків [10]:

- проектні ризики (ризики при створенні системи);
- ризики розробки (помилки, недостатня оптимізація); технічні ризики (простої, відмови, втрата даних);
- бізнес-ризики (виникають через технічні ризики й пов’язані з експлуатацією системи);
- невизначеності (пов’язані з варіативністю бізнесів-процесів і складаються з необхідності внесення змін у систему й не оптимальну процедуру функціонування);
- операційні (мають на увазі невиконання набору операцій, можуть виникати через технічні ризики й бути ініціаторами бізнесів-ризиків).

Концепція архітектури інформаційної системи повинна формуватися ще на етапі технікоекономічного обґрунтування й вибиратися такою, щоб вартість володіння нею була мінімальною.

Для того щоб конструктивно визначити архітектуру, необхідно відповісти на ряд питань [10]:

- Що робить система?
- На які складові частина вона розділена?
- Яким чином відбувається взаємодія цих частин?
- Як і де ці частини розміщені?

Таким чином, можна вважати архітектуру інформаційної системи моделлю, що визначає вартість володіння через наявну в даній системі інфраструктуру.

Термін фреймворк можна визначити як загальноприйняті архітектурно-структурні рішення й підходи до проектування. Можна сказати, що фреймворк являє собою загальне рішення складного завдання. Класифікація фреймворків наведена на рисунку 2.1.[10]

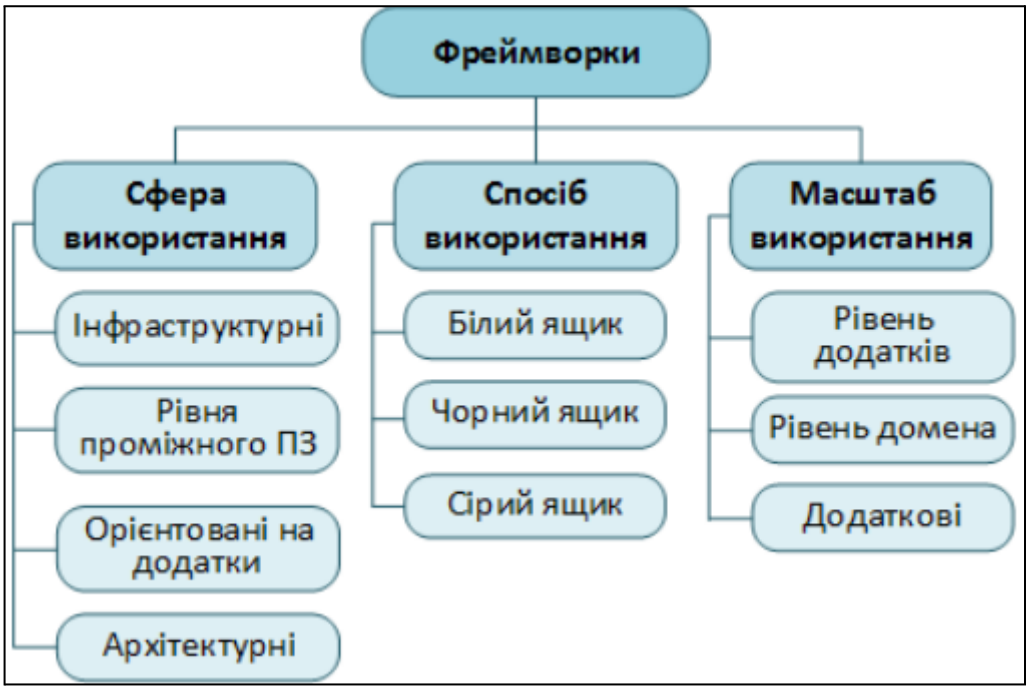

Рисунок 2.1 – Класифікація фреймворків [10]

Інфраструктурні фреймворки (System Infrastructure Frameworks) спрощують процес розробки інфраструктурних елементів, застосовуються усередині організації й не продаються. Фреймворки рівня проміжного програмного забезпечення (Middleware Frameworks) застосовуються для вбудовування додатків і компонентів.

Фреймворки, орієнтовані на додатки, використовуються для підтримки систем, орієнтованих на роботу з кінцевими користувачами в конкретній предметній області. У відповідності зі стандартом КОЛЕС42010 архітектурний фреймворк визначається як «сукупність угод, принципів і практик, які використовуються для опису архітектур і прийнятих відповідно деякому предметному домену й (або) у співтоваристві фахівців (зацікавлених осіб)».[10]

Архітектурний фреймворк містить у собі опис зацікавлених осіб, типові проблеми предметної області, архітектурні точки зору й методи їхньої інтеграції. Фреймворки, використовувані за принципом білого ящика (Architecture-driveng framework), застосовують методи спадкування й динамічного зв’язування для формування основних елементів додатка. Такі фреймворки визначаються через

інтерфейси об’єктів, що додають у систему. Для роботи з ними необхідна докладна інформація про класи, розширення яких необхідно.

Фреймворки, що функціонують за принципом чорного ящика, також називають фреймворками, керованими даними. Основними механізмами формування додатків, у цьому випадку, виступають композиція й параметризація, при цьому функціональність забезпечується додаванням додаткових компонентів. Слід зазначити, що процес використання фреймворків, що працюють за принципом чорного ящика простіше, ніж працюючих за принципом білого ящика, однак їхня розробка складніше. На практиці застосовують підхід сірого ящика (grey box), що є комбінацією обох підходів.[10]

Фреймворки рівня додатків (application frameworks) надають функціонал по реалізації типових додатків (GUI, бази даних і т.д). Фреймворки рівня домену (Domain Frameworks) застосовуються для створення додатків у певній предметній області. Класифікація фреймворків рівня домену представлена на рисунку 2.2.

М’які фреймворки мають на увазі можливість власного настроювання для вирішення конкретного завдання, у той час як тверді – ні.[10]

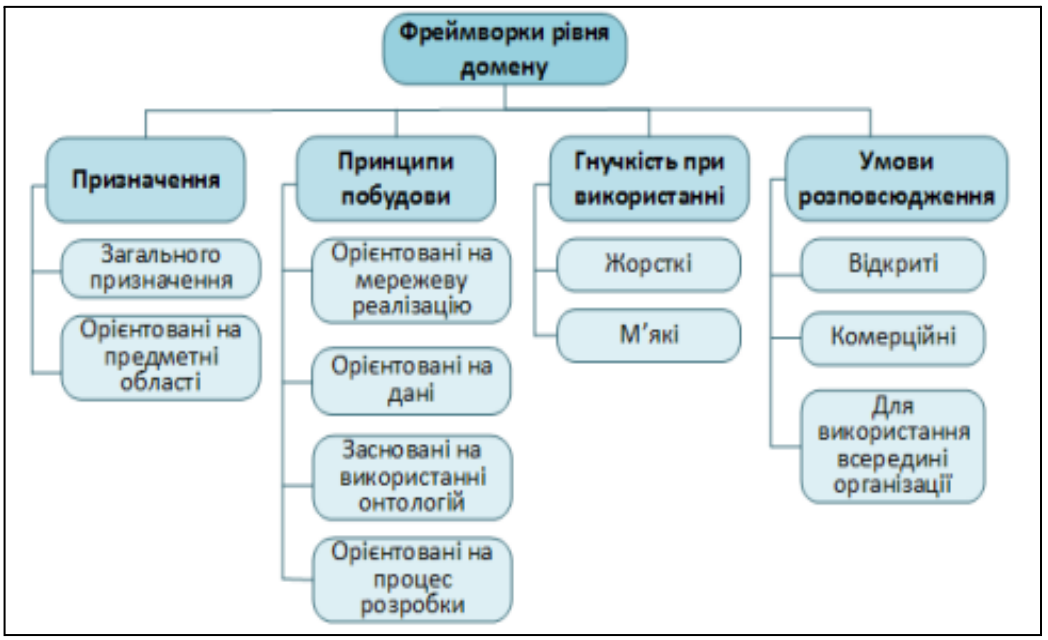

Рисунок 2.2 – Класифікація фреймворків рівня домену[10]

Допоміжні фреймворки (Support Frameworks) застосовуються для рішення приватних завдань.

Термін інтеграція має широке значення. Під ним можна розуміти об’єднання інформаційних систем, додатків, різних компаній або людей. Інтеграція поділяється на зовнішню і внутрішню. Внутрішня інтеграція передбачає об’єднання корпоративних додатків в одній організації (Enterprise Application Integration), а зовнішня – інтеграцію інформаційних систем організацій (Business-to-Business Application).[10]

Існують чотири основні типові інтеграційні підходи:

- інтеграція на рівні даних;
- інтеграція на рівні бізнес-функцій і бізнес-об’єктів;
- інтеграція на рівні бізнес-процесів;
- портали.

Інтеграція на рівні даних (Information- Oriented Integration) має на увазі наявність в системах баз даних, для роботи з якими необхідно розробити єдиний програмний інтерфейс.

До основних технологічних рішень даного підходу відносяться:

- системи реплікації даних; федеративні бази даних;
- використання API для доступу до EPR- системам.

Реплікація є процесом синхронізації даних між різними джерелами. Необхідність у цьому виникає в момент зміни блоку інформації в розподілених системах зберігання для гарантії коректності і несуперечності даних, які використовуються в усіх модулях або додатках інформаційної системи. Зазвичай функції реплікації покладають на проміжне програмне забезпечення.

Федеративні бази даних (Federated Database Systems) надають єдиний інтерфейс до розподілених даних. Це забезпечує інтеграцію безлічі автономних даних, які можуть бути фізично розташовані на різних пристроях в мережі. Такі бази даних прийнято називати віртуальними.[10]

Використання API для доступу до ERP-систем покликане спростити механізми обміну інформацією між одними додатками і програмним забезпеченням, призначеним для управління функціонуванням виробничих інформаційних систем (ERP).

Інтеграція на рівні бізнес-функцій і бізнес об’єктів передбачає реалізацію спільно використовуваних служб (сервісів). Служба може бути набором функцій, використовуваному в декількох додатках. Набір служб і буде бізнес-функціями.

При використанні сервіс-орієнтованої архітектури, бізнес-функції можна розглядати як бізнессервіси, а при компонентному підході – бізнес-об’єктами (бізнес-компонентами). Інтеграція на рівні бізнес-процесів розрізняється залежно від рівня інтеграції.

При внутрішній інтеграції взаємодіє велика кількість сервісів, а при зовнішній інтеграції, в основному, два. Самі бізнес-процеси функціонують над виділеними службами, для управління якими існує спеціальний інтерпретована мова.

Портали можна вважати графічними інтерфейсами бізнес-процесів, оскільки вони призначені для персоналізованого доступу до інформації та консолідації даних з декількох джерел. Головне призначення процесу інтеграції – об’єднання функцій додатків або модулів для надання нової функціональності. [10]

При інтеграції додатків можна виділити два основних типи завдань:

- завдання інтеграції корпоративних додатків;
- завдання інтеграції додатків з різних інформаційних систем.

Для вирішення завдань першого типу застосовуються системи EAI, які іноді називаються A2A (Application-to-Application Integration), а для вирішення завдань другого типу застосовуються системи B2B (Business-to-Business Integration).

У деяких ситуаціях дуже складно визначити різницю між інтеграцією A2A і B2B, оскільки складність деяких рішень всередині інформаційних систем може перевищувати складність рішень для їх спільного функціонування.[10]

Існують три альтернативних топології інтеграції:

- точка-точка (Point-to-Point);
- шлюз (hub-and-spoke);
- шина (Bus).

У топології «точка-точка» представлена на рисунку 2.3, всі об’єкти мають прямі зв’язки один з одним. Слід зазначити, що кожен зв’язок може бути реалізований яким завгодно способом. Варіанти реалізації залежать від вимог і характеристик взаємодії між об’єктами.

До недоліків топології можна віднести:

- недостатня гнучкість;
- складність підтримки численних з’єднань «точка-точка»;
- зміни одного об’єкта впливають на решту;
- логіка маршрутизації часто програмується в коді об’єктів;
- відсутність загальної моделі безпеки;
- використання різних API;
- низька надійність; складність створення фреймворків;
- складність підтримки асинхронного взаємодії.

Для скорочення числа використовуваних інтерфейсів слід використовувати топологію із загальним шлюзом представлена на рисунку 2.4, або топологію із загальною шиною представлена на рисунку 2.5. Такі моделі інтеграції реалізуються на рівні проміжного ПЗ.[10]

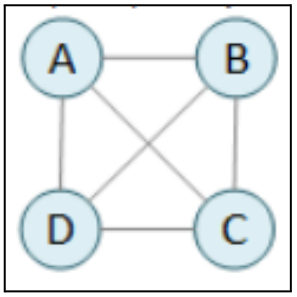

Рисунок 2.3 – Топологія «точка-точка»[10]

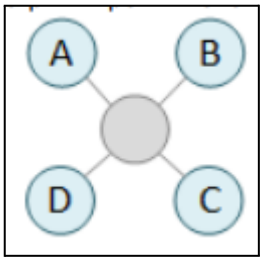

Рисунок 2.4 – Топологія із загальним шлюзом[10]

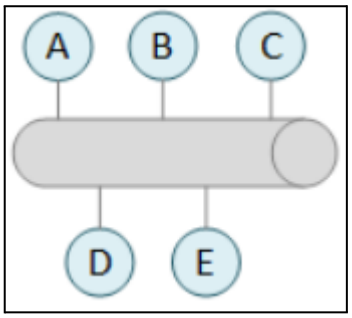

Рисунок 2.5 – Топологія із загальною шиною[10]

Наступним кроком у розробці інтеграційних архітектур можна вважати появу корпоративної сервісної шини (Enterprise Service Bus – ESB).

Ряд авторів розглядають системи ESB, як наступну сходинку розвитку EAI. Однак є кілька відмінностей: EAI – централізована архітектура, з обміном інформації через хаб (брокер), а ESB – шинна архітектура, яка може бути реалізована у вигляді декількох розподілених систем; на відміну від EAI, ESB орієнтована на використання відкритих стандартів.

Ці дві відмінності наочно демонструють можливість використання ESB як інтеграційної платформи, що дозволяє використовувати різні механізми інтеграції. ESB дозволяє проводити як внутрішню, так і зовнішню інтеграції, і являє собою шину (backbone), що працює як слабо зв'язана система, керована подіями.

Концепції сервіс-орієнтованих архітектур (СОА) і ESB дуже сильно пов’язані. ESB підтримує принцип реалізації СОА: поділ подання служби і її реалізації. Функції ESB[10]:

- надання інтерфейсів взаємодії;
- відправлення і маршрутизація повідомлень;
- перетворення даних;
- реакція на події;
- управління політиками; віртуалізація.

Рівень сполучення покликаний вирішувати проблему використання різних інтерфейсів. На цьому рівні функціонують адаптери, які відстежують події в додатках і в інтеграційній підсистемі, і забезпечують перетворення переданих об’єктів при взаємодії з транспортною підсистемою. Існує можливість, окрім

заздалегідь створених адаптерів інтеграційної платформи, використовувати створені самостійно.

Адаптери можна розділити на дві категорії: технологічні (застосовуються для інтеграції технологічних компонент, за відсутності у них API), адаптери для додатків (застосовуються для інтеграції з конкретним додатком).

Транспортна підсистема надає можливість асинхронного взаємодії інтегрованим додаткам. Даний рівень також відповідає за управління та безпеку інформації, може виконувати маршрутизацію повідомлень і їх обробку.

Рівень реалізації бізнес-логіки надає функції для трансформації і маршрутизації повідомлень. На цьому рівні функціонують брокери повідомлень, що обмінюються повідомленнями через транспортну підсистему.[10]

Брокер повідомлень може виконувати такі функції:

- прийняття повідомлень і їх відправка за вказаними адресами;
- перетворення форматів повідомлень;
- агрегування і фрагментація повідомлень;
- взаємодія з репозиторіями;
- вибірка даних через виклики Web-служб;
- обробка помилок і подій;
- маршрутизація повідомлень за адресою, вмістом, темою.

Управління бізнес-процесами на однойменному рівні здійснюється за допомогою BPEL (Business Process Execution Language) за допомогою web-сервісів.

Рівень бізнес-управління представляє з себе надбудову на попередньому рівню і призначений для управління бізнес-процесами в термінах відповідної предметної області.

Підхід ESB має безліч переваг і дозволяє будувати інтеграційні архітектури практично будь якої складності. Типова структура інтеграційної системи представлена на рисунку 2.6.[10]

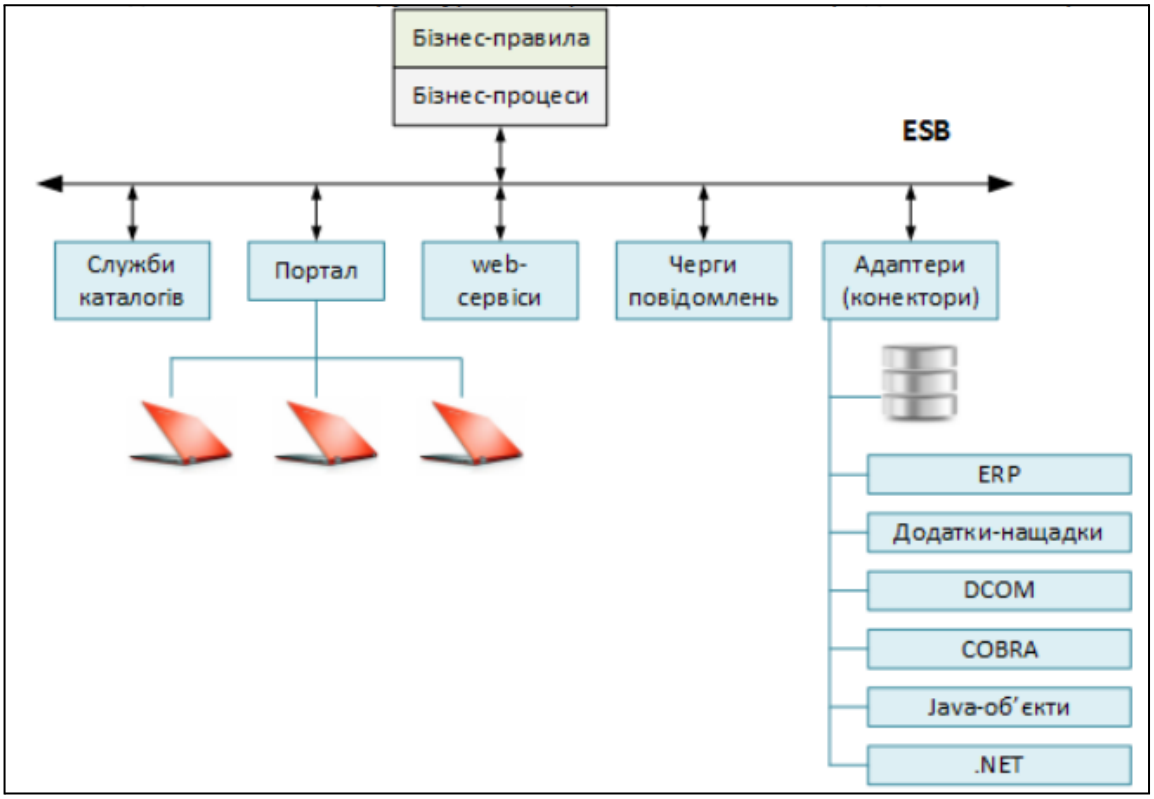

Рисунок 2.6 – Типова структура інтеграційної системи[10]

Під архітектурою програмних систем розуміється сукупність рішень щодо [11]:

- організації програмної системи;
- вибору структурних елементів, що складають систему і їх інтерфейсів;
- поведінки цих елементів у взаємодії з іншими елементами; об'єднання цих елементів у підсистеми;
- архітектурного стилю, що визначає логічну й фізичну організацію системи;
- статичні і динамічні елементи, їх інтерфейси і способи їх об'єднання.

Архітектура програмної системи охоплює не тільки її структурні і поведінкові аспекти, але й правила її використання та інтеграції з іншими системами, функціональність, продуктивність, гнучкість, надійність, можливість повторного застосування, повноту, економічні та технологічні обмеження, а також питання для користувача інтерфейсу. [10]

По мірі розвитку програмних систем все більшого значення набуває їх інтеграція одна з одною з метою побудови єдиного інформаційного простору підприємства. Як можна бачити з вищенаведених визначень інтеграція є найважливішим елементом архітектури.

Для того, щоб побудувати правильну і надійну архітектуру і грамотно спроектувати інтеграцію програмних систем необхідно чітко слідувати сучасним стандартам в цих областях. Без цього велика ймовірність створити архітектуру, яка не здатна розвиватися і задовольняти зростаючим потребам користувачів IT. В якості законодавців стандартів у цій галузі виступають такі міжнародні організації як SEI (Software Engineering Institute), WWW (консорціум World Wide Web), OMG (Object Management Group), організація розробників Java - JCP (Java Community Process), IEEE (Institute of Electrical and Electronics Engineers) та інші.[10]

Кожен архітектор інформаційних систем повинен враховувати три важливі рекомендації.

- контролюйте рамки проекту;
- завжди пам’ятайте, для чого потрібна проектуєма модель. Моделі даних призначені для реалізації, а інформаційні моделі – для документації та обміну інформацією з людьми;
- не дозволяйте інструментальним засобам визначати ваш погляд на речі. Важко документувати обширні і несуворі відношення спадкування, використовуючи тільки SQL або ER-діаграми.

З огляду на існуючі технології реалізації фреймворків. Було прийнято рішення обрати архітектуру з корпоративної сервісної шини (Enterprise Service Bus – ESB).

Підхід ESB дозволяє будувати інтеграційні архітектури практично будь якої складності. Це дозволить обьеднати модулі та зробити декоратор(прослойку) для інтегрування фреймворку зі сторонніми сервісами клієнта. [10]

2.3 Аналіз, моделювання предметної області та розробка схеми бази даних

Програмна система електронної комерції має два типи користувачів: адміністратор системи та користувач. У якості адміністратора системи можуть BI-спеціалісти або менеджери компанії. Опис функціональності і поведінки представлено на діаграмі Use-case, яка представлена нижче на рисунку 2.7. [11]

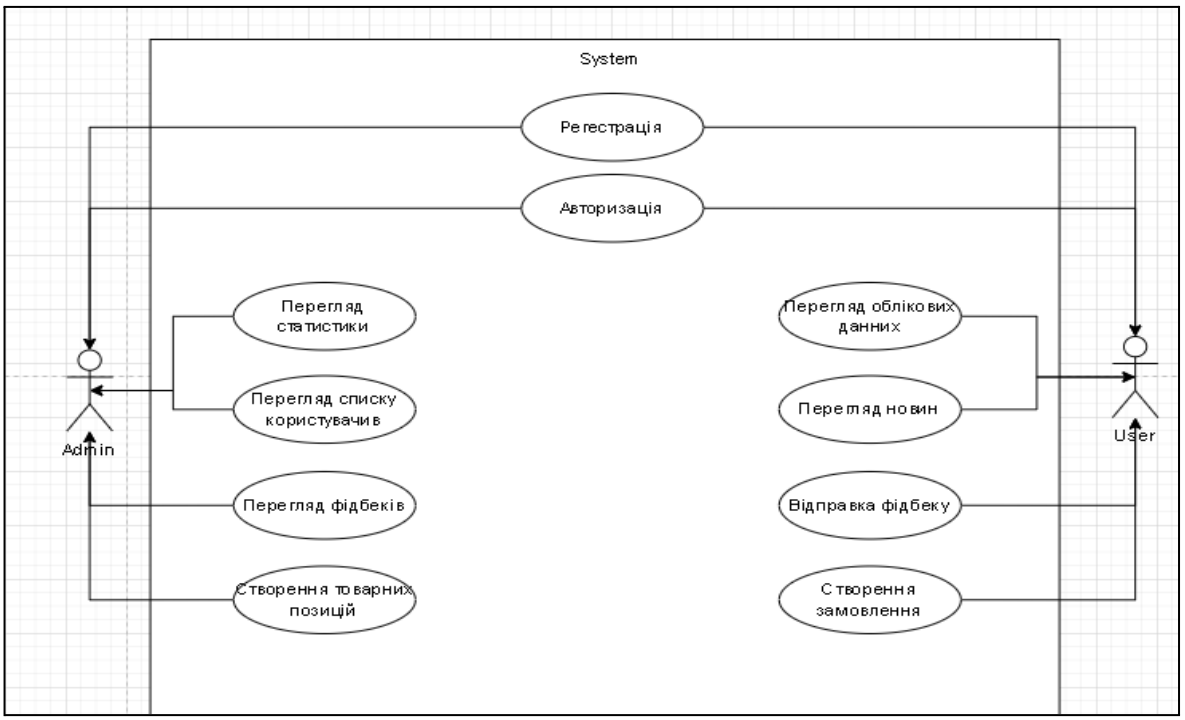

Рисунок 2.7 – Use-case діаграма (Рисунок виконаний самостійно)

До функціональних можливостей адміністратора системи входять наступні можливості:

- реєстрація;
- авторизація;
- перегляд списку користувачів;
- перегляд відгуків;
- перегляд статистики за показниками;
- створення товарної позиції.

До функціональних можливостей користувача системи входять наступні можливості:

- реєстрація;
- авторизація;

- перегляд аккаунту;
- створення замовлення;
- відправка відгуку.

З вище приведених функціональних можливостей реалізований функціонал системи.

Для підтримки приведених функціональних можливостей в складі фреймворки повинна бути реалізована БД , що у собі містить основні сутність інтернет магазину.

Були виведені основні сутність БД бази та побудована діаграма відносин між ними на рисунку 2.8.

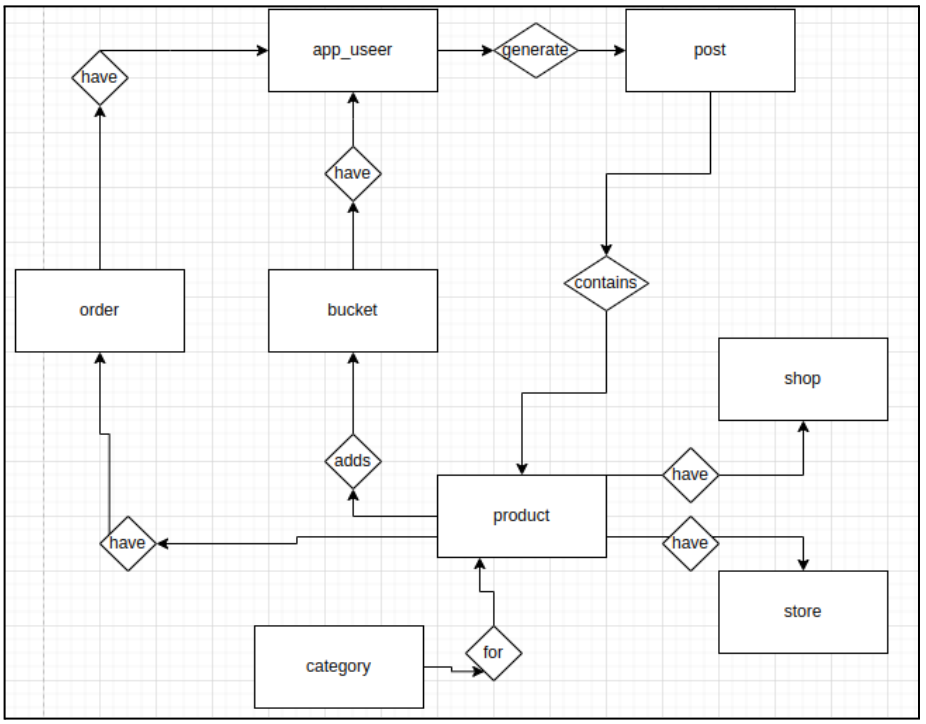

Рисунок 2.8 – Діаграма відносин сутність бази (Рисунок виконаний самостійно)

Було виділені такі сутності як:

- product входять до Shop та Store;
- category для продуктів;
- bucket карзина покупця;
- post комментарій до товару;
- app_user кліент магазину;

- order заказ клієнта.

У процесі моделювання було прийнято рішення використовувати реляційну базу даних. У якості СКБД було обрано PosgreSQL Для доступу до даних буде використовуватися технологія ORM, а саме Hibernate. Це ORM фреймвор який надає можливість взаємодіяти із об'єктами бази даних, як з звичайними об'єктами об'єктно орієнтованих мов програмування.

Базуючись на вимогах до фреймворку, була розроблена наступна схема бази даних, яка складається із 32 таблиці. Були виділені декілько таблиць для сутності, також база була приведена до 3ї форми нормалізації. Схема бази даних зображена на рисунку 2.9.

Загалом атрибути самих сутностей для фреймворку не такі вже і важливі на відмінну від сформованих вадносин між ними. Тому що замовник має мати змогу використовувати фреймворк для різних типів магазинів(електоронні товари, одяг, прикраси то що). У данному випадку сформулювати зв'язки між сутностями і є задачою фреймворку, а вже вибір атрибутив наповнення БД залежіть від конкретного випадку.

Згідно зазначених вище схем, було прийнято такі рішення як:

Розділити сутність користувача (кліент магазину) на декілька таблиць:

- app_user з полями: username, password, email;
- user_details інщі деталі користувача;
- user_payment деталі платежів;
- user_address адресні данні користувача;
- user_role ролью у магазіні(адміністратор чи користувач);
- viewed переглянуті товари користувачом;
- liked товири які користувач помітив як потенційно цікаві.

Bucket карзина покупця так і залишилась окремою сутність.

Order заказ так і залишилась окремою сутність та має додатковуі таблиці:

- order_history для збереження історії замовлень клієнта;
- order_status статус обробки замовлення.

Product продукт розділили між декілька таблиць:

- product_type тип продукту;
- brand бренд товару;
- discount скидка на товар;
- option опції товару(розмір, колір та інше).

Post коментар покупця так і залишилась окремою сутність.

Category кагеторія товару так і залишилась окремою сутність.

Shop кагеторія товару так і залишилась окремою сутність, але з додатковою таблицею shop_address.

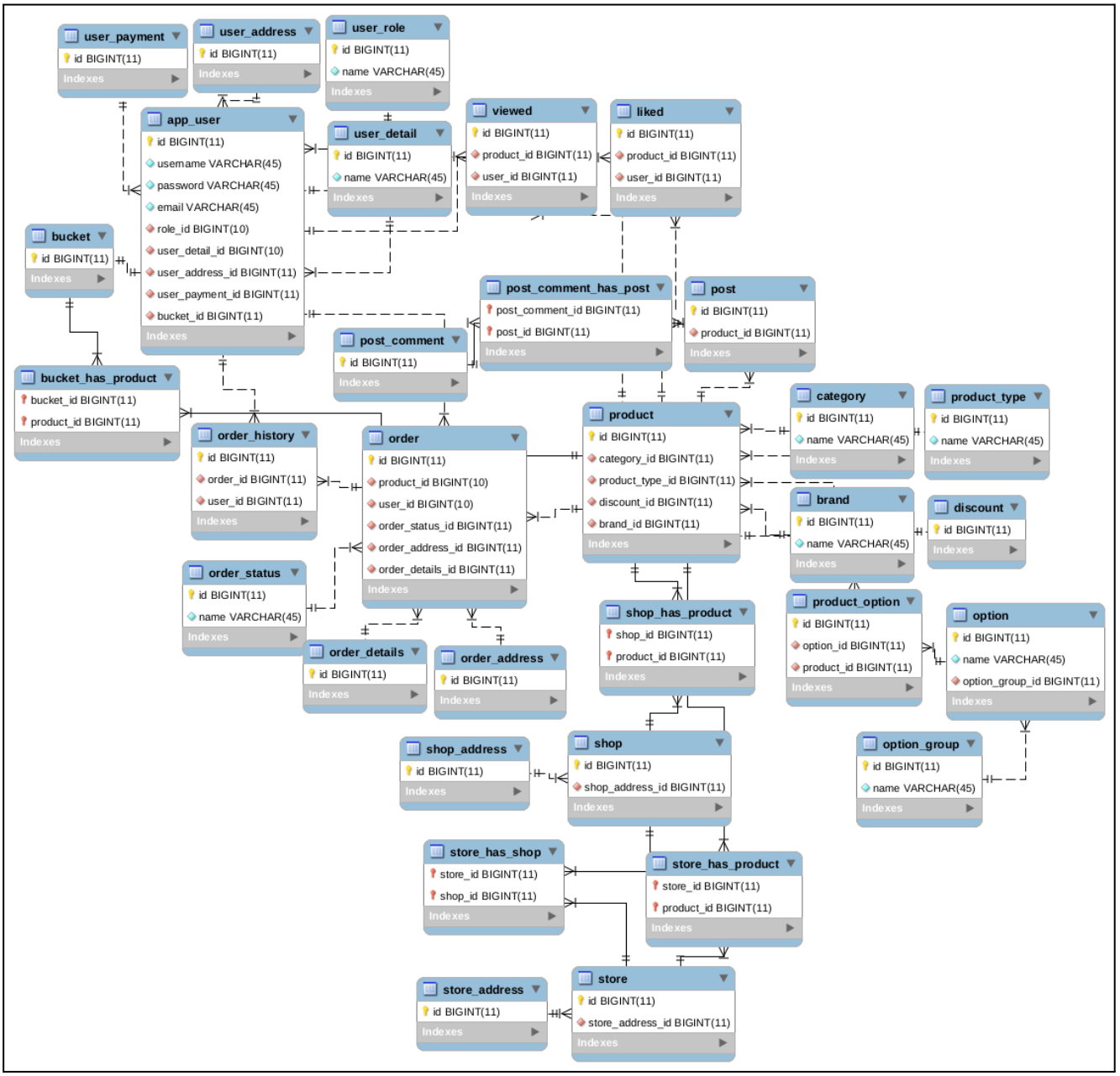

Рисунок 2.9 – Схема бази даних (Рисунок виконаний самостійно)

Store кагеторія товару так і залишилась окремою сутність, але з додатковою таблицею store_address.

2.4 Проектування архітектури фреймворку

Враховуючи кращі практики та предметну галузь була обрана мікросервісна архітектура з використанням ESB сервісу як серві проксі.

Це дозволить використовувати один и той же бекенд для декількох клієнтів. Всі компоненти системи спілкуються через HTTP протокол передачі данних. Діаграмма розгортання фреймворку для одного тенанту представлена на рисунку 2.10.

Основна задача розробити архітектуру для зручного використання як бізнесу так і кліентів. Згідно зі статистикою, більше половини населення України користується смартфонами, водночас близько 30% українців ніколи не користувалися персональними комп'ютерами. Користувачі смартфонів проводять більшу частину свого часу саме у мобільних додатках. Тому більшість покупок та різних операцій проводяться за допомогою смартфона та додатків, які дають доступ до нових можливостей.

Мобільний додаток зручніше та швидше. Зазвичай, розробники додатків дбають про те, щоб клієнти не витрачали багато часу на здійснення необхідної операції. Користувачеві залишається тільки зайти, ознайомитися з послугою та здійснити потрібну операцію. Потрібні покупки або оформлення кредитів здійснюються в кілька кліків.

Персоналізовані повідомлення. Програма може надсилати вам повідомлення, нагадувати про нові акції, знижки, послуги. Перевага в тому, що подібні повідомлення надходять відразу на мобільний телефон, де користувач протягом декількох хвилин може з ним ознайомитися.

Знижки та акції. Користувачам мобільних додатків найчастіше надаються різноманітні знижки або дають можливість взяти участь у цікавих акціях. Крім того, ви обов'язково дізнаєтесь про це, оскільки отримаєте повідомлення.

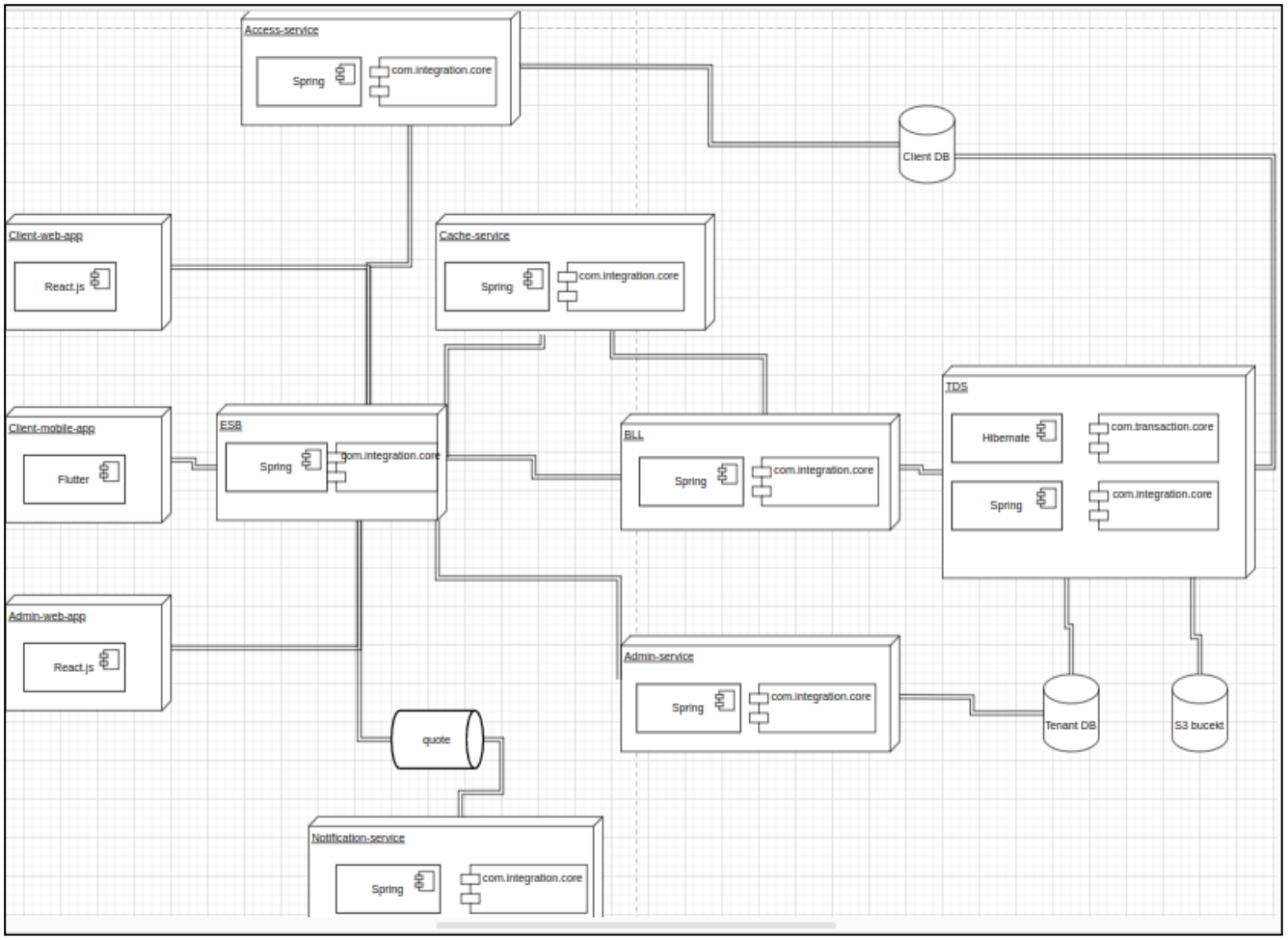

Рисунок 2.10 – Діаграма розгортання фреймворку для одного тенанту (Рисунок виконаний самостійно)

Також кліент бізнесу повинен мати свій відокремлений доступ для того щоб мати змогу змінювати за своїм бажанням бізнес модель(додавання товар, проводити акції, змінна цін).

Для цього є відокремленний веб модуль що дозволить це робити адміністратору бізнесу. Діаграма розгортання UI-сервісів з ESB представлена на рисунку 2.11.

Основна частина архітектури фреймверка знаходиться у цих двух сервісів TDS transaction data service та BLL bisness logic layer.

Задача TDS сервіс для роботи з базою даних кліентів та конфігурацією бізнеса усі запроси з базою повинні проходити через цей сервіс.

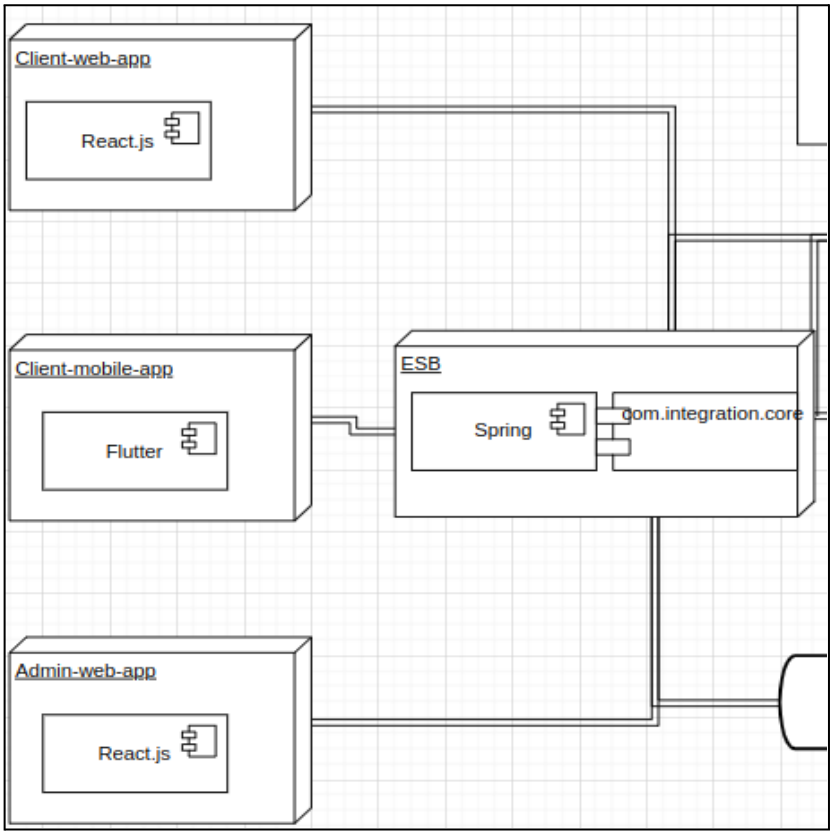

Рисунок 2.11 – Діаграма розгортання UI-сервісів з ESB (Рисунок виконаний самостійно)

Задача BLL сервіс містить основну логіку бізнес клієнта в ньому міститься робота з корзиною покупця, перегляд товарів, перегляд відгуків. Діаграма розгортання BLL та TDS сервісів з клієнською та конфігураційною БД представлена на рисунку 2.12.

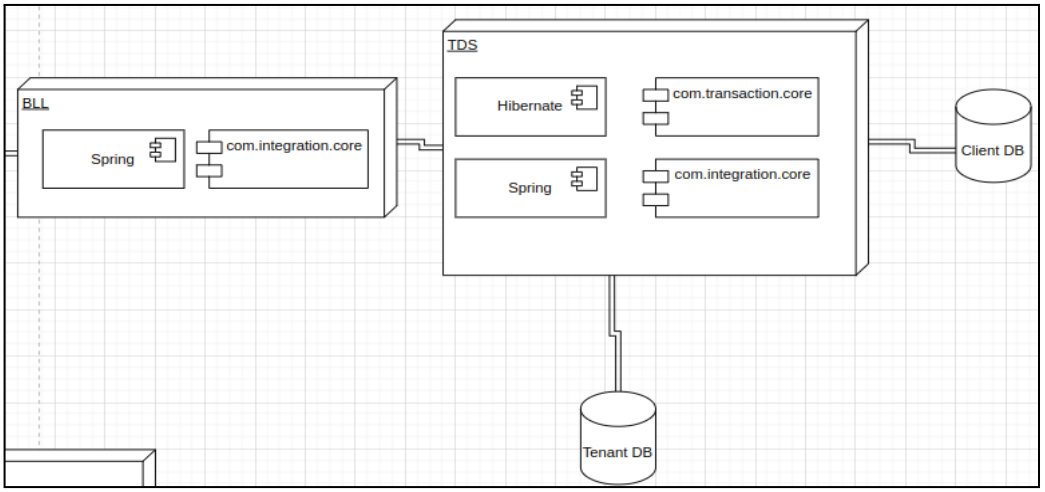

Рисунок 2.12 – Діаграма розгортання BLL та TDS сервісів з клієнською та конфігураційною БД (Рисунок виконаний самостійно)

Access-service містить логіку для аутентифікації та регестрації у облікової даних. Діаграма розгортання access-service та його компонентів представлена на рисунку 2.13.

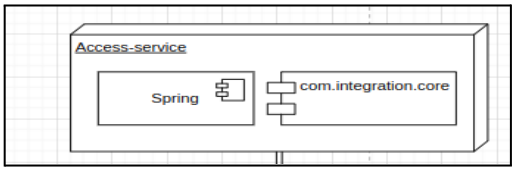

Рисунок 2.13 – Діаграма розгортання access-service та його компонентів (Рисунок виконаний самостійно)

Cache-service містить основну логіку кешування запросів клієнтів, що дозволить зменьшити час запросів до БД. Діаграма розгортання cache-service та його компонентів представлена на рисунку 2.14.

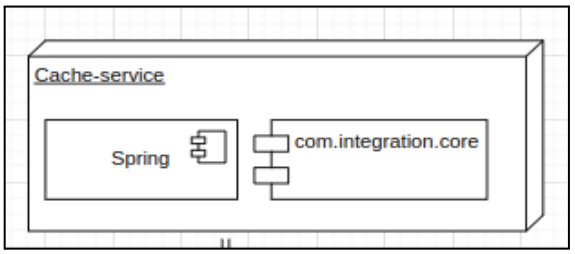

Рисунок 2.14 – Діаграма розгортання cache-service та його компонентів (Рисунок виконаний самостійно)

Notification-service містить логіку відправки імейлів, або повідомлень у мобільний додаток за використанням черги що дозволить планово оброблювати повідомлень. Діаграма розгортання notification-service та його компонентів представлена на рисунку 2.15.

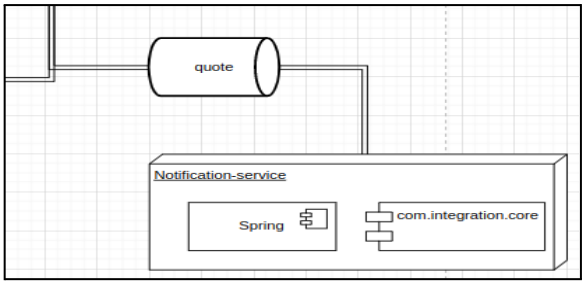

Рисунок 2.15 – Notification-service та його компоненти (Рисунок виконаний самостійно)

Здача розробки фреймворки складна, тому треба розділити її на розроблені блоки. Умновно саму розробку фреймфорку можно було б поділити за логікою розробки модулів, це дозволило б розпаралелити процесс розробки між командами розробки розробниками. В цілому розробка складалась з таких пунктів:

- створення бази данних товарів;
- створення бази данних клієнта магазина;
- створення модулю інтеграційних моделій(integration.core);
- створення модулю роботи з базою данних(cos-sdk);
- створення ядра сервісу роботи з БД та S3 бакетом(tds-core);
- створення ядра для сервісу клієнської бізнес логіки інтернет магазину(bll-core);
- створення ядра сервісу адміністрування інтернет магазину(ops-core);
- створення ядра сервісу авторизації та аутентифікації клієнтів (autentification-core);
- створення ядра сервісу роботи клієнськими повідомленнями(nonification-core);
- створення ядра сервісу кешування(cache-core);
- створення ядра декоратора запросів(esb-core);
- створення ядра для десктопного клієнт сервісу(web-core);
- створення ядра для мобільного клієнт сервісу(mobile-core);
- створення ядра для адмінського клієнт сервісу(admin-view-core);
- створення механізмів CI/CD для кожного модуля.

2.5 Планування експериментального дослідження

При розробці модулів фреймворку можно зауважити постійну залежності від API, вкрай важливо оптимізувати їх роботу і уникнути повільного відгуку, що може суттєво утруднити роботу користувачів.

Хороша продуктивність API гарантує, що програми які їх використовують, зможуть забезпечити безперебійну роботу користувачів. Низька продуктивність

може призвести до повільного часу відгуку, зниження функціональності або навіть збоїв у роботі програми, що призведе до невдоволення користувачів та втрати клієнтів.. Вирішення проблем затримки та підвищення пропускної спроможності вимагає реалізації ефективних алгоритмів та структур даних, а також оптимізації мережевих протоколів та інфраструктури. Це вимагає від розробників глибокого розуміння базових систем та властивих їм обмежень.

Підвищення продуктивності API вимагає багатогранного підходу, який охоплює різні аспекти життєвого циклу розробки програмного забезпечення, від проектування та реалізації до моніторингу та оптимізації.

Мета даного дослідження - висвітлити основні методи та кращі практики для підвищення продуктивності API, які тим самим забезпечують швидкий час відгуку для користувача(заміри будуть робитися у мілісекундах).

Ретельно вивчивши базову архітектуру, розумно використовуючи механізми кешування, оптимізуючи навантаження запитів та відповідей, розробники зможуть по-справжньому використати потенціал API для створення гнучких та ефективних цифрових платформ.

Таким чином, була поставлені наступні задача:

- провести аналіз існуючих методів оптимізації інтеграції сервісів;
- спланувати експериментальну частину дослідження, що включає розробку сервісів та інтеграцій між ними;
- провести експериментальне дослідження методів обробки запитів у мікросервісній архітектурі.

Проаналізовані наступні методи оптимізації інтеграцій між сервісами при розробці фреймворку для e-commerce рішень[12].:

- використання ефективних алгоритмів та структур даних для зниження складності обчислень, а також багатопоточної обробки запитів;
- балансування навантаження: розподіл вхідних API-запитів між кількома серверами дозволяє усунути вузькі місця та забезпечити стабільну продуктивність у періоди підвищеного попиту;

- кешування: впровадження стратегій кешування, таких як прикордонне кешування або кешування на рівні програми, може значно скоротити час відгуку за рахунок мінімізації надлишкового пошуку та обробки даних.

В ході експериментального дослідження було розроблено декілька сервісів, які були застосовані у розробці фреймворку для e-commerce рішень (див. рис. 2.10). Розглянемо прийняті рішення:

- інтеграція проходить через ESB сервіс, він і є декоратором для зовнішнього світу; тому запити робляться через нього до TDS сервісу;
- деякі запити для замірів робляться через сервіс кешування або на пряму через Bisness logic layer сервіс;
- методи оптимізації сервісів досліджувалися шляхом порівняння продуктивності запитів до БД через API; база даних розгорнута у Postgresql.

Для тестування нагрузки можно наповнити базу даних на 10 тисяц записів.

Для заміру продуктивності запитів будуть використовані такі запроси до бази:

- запит на дані користувача та корзину користувача (select u.id, u.username, u.password, u.email, b.id from app_user u join bucket b);
- запит на дані користувача та замовлення користувача (select u.id, u.username, u.password, u.email, oh.dataOfOrder from app_user u join order_item oi join order_history oh on oh.order_id = oh.user_id).

3 ОПИС ПРОГРАМНОЇ РЕАЛІЗАЦІЇ

3.1 Вибір засобів програмної реалізації

Система модулів, які є собою ядрами як мікросервісного так і монолітного рішення в залежності від потреб клієнта.

Модулі створені за допомгою фреймворку Spring, яка базується на Java, що забезпечує стійкість роботи системи, високу швидкість роботи та забезпечити легку горизонтальну масштабованість.

В якості бази даних була обрана PostgreSQL, яка забезпечує високу продуктивність обробки даних.

Модулі клієнтської частина створена за допомогою ядер фреймворку що надає змогу повторного використання компонентів, що в свою чергу надає гнучкості системі, та оптимізує її роботу.

За необхідністю серверну частину програмної системи можна розгорнути в Docker контейнері, для спрощення розгортання сервісу на різноманітних платформах та використовувати AWS як хмарне рішення для розгортання сервісів.[13]

AWS (Amazon Web Services):

- найбільший і найдосвідченіший провайдер хмарних послуг з великим набором сервісів та рішень.
- широке географічне охоплення дата-центрами, що дозволяє забезпечити високу доступність та надійність системи.
- велика кількість сервісів для обчислення, зберігання, баз даних, мереж та аналітики, що дозволяє гнучко побудувати та масштабувати систему;
- широкий спектр інструментів для безпеки, моніторингу та управління ресурсами.

Azure (Microsoft Azure):

- міцна інтеграція з існуючими технологіями Microsoft, такими як Windows Server, Active Directory та SQL Server;
- гнучкі моделі ціноутворення та планування ресурсів, що дозволяють оптимізувати витрати;

- широкий набір сервісів для обчислення, зберігання, баз даних, мереж, штучного інтелекту та аналітики;
- сильні інструменти для розгортання, управління та моніторингу ресурсів.

Google Cloud (Google Cloud Platform):

- сильний фокус на інтелектуальних послугах, машинному навчанні та штучному інтелекті;
- гнучкі моделі ціноутворення з великими можливостями для розміщення та обробки даних;
- широкий набір сервісів для обчислення, зберігання, баз даних, мереж та аналітики;
- сильні інструменти для розгортання, управління та моніторингу ресурсів.

Загалом, усі три провайдери мають свої переваги і підходять для розгортання систем електронної комерції. Але для розробки фреймоврку вибір припав на AWS.[13]

3.2 Опис фізичної моделі БД

Основний опис фізичної моделі БД вже наведений у пункті 3.3 Аналіз, моделювання предметної області та розробка схеми бази даних.

Скрипт таблиць основних сутностей БД:

- створення таблиці корзини покупця:

```
DROP TABLE IF EXISTS `mydb`.`bucket` ;
CREATE TABLE IF NOT EXISTS `mydb`.`bucket` (
  `id` BIGINT(11) NOT NULL AUTO_INCREMENT,
  PRIMARY KEY (`id`))
ENGINE = InnoDB;
```

- створення таблиці користувача:

```
DROP TABLE IF EXISTS `mydb`.`app_user` ;
CREATE TABLE IF NOT EXISTS `mydb`.`app_user` (
  `id` BIGINT(11) NOT NULL,
  `username` VARCHAR(45) NOT NULL,
  `password` VARCHAR(45) NOT NULL,
  `email` VARCHAR(45) NOT NULL,
```

```
  `role_id` BIGINT(10) NOT NULL,
  `user_detail_id` BIGINT(10) NOT NULL,
  `user_address_id` BIGINT(11) NOT NULL,
  `user_payment_id` BIGINT(11) NOT NULL,
  `bucket_id` BIGINT(11) NOT NULL,
  PRIMARY KEY (`id`),
  INDEX `fk_user_role1_idx` (`role_id` ASC) VISIBLE,
  INDEX `fk_user_user_detail1_idx` (`user_detail_id` ASC) VISIBLE,
  INDEX `fk_user_user_address1_idx` (`user_address_id` ASC) VISIBLE,
  INDEX `fk_user_user_payment1_idx` (`user_payment_id` ASC) VISIBLE,
  INDEX `fk_user_bucket1_idx` (`bucket_id` ASC) VISIBLE,
  CONSTRAINT `fk_user_role1`
    FOREIGN KEY (`role_id`)
    REFERENCES `mydb`.`user_role` (`id`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION,
  CONSTRAINT `fk_user_user_detail1`
    FOREIGN KEY (`user_detail_id`)
    REFERENCES `mydb`.`user_detail` (`id`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION,
  CONSTRAINT `fk_user_user_address1`
    FOREIGN KEY (`user_address_id`)
    REFERENCES `mydb`.`user_address` (`id`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION,
  CONSTRAINT `fk_user_user_payment1`
    FOREIGN KEY (`user_payment_id`)
    REFERENCES `mydb`.`user_payment` (`id`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION,
  CONSTRAINT `fk_user_bucket1`
    FOREIGN KEY (`bucket_id`)
    REFERENCES `mydb`.`bucket` (`id`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION)
ENGINE = InnoDB;
```

– створення таблиці категорії товару:

```
DROP TABLE IF EXISTS `mydb`.`category` ;
CREATE TABLE IF NOT EXISTS `mydb`.`category` (
  `id` BIGINT(11) NOT NULL AUTO_INCREMENT,
  `name` VARCHAR(45) NOT NULL,
  PRIMARY KEY (`id`))
ENGINE = InnoDB;
```

– створення таблиці товару:

```
DROP TABLE IF EXISTS `mydb`.`product` ;
CREATE TABLE IF NOT EXISTS `mydb`.`product` (
  `id` BIGINT(11) NOT NULL AUTO_INCREMENT,
```

```
  `category_id` BIGINT(11) NOT NULL,
  `product_type_id` BIGINT(11) NOT NULL,
  `discount_id` BIGINT(11) NOT NULL,
  `brand_id` BIGINT(11) NOT NULL,
  PRIMARY KEY (`id`),
  INDEX `fk_product_category1_idx` (`category_id` ASC) VISIBLE,
  INDEX `fk_product_type1_idx` (`product_type_id` ASC) VISIBLE,
  INDEX `fk_product_discount1_idx` (`discount_id` ASC) VISIBLE,
  INDEX `fk_product_brand1_idx` (`brand_id` ASC) VISIBLE,
  CONSTRAINT `fk_product_category1`
    FOREIGN KEY (`category_id`)
    REFERENCES `mydb`.`category` (`id`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION,
  CONSTRAINT `fk_product_type1`
    FOREIGN KEY (`product_type_id`)
    REFERENCES `mydb`.`product_type` (`id`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION,
  CONSTRAINT `fk_product_discount1`
    FOREIGN KEY (`discount_id`)
    REFERENCES `mydb`.`discount` (`id`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION,
  CONSTRAINT `fk_product_brand1`
    FOREIGN KEY (`brand_id`)
    REFERENCES `mydb`.`brand` (`id`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION)
ENGINE = InnoDB;
```

– створення таблиці магазину:

```
DROP TABLE IF EXISTS `mydb`.`shop` ;
CREATE TABLE IF NOT EXISTS `mydb`.`shop` (
  `id` BIGINT(11) NOT NULL AUTO_INCREMENT,
  `shop_address_id` BIGINT(11) NOT NULL,
  PRIMARY KEY (`id`),
  INDEX `fk_shop_shop_address1_idx` (`shop_address_id` ASC) VISIBLE,
  CONSTRAINT `fk_shop_shop_address1`
    FOREIGN KEY (`shop_address_id`)
    REFERENCES `mydb`.`shop_address` (`id`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION)
ENGINE = InnoDB;
```

– створення таблиці замовлення:

```
DROP TABLE IF EXISTS `mydb`.`order` ;
CREATE TABLE IF NOT EXISTS `mydb`.`order` (
  `id` BIGINT(11) NOT NULL AUTO_INCREMENT,
  `product_id` BIGINT(10) NOT NULL,
```

```
        `user_id` BIGINT(10) NOT NULL,
        `order_status_id` BIGINT(11) NOT NULL,
        `order_address_id` BIGINT(11) NOT NULL,
        `order_details_id` BIGINT(11) NOT NULL,
        PRIMARY KEY (`id`),
        INDEX `fk_order_product1_idx` (`product_id` ASC) VISIBLE,
        INDEX `fk_order_user1_idx` (`user_id` ASC) VISIBLE,
        INDEX `fk_order_order_status1_idx` (`order_status_id` ASC) VISIBLE,
           INDEX  `fk_order_order_address1_idx`  (`order_address_id`  ASC)
VISIBLE,
           INDEX  `fk_order_order_details1_idx`  (`order_details_id`  ASC)
VISIBLE,
        CONSTRAINT `fk_order_product1`
          FOREIGN KEY (`product_id`)
          REFERENCES `mydb`.`product` (`id`)
          ON DELETE NO ACTION
          ON UPDATE NO ACTION,
        CONSTRAINT `fk_order_user1`
          FOREIGN KEY (`user_id`)
          REFERENCES `mydb`.`app_user` (`id`)
          ON DELETE NO ACTION
          ON UPDATE NO ACTION,
        CONSTRAINT `fk_order_order_status1`
          FOREIGN KEY (`order_status_id`)
          REFERENCES `mydb`.`order_status` (`id`)
          ON DELETE NO ACTION
          ON UPDATE NO ACTION,
        CONSTRAINT `fk_order_order_address1`
          FOREIGN KEY (`order_address_id`)
          REFERENCES `mydb`.`order_address` (`id`)
          ON DELETE NO ACTION
          ON UPDATE NO ACTION,
        CONSTRAINT `fk_order_order_details1`
          FOREIGN KEY (`order_details_id`)
          REFERENCES `mydb`.`order_details` (`id`)
          ON DELETE NO ACTION
          ON UPDATE NO ACTION)
      ENGINE = InnoDB;
```

– створення таблиці складу:

```
      DROP TABLE IF EXISTS `mydb`.`store` ;

      CREATE TABLE IF NOT EXISTS `mydb`.`store` (
        `id` BIGINT(11) NOT NULL AUTO_INCREMENT,
        `store_address_id` BIGINT(11) NOT NULL,
        PRIMARY KEY (`id`),
           INDEX  `fk_store_store_address1_idx`  (`store_address_id`  ASC)
VISIBLE,
        CONSTRAINT `fk_store_store_address1`
          FOREIGN KEY (`store_address_id`)
          REFERENCES `mydb`.`store_address` (`id`)
          ON DELETE NO ACTION
          ON UPDATE NO ACTION)
      ENGINE = InnoDB;
```

– створення таблиці коментарю:

```
DROP TABLE IF EXISTS `mydb`.`post` ;

CREATE TABLE IF NOT EXISTS `mydb`.`post` (
  `id` BIGINT(11) NOT NULL AUTO_INCREMENT,
  `product_id` BIGINT(11) NOT NULL,
  PRIMARY KEY (`id`),
  INDEX `fk_post_product1_idx` (`product_id` ASC) VISIBLE,
  CONSTRAINT `fk_post_product1`
    FOREIGN KEY (`product_id`)
    REFERENCES `mydb`.`product` (`id`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION)
ENGINE = InnoDB;
```

3.3 Опис реалізації модулів фреймворку

3.3.1 Реалізація модуля “tds.core”

Модуль “tds.core” – це модуль для роботи з базою даних. Сам модуль є декоратором для роботи з БД та S3 бакетом. Розглянемо діаграму розгортання клієнського TDS сервісу з використанням внутрішніх бібліотек, як взаємодіють між собою у реальному середовищі на рисунку 3.1.

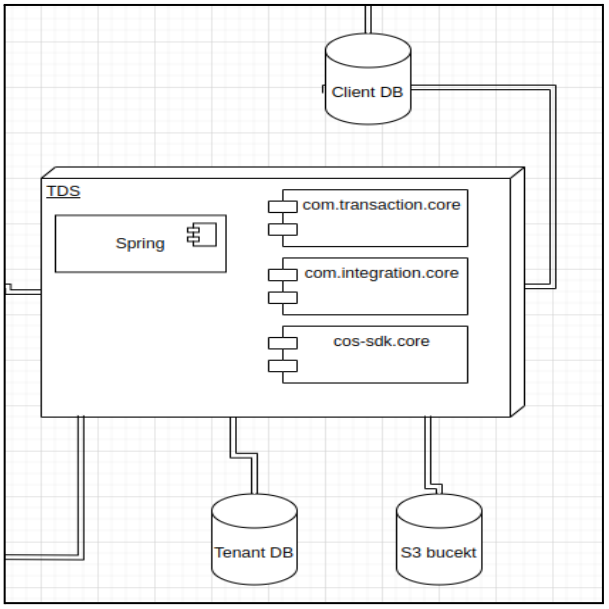

Рисунок 3.1 – Діаграма розгортання клієнського TDS сервісув з декількома БД та S3 бакетом (Рисунок виконаний самостійно)

Як можна зазначити з діаграми попре опенсорсної бібліотеки Spring для розробки клієнського сервісу були використані власні бібліотеки такі як:

- com.integration.core: бібліотека інтеграційних моделей, містить в собі обьекти БД, обьекти запитів та відповідей API та запити до БД. Використовується як клієнське рішення для проектування БД та змінюється від вимог клієнта;
- cos-sdk.core: бібліотека роботи пулом з'єднань до БД за допомогою ORM фреймворка Hibernate, має декілька варіантів пулі(hikari, c3p0, dbcp2, vibur). Не залежить від клієнської логіки та має основну логіку з створенням, видаленням, видаленням обьектів за БД, основан на дженерік підході;
- transaction.core: модуль має основну логіку для роботи сущностями БД реалізує DAO паттерн. Реалізує механізм транзакцій та пропонує готові сервіси для використаня у клієнському коді.

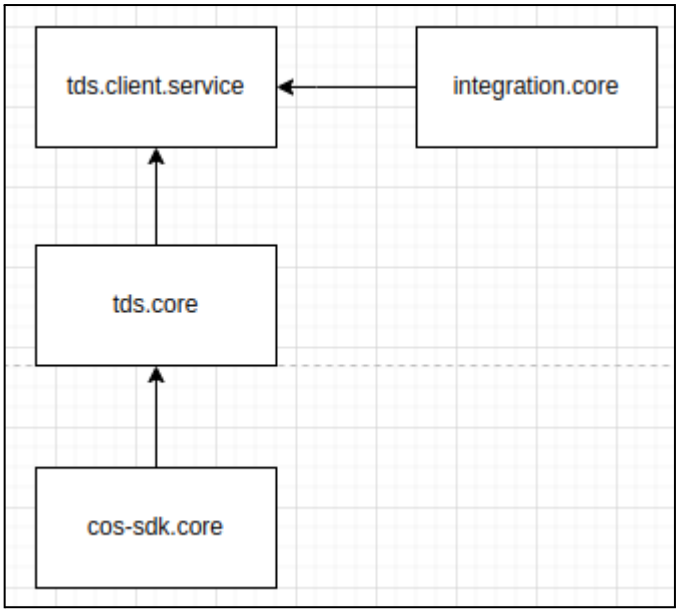

Рисунок 3.2 – Діагрма відносин бібліотек у TDS сервісі (Рисунок виконаний самостійно)

З цієї діаграми можно побачити що сам клієнський сервіс буде використовувати 2 основні бібліотеки(integration.core та tds.core), cos-sdk.core вже є частиною tds.core. Клієнському коду залишиться лише викликати сервіси з

модулю tds.core. Попри все сам клієнт або OPS розробник повинен подбати про інтеграційну модель та заповнити свою версію полів для обьектів бібліотеки integration.core.

3.3.2 Реалізація модуля “cache.core”

Модуль “cache.core” – це модуль для роботи механізмом кешування даних. Розсмотремо діаграму розгортання клієнського cache сервісу з використанням внутрішніх бібліотек, як взаємодіють між собою у реальному середовищі на рисунку 3.3.

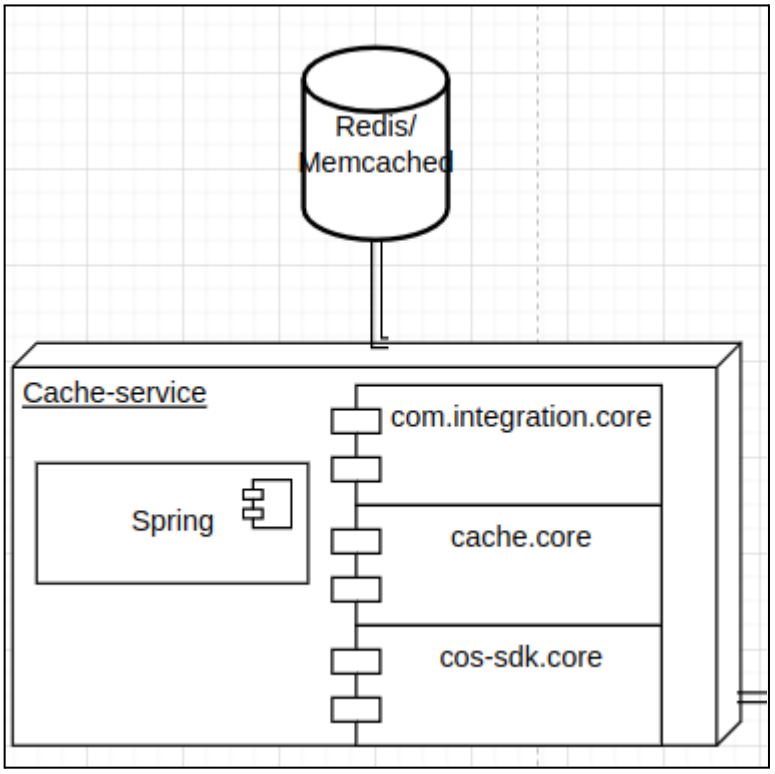

Рисунок 3.3 – Діаграма відносин бібліотек у TDS сервісі (Рисунок виконаний самостійно)

Як можна зазначити з діаграми попре опенсорсної бібліотеки Spring для розробки клієнського сервісу були використані власні бібліотеки такі як:

- com.integration.core: бібліотека інтеграційних моделей, як клієнське рішення для проектування БД та змінюється від вимог клієнта;
- cos-sdk.core: бібліотека роботи пулом з’єднань до БД, як Redis.
- cache.core: модуль має основну логіку механізму кешування.

З цієї діаграми можно побачити що сам клієнський сервіс буде використовувати 2 основні бібліотеки(integration.core та cache.core), cos-sdk.core вже є частиною cache.core. Клієнському коду залишиться лише викликати сервіси з модулю cache.core. Попри все сам клієнт або OPS розробник повинен подбати про інтеграційну модель та заповнити свою версію полів для обьектів бібліотеки integration.core.

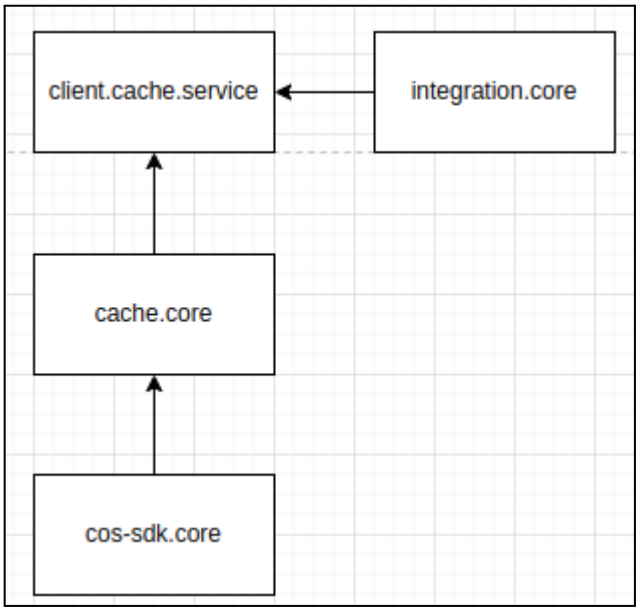

Рисунок 3.4 – Діаграма відносин бібліотек у cache сервісі (Рисунок виконаний самостійно)

4 РЕЗУЛЬТАТИ ЕКСПЕРИМЕНТАЛЬНОГО ДОСЛІДЖЕННЯ

4.1 Результати дослідження продуктивности фреймворків

У ході проведеного дослідження було досліджено основні методи та практики для підвищення продуктивності API. Зокрема, були розглянуті такі методи оптимізації:

- використання ефективних алгоритмів та структур даних для зниження складності обчислень та багатопоточної обробки запитів;
- балансування навантаження, що дозволяє розподіляти вхідні API-запити між кількома серверами для усунення вузьких місць та забезпечення стабільної продуктивності;
- кешування, включаючи прикордонне кешування та кешування на рівні програми, що допомагає скоротити годину відкликання шляхом мінімізації надлишкового пошуку та обробки даних;
- в результаті експериментального дослідження було розроблено сервіси, які були використані у фреймворку для e-commerce рішень. Було використано інтеграцію через ESB сервіс, кешування запитів та реактивне програмування для оптимізації сервісів.

На основі розробленого фреймворку було проведено експерименти (див. рис. 4.1), які дозволили отримати наступні результати:

- перший виклик базового API сервісу тривав 219ms;
- багаторазовий виклик базового API тривав сервісу 21ms;
- перший виклик сервісу з багатопоточною оптимізацією тривав 21ms;
- багаторазовий виклик сервісу з багатопоточною оптимізацією тривав 7ms;
- перший виклик сервісу з механізмом кешування тривав 427ms;
- багаторазовий виклик сервісу з механізмом кешування тривав 4ms.

На діаграмі показаний середній час у мілісекундах витрачений у різних експерементальних ситуаціях.

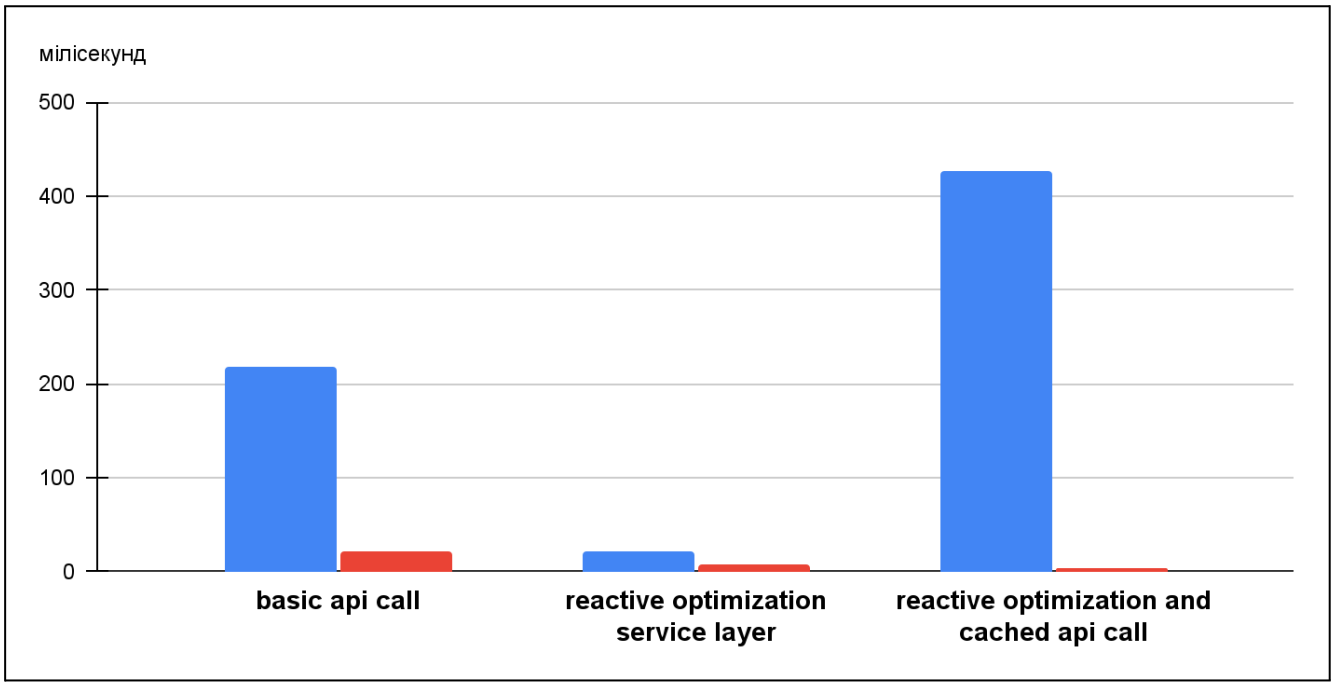

Рисунок 4.1 – Діаграма результатів експериментів першого та багаторазових викликів системи у декількох варіантах оптимізації. (Рисунок виконаний самостійно)

За результатами проведених експериментів було зроблено такі висновки:

- кешування запитів є затратною операцією, але дозволяє значно зменшити годину відкликання, досягаючи 16-кратного зменшення часу в порівнянні зі звичайним GET HTTP запитом;
- використання реактивного програмування у фреймворку дало приріст у швидкості обробки запиту в 5 разів порівняно з звичайним запитом.

Отже, використання цих методів оптимізації може допомогти покращити продуктивність API та забезпечити швидку годину відкликання для користувачів.

4.2 Опис рекомендацій

Оптимізація продуктивності роботи сервісів має першорядне значення для забезпечення ефективного та безперебійного обслуговування користувачів. Проведені експерименти дозволили сформулювати такі рекомендації щодо підвищення продуктивності API:

- реалізація розумних стратегій кешування дозволяє тимчасово зберігати дані, що часто використовуються, тим самим знижуючи час очікування за рахунок мінімізації надлишкового пошуку та обробки даних;
- кешування має сенс використовувати лише у багатьох повторних запитах, у найпростіших запитах краще використовувати звичайний запит до сервісу;
- використання методу PATCH для часткових оновлень може значно підвищити продуктивність.

ВИСНОВКИ

В ході виконання кваліфікаційної роботи проаналізована предмета область та варіанти рішень по ринку з аналізом їх переваг та недоліків, проаналізовані основні підходи до розробки фреймворків та аналіз архітектурних рішень.

Вибран архітектурне підхід при розробці фреймворку, вибрані засоби програмної реалізації, були виділені основні модулі фреймворку та їх взаємодію, та описан функціонал програмної системи.

З метою покращення програмного рішення при розробці нового фреймворку для e-commerce прикладної області, запровадити нові технології такі як: SPA, Micro Frontends, query caching, mobile solution одночасно разом з розробкою web версії продукту.

Фреймворк диктує правила для команди розробників роботи з ним, що дає змогу зробити розробку більш зрозумілою для кожного члена команди.

Заощадити час клієнта та команди розробки у побудові свого клієнтського рішення, так як увесь основний функціонал буде вже наданий та декарований в декілька модулів.

Дає можливість розділити розробку між дизайнерами, back-end та front-end розробниками, mobile командою та девопс роботою (розгортанням у хмарі або локальних серверах).

В ході розробки фреймворку було виконано:

- проведен аналіз предметної та існуючих технологій в області роботи інтернет-магазинів;
- проведена моделювання предметної області та розробити схему бази даних;
- розроблена архітектура фреймворку;
- проведено дослідження продуктивності отриманих програмних рішень та сформулювати рекомендації.

В ході виконання кваліфікаційної роботи було складено план експериментального дослідження, виявленні основні метрики якості, а саме:

- рівномірність розподілення;

- відповідність операцій даних для сегментів;
- час інтеграції в існуючу структуру даних;
- час обчислювальних затрат;
- час очікування.

Було проведено замір кожної з метрик для 2-х баз даних на 3-х різних наборах даних (великий об’єм 1200 Мб, середній 600 Мб та малий 100 Мб). Було впроваджено скрипти для інтеграції в існуючу структуру даних.

Були отримані результати дослідження та сформована оцінка якості з рекомендаціями стосовно оптимізації продуктивності роботи сервісів.

За результатами дослідження було опубліковано тези IV Міжнародна науково-практична конференції молодих вчених та студентів «Інженерія програмного забезпечення і передові інформаційні технології (SoftTech-2023)» присвяченої 125-й річниці КПІ ім. Ігоря Сікорського [21] (див. додаток Г).

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Software Architecture: The Hard Parts: Modern Trade-Off Analyses for Distributed Architectures – Neal Ford, Mark Richards, Pramod Sadalage, Zhamak Dehghani.

2. Fundamentals of Software Architecture: An Engineering Approach – Mark Richards and Neal Ford.

3. Моделі і методи проектування інформаційних систем URL: https://elearning.sumdu.edu.ua/free_content/lectured:de1c9452f2a161439391120eef364dd8ce4d8e5e/20160217112601/170352/index.html#p17 (дата звернення: 25.03.2023).

4. Khovrat A., Kobziev V., Nazarov A., Yakovlev S. Parallelization of the VAR Algorithm Family to Increase the Efficiency of Forecasting Market Indicators During Social Disaster. CEUR Workshop Proceedingsthis, 2022, 3347, pp. 222–233.

5. P. Nichols, Social Survey Methods: A field guide for development workers. Development Guidelines, No. 6, Oxfam GB, 2000.

6. Mazurova, O. Research of ACID transaction implementation methods for distributed databases using replication technology / Mazurova, O., Naboka, A., Shirokopetleva, M. Innovative technologies and scientific solutions for industries, (2(16), pp. 19-31. Doi: 10.30837/ITSSI.2021.16.019).

7. The BUILD Framework: A Heart-Based System for Personal and Professional Growth Paperback – January 22, 2019 by John Peitzman

8. Build a Frontend Web Framework by Ángel Sola Orbaiceta, February 2023, (ISBN 9781633438064)

9. Spring Microservices in Action First Edition by John Carnell, Feb 2005, (ISBN-101617293989)

10. Build Your Own Framework: https://vite-plugin-ssr.com/build-your-own-framework(дата звернення: 25.03.2023).

11. Creating a custom framework: https://docs.aws.amazon.com/audit-manager/latest/userguide/custom-frameworks.html(дата звернення: 25.03.2023).

12. Bell, Michael. "Introduction to Service-Oriented Modeling". Service-Oriented Modeling: Service Analysis, Design, and Architecture. Wiley & Sons. 2006, pp. 3. ISBN 978-0-470-14111-3.

13. How to Create Your Own Framework: https://ideas.repec.org/h/spr/sprchp/978-3-030-36838-8_11.html(дата звернення: 25.03.2023).

14. Build your own web framework: https://vercel.com/blog/build-your-own-web-framework(дата звернення: 25.03.2023).

15. how to build your own JS framework: https://mikeguoynes.medium.com/part-1-build-your-own-js-framework-from-scratch-f4e35d0dffa6(дата звернення: 25.03.2023).

16. Architectural Programming: https://www.forakergroup.org/predevelopment/resources/architectural-programming/#:~:text=What%20is%20it%3F,of%20work%20to%20be%20designed(дата звернення: 25.03.2023).

17. Architectural Programming: https://www.wbdg.org/design-disciplines/architectural-programming(дата звернення: 25.03.2023).

18. Software architecture https://en.wikipedia.org/wiki/Software_architecture(дата звернення: 25.03.2023).

19. Software Architecture & Design Introduction https://www.tutorialspoint.com/software_architecture_design/introduction.htm(дата звернення: 25.03.2023).

20. Build Your Own Frontend Framework from the Ground Up https://freecontent.manning.com/build-your-own-frontend-framework-from-the-ground-up/(дата звернення: 25.03.2023).