Міністерство освіти і науки України
Харківський національний університет радіоелектроніки

Навчально-науковий центр заочної форми навчання
(повна назва)
Кафедра Інформаційних управляючих систем
(повна назва)

КВАЛІФІКАЦІЙНА РОБОТА
Пояснювальна записка

рівень вищої освіти другий (магістерський)

Дослідження моделей даних
для маркетплейсів
(тема)

Виконав:
студент _2_ курсу, групи _УПГІТзм-21-1__
Максим СІЛЕНКО
(власне ім'я, прізвище)

Спеціальність 122 Комп'ютерні науки__
(код і повна назва спеціальності)
Тип програми освітньо-наукова
(освітньо-професійна або освітньо-наукова)
Освітня програма _Управління проєктами в галузі інформаційних технологій
(повна назва освітньої програми)

Керівник доц. каф. ІУС Віталій БРУСЕНЦЕВ
(посада, власне ім'я, прізвище)

Допускається до захисту

Зав. кафедри _____ _Костянтин ПЕТРОВ__
(підпис) (власне ім'я, прізвище)

2023 р.

Харківський національний університет радіоелектроніки

Навчально-науковий центр заочної форми навчання

Кафедра Інформаційних управляючих систем

Рівень вищої освіти другий (магістерський)

Спеціальність 122 Комп'ютерні науки
(код і повна назва)

Тип програми освітньо-наукова
(освітньо-професійна або освітньо-наукова)

Освітня програма Управління проєктами в галузі інформаційних технологій
(повна назва)

ЗАТВЕРДЖУЮ:
Зав. кафедри _____
(підпис)
«_27_»__березня__ 20 23_ р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ

студентові Сіленку Максиму Сергійовичу
(прізвище, ім'я, по батькові)

1. Тема роботи Дослідження моделей даних для маркетплейсів

затверджена наказом університету від _03_ __квітня__2023_ р. № _87 Стз_
2. Термін подання студентом роботи до екзаменаційної комісії _15_ _травня_ 2023 р.
3. Вихідні дані до роботи 1) сфера використання: інструменти та платформи для управління даними
2) формати даних: xml, json, yaml, csv
3) архітектурний стиль: мікросервіси

4. Перелік питань, що потрібно опрацювати в роботі
1) аналіз проблеми
2) огляд існуючих форматів даних
3) огляд існуючих конверторів
4) огляд існуючих платформ управління даними
5) висновки

КАЛЕНДАРНИЙ ПЛАН

№	Назва етапів роботи	Терміни виконання етапів роботи	Примітка
1	Аналіз проблеми	27.03.23-01.04.23	Виконано
2	Огляд існуючих форматів даних	02.04.23-09.04.23	Виконано
3	Огляд існуючих конверторів	10.04.23-15.04.23	Виконано
4	Огляд існуючих платформ для маркетплейсів	16.04.23-21.04.23	Виконано
5	Дослідження мікросервісного та монолітного стилів	22.04.23-25.04.23	Виконано
6	Структуризація проекту	26.04.23-28.04.23	Виконано
7	Оформлення матеріалів кваліфікаційної роботи	29.04.23-04.05.23	Виконано
8	Подання кваліфікаційної роботи керівникові	05.05.23-09.05.23	Виконано
9	Захист кваліфікаційної роботи в ЕК	16.05.23	Виконано

Дата видачі завдання 27 березня 2023 р.

Студент _____
(підпис)

Керівник роботи _____ доц. каф. ІУС Віталій БРУСЕНЦЕВ
(підпис) (посада, власне ім’я, прізвище)

РЕФЕРАТ

Пояснювальна записка кваліфікаційної роботи містить: 86 сторінок, 26 рисунків, 1 таблицю, 3 лістинга, 21 джерело.

ІНФРАСТРУКТУРА, МАРКЕТПЛЕЙСИ, МІКРОСЕРВІС, МОДЕЛЬ, ПЛАТФОРМА, ФОРМАТ ДАНИХ, ХМАРНІ ТЕХНОЛОГІЇ

Метою кваліфікаційної роботи є дослідження різних моделей та форматів даних для маркетплейсів.

Даний вид роботи сприяє поглибленому аналізу та вивченню моделей і форматів даних, що використовуються на різних платформах, створених під маркетплейси.

В рамках роботи описано основні характеристики форматів даних, їх відмінності, архітектурні стилі розробки ПЗ, а також методи та технології що можуть бути використанні під час проектування системи управління форматами даних.

ABSTRACT

The explanatory note contains: 86 pages, 26 figures, 1 tables, 3 listings, 21 sources.

INFRASTRUCTURE, MANAGEMENT TOOLS, MICROSERVICES, MODEL, PLATFORM, DATA FORMAT, CLOUD TECHNOLOGIES

The purpose of the qualification work is discovering various models and data formats for marketpalces.

This type of work facilitates in-depth analysis and study of data models and formats used in different platforms that were created for marketplaces.

The work describes the main characteristics of data formats, their differences, architectural styles of software development, as well as methods and technologies that can be used when designing a data format management system.

ЗМІСТ

СКОРОЧЕННЯ ТА УМОВНІ ПОЗНАКИ

ПЗ – програмне забезпечення

ПО – предметна область

API – Application programming interface – описання методів за допомогою яких одна програма може комунікувати з іншою

AWS – Amazon Web Services – дочірня компанія Amazon, яка надає платформи хмарні на основі оплати за використання

B2B – Business-to-Business – вид взаємозв'язків учасниками яких є компанії

B2C – Business-to-Consumer – вид взаємозв'язків учасниками яких є компанія та фізичний споживач

C2C – Consumer-to-Consumer – вид взаємозв'язків учасниками яких є фізичні споживачі

CSV – Comma-Separated Values – файловий формат, котрий є відмежовувальним форматом для представлення табличних даних

IaaS – це модель обслуговування, яка дає можливість клієнту керувати обчислювальними ресурсами (анг., Infrastructure as a Service)

JSON – JavaScript Object Notation – об'єктна нотація JavaScript

PaaS – модель надання хмарних обчислень, при якій споживач отримує доступ до використання інформаційно-технологічних платформ (анг., Platform as a Service)

S3 – Simple Storage Service – послуга, яку пропонує AWS, яка забезпечує зберігання об'єктів через інтерфейс веб-служби

SaaS – це бізнес-модель розгортання та реалізації програмного забезпечення (анг., Software as a Service)

XML – Extensible Markup Language – розширювана мова розмітки, стандарт побудови мов розмітки ієрархічно структурованих даних для обміну між різними застосунками

ВСТУП

Прогрес не стоїть на місці, діджиталізація всіх сфер бізнесу стрімко набирає обороти. Це стосується і точок збуту товарів. Просування своєї продукції є однією з основних цілей маркетингового плану, в ній фіксуються цілі рекламних кампаній, базові принципи і підходи маркетингових комунікацій бренду, загальна довгострокова стратегія просування продукту на ринок. Маркетинг передбачає перенесення основного акценту з виробництва на проблеми споживача. Акцент на потреби споживача – це не тільки структурні і технологічні проблеми, а й серйозна організаційна, структурна, психологічна перебудова всієї роботи будь-якого підприємства. Тому збільшення кількості торгівельних площадок де можливо реалізувати продаж свого товару має дуже велику роль, особливо коли величезна кількість покупок відбувається через онлайн магазини. Однак для продавців, які щойно почали освоювати або вже відносно довгий термін використовують такі площадки все ще виникає складність при реалізації продукту.

Різні маркетплейси працюють з різними форматами вигрузок, що в свою чергу спричиняє незручності, у вигляді додаткової конвертації, якщо дилер одразу працює з декількома площадками. Між дилером і постачальником не завжди підтримується постійний зв’язок, що відображається в не актуальності інформації стосовно товару. Ну і звичайно формат даних наданих постачальником може відрізнятися від формату, який використовується маркетплейсом. У дилерів виникає необхідність в вивченні різних онлайн платформ для реалізації своєї продукції, так як на сьогоднішній день ринок електронної комерції активно розвивається і стає крупніше та різноманітнішим як ніколи раніше.

Метою даної роботи є ознайомлення з концепцією маркетплейсів та принципами їх роботи для подальшого проектування системи управління

форматами даних різних маркетплейсів. Сутність цієї моделі управління полягає в тому, що формати надані дилером чи маркетплейсом зберігаються у системі. Для перенесення своєї продукції в інший маркетплейс буде використовуватися зазначена система управління яка визначає вхідний та вихідний формат, а також буде конвертувати інформацію стосовно товару з вхідного у вихідний формат. Використання цієї моделі також сприяє синхронізації товару в різних маркетплейсах, адже після внесення змін в одному з них, дані будуть змінені в інших, що було неможливим і потребувало ручної зміни на всіх торговельних площадках, через різні формати якими користуються маркетплейси.

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Опис та аналіз предметної області

Маркетплейси – це веб-сайти або платформи, де кілька сторонніх продавців можуть пропонувати продукти або послуги клієнтам [2]. З роками ці ринки стають дедалі популярнішими, оскільки пропонують багато переваг як продавцям, так і покупцям.

Для продавців онлайн-ринки надають готову платформу для продажу їхніх продуктів або послуг із великою та потенційно глобальною клієнтською базою. Вони також виграють від усталеної репутації ринку та довіри серед клієнтів.

Для покупців онлайн-ринки пропонують широкий асортимент продуктів або послуг від багатьох продавців, часто за конкурентними цінами. Клієнти можуть порівнювати товари від різних продавців і читати відгуки інших покупців, перш ніж зробити покупку.

Деякі з найпопулярніших онлайн-ринків включають Amazon, eBay, Alibaba та Etsy, в Україні такими є Rozetka, Prom, E-katalog. Однак існують також ринкові ніші, які обслуговують конкретні галузі чи види продукції [5].

Загалом онлайн-ринки змінили спосіб купівлі та продажу продуктів і послуг, забезпечивши зручність, вибір і конкурентоспроможні ціни як для покупців, так і для продавців, а також створивши нові можливості як для компаній, так і для споживачів.

Існують різні типи онлайн маркетплейсів, зокрема B2B (бізнес-бізнес), B2C (бізнес-споживач), C2C (споживач-споживач) і вертикальні ринки (орієнтовані на конкретні галузі або продукти) [4].

B2B ринки з’єднують підприємства з іншими підприємствами як покупцями та продавцями, B2C ринки з’єднують підприємства з окремими споживачами як покупцями, C2C ринки з’єднують окремих споживачів з іншими окремими споживачами як покупцями та продавцями, а вертикальні

ринки обслуговують певну галузь чи сегмент ринку.

B2B ринки зосереджені на поліпшенні транзакційних операцій між компаніями щодо товарів і послуг, які підтримують їх діяльність, тоді як B2C ринки зосереджені на наданні споживачам широкого спектру продуктів і послуг. Торгові майданчики C2C дозволяють окремим споживачам купувати та продавати товари чи послуги іншим споживачам. Вертикальні ринки зосереджені на обслуговуванні конкретних потреб вузьконаправленної частини ринку.

B2B-транзакції зазвичай передбачають більші обсяги закупівель і вищу вартість транзакцій порівняно з B2C-транзакціями. Операції C2C можуть сильно відрізнятися за обсягом і вартістю. Вертикальні ринки можуть мати менші обсяги транзакцій, але вони можуть зосереджуватися на вищих або спеціалізованих продуктах і послугах.

Продукти та послуги B2B часто складніші та вимагають більше технічних знань, ніж продукти B2C. Продукти та послуги C2C можуть значно відрізнятися за складністю та необхідним досвідом. Вертикальні ринки можуть пропонувати вузькоспеціалізовані продукти та послуги, адаптовані до конкретних потреб.

Цикл продажів B2B зазвичай триваліші та передбачають більше переговорів і налаштувань порівняно з циклами продажів B2C. Цикли продажів C2C часто коротші та простіші. Вертикальні ринки можуть мати довший цикл продажів через спеціалізований характер продуктів і послуг, які вони пропонують [9].

Ціноутворення B2B часто є більш складним і може включати знижки за обсяг, узгоджене ціноутворення та інші спеціалізовані структури ціноутворення, тоді як ціноутворення B2C зазвичай є більш простим. Ціни C2C зазвичай встановлюються окремими продавцями та можуть значно відрізнятися. Вертикальні ринки можуть мати структуру ціноутворення, яка є специфічною для певного сегмента ринку.

B2B маркетинг і реклама часто зосереджуються на побудові відносин із

клієнтами та демонстрації досвіду, тоді як B2C маркетинг і реклама зазвичай зосереджені на підвищенні впізнаваності бренду та створенні імпульсивних покупок. Маркетинг і реклама C2C можуть включати маркетинг у соціальних мережах, рекомендації з вуст в уста та інші маркетингові заходи з низового рівня. Вертикальні ринкові майданчики можуть використовувати спеціалізовані маркетингові та рекламні канали, які є специфічними для відповідної категорії ринку [8].

Ринкові майданчики B2B можуть пропонувати більш спеціалізоване обслуговування клієнтів і технічну підтримку, ніж ринкові майданчики B2C і C2C. Торгові майданчики C2C можуть мати обмежені можливості обслуговування клієнтів і покладатися на оцінки та відгуки користувачів для забезпечення контролю якості. Вертикальні ринкові майданчики можуть пропонувати клієнтам обслуговування, адаптоване до потреб конкретної галузі. Таким чином, B2B, B2C, C2C та вертикальні ринки відрізняються за цільовою аудиторією, метою, обсягом, складністю продукту, циклом продажів, структурою ціноутворення, маркетингом і рекламою, а також обслуговуванням клієнтів. Конкретний тип використовуваного ринку залежатиме від потреб підприємства чи галузі, що обслуговується.

Маркетплейси дозволяють ділерам розширювати свою клієнтську базу та виходити на нові ринки, а також надають покупцям доступ до більш широкого спектру постачальників і продуктів [10]. Вони зазвичай пропонують ряд функцій і послуг, зокрема:

− компанії можуть розміщувати свої продукти чи послуги на ринку, включаючи докладні описи, зображення та інформацію про ціни;

− покупці можуть шукати продукти або послуги на основі різних критеріїв, таких як категорія продукту, місце розташування та ціна;

− ринкові майданчики B2B часто надають інструменти для керування транзакціями, такими як відстеження замовлень, виставлення рахунків і обробка платежів;

− ринкові майданчики B2B можуть пропонувати послуги підтримки

клієнтів, щоб допомогти покупцям і продавцям вирішити будь-які проблеми;

− багато B2B ринків включають систему рейтингів і оглядів, що дозволяє покупцям оцінювати якість продуктів і послуг, які пропонують різні постачальники.

Торгові майданчики можуть запропонувати продавцям низку переваг, зокрема підвищення видимості товару, доступ до ширшого кола клієнтів і спрощене керування транзакціями. Однак підприємства також повинні знати про потенційні ризики та проблеми використання ринків, такі як посилення конкуренції та необхідність ретельного управління цінами та списками продуктів.

Інтернет-ринки надають продавцям платформу для охоплення ширшої аудиторії та збільшення продажів. Вони також пропонують інструменти для керування запасами, ціноутворенням і доставкою. Продавці повинні сплачувати комісію онлайн-ринку за свої послуги, і вони можуть зіткнутися з конкуренцією з боку інших продавців. Вони також повинні дотримуватися політики та вказівок ринку [8].

Покупці можуть насолоджуватися широким вибором продуктів і послуг від багатьох продавців в одному місці. Вони також можуть порівнювати ціни, читати відгуки та користуватися політиками захисту клієнтів. покупці можуть зіткнутися з проблемами, такими як підроблені продукти, оманлива інформація або шахрайство з боку недобросовісних продавців. Однак більшість онлайн-ринків мають політику та заходи для вирішення цих проблем.

1.2 Формати даних

Маркетплейси зазвичай використовують різноманітні формати даних для керування потоком інформації. Нижче наведено кілька поширених

форматів даних, які використовуються на ринках.

XML — це мова розмітки, яка використовується для зберігання та транспортування даних (рисунок 1.1).

```
<product>
    <property 'name' = 'name'>product1</property>
    <property 'description' = 'name'>product description</property>
    <property 'price' = 'name'>30.6</property>
    <property 'owner' = 'name'>owner1</property>
</product>
```

Рисунок 1.1 – Приклад XML файлу

Попередньо визначені правила спрощують передачу даних у вигляді XML-файлів у будь-яку мережу, оскільки отримувач може використовувати ці правила для точного та ефективного зчитування даних. На відміну від інших мов програмування, XML не може виконувати обчислювальні операції сам по собі. Замість цього для управління структурованими даними можна використовувати будь-яку мову програмування або програмне забезпечення. Для визначення даних використовуються символи розмітки, звані тегами в XML. Наприклад, для представлення даних для книжного магазину можна створити такі теги, як <book>, <title> і <author>. Коли компанія продає товар або сервіс іншої компанії, їм необхідно обмінюватися такими відомостями, як вартість, характеристики та графіки поставок. За допомогою розширеної мовної розмітки (XML) вони можуть обмінюватися всією необхідною інформацією в електронному вигляді та автоматично закривати складні операції без втручання людини. Комп'ютерні програми, такі як пошукові системи, можуть сортувати і класифікувати XML-файли більш ефективно і точно, ніж інші типи документів. На основі тегів XML пошукові системи можуть точно класифікувати мітки для релевантних результатів пошуку. Таким чином, XML допомагає комп'ютерам більш ефективно інтерпретувати природну мову. Багато ринків використовують XML для визначення своїх структур даних і полегшення обміну даними між різними системами [1].

CSV — це простий формат файлу, який використовується для зберігання табличних даних, наприклад списків продуктів (рисунок 1.2).

```
name, description, price, owner
product1, product description, 30.6, owner1
```

Рисунок 1.2 – Приклад CSV файлу

Це спосіб обміну структурованою інформацією, наприклад вмістом електронної таблиці, між програмами, які не обов'язково можуть спілкуватися одна з одною напряму. Поки дві програми можуть відкривати файл CSV, вони можуть обмінюватися даними. Наприклад, можна зберегти контактну інформацію з Microsoft Excel як файл CSV та імпортувати її в адресну книгу Microsoft Outlook. Незважаючи на назву, у файлі CSV не потрібно покладатися на коми як роздільник між фрагментами інформації. Цей роздільник може бути крапкою з комою, пробілом або іншим символом, хоча кома зустрічається найчастіше. Файли CSV можна легко імпортувати та експортувати з різних систем, і вони зазвичай використовуються на ринках для обміну даними з продавцями. Цей формат служить для різних бізнес-цілей. Наприклад, компанії використовують його для експортування великого обсягу даних у більш концентровану базу даних [7]. Вони також виконують дві інші основні бізнес-функції:

– файли CSV є звичайними текстовими файлами, що полегшує їх створення для розробників веб-сайтів;

– оскільки вони являють собою звичайний текст, їх легше імпортувати в електронну таблицю чи іншу базу даних, незалежно від конкретного програмного забезпечення, щоб краще організувати великі обсяги даних.

У світі онлайн-комерції однією з головних цілей є охоплення великої кількості клієнтів. Оскільки файли CSV легко впорядкувати, власники компаній електронної комерції можуть маніпулювати цими файлами різними

способами. Файли CSV здебільшого використовуються для імпорту та експорту важливої інформації, як-от даних клієнтів або замовлень, у базу даних і з неї. Більш практичним прикладом цього може бути бізнес електронної комерції, який купує дані клієнтів із веб-сайту соціальних мереж. Інтернет-мережа, швидше за все, надсилатиме інформацію про споживача до вашої бази даних у форматі CSV, що спрощуватиме швидкий і легкий обмін даними. Якщо файли CSV відформатовано правильно, їх легко конвертувати в інші типи файлів. Файли CSV також не є ієрархічними чи об'єктно-орієнтованими, тобто вони мають повсюдну структуру, що є ще одним фактором, який полегшує їх імпорт, експорт і конвертацію.

JSON — це полегшений формат даних, який зазвичай використовується для передачі даних між веб-сервером і клієнтом як альтернатива XML. Файл JSON зберігає дані в парах ключ-значення та масивах; програмне забезпечення, для якого його створено, отримує доступ до даних. JSON дозволяє розробникам зберігати різні типи даних як зрозумілий людині код, де ключі служать іменами, а значення містять пов'язані дані [7]. Синтаксис JSON походить від синтаксису нотації об'єктів JavaScript (рисунок 1.3):

- дані представлені в парах ключ/значення;
- дані відокремлюються комами;
- фігурні дужки містять об'єкти;
- квадратні дужки містять масиви.

```
{
    name: 'product1';
    description: 'product description';
    price: 30.6;
    owner: 'owner1'
}
```

Рисунок 1.3 – Приклад JSON файлу

Виходячи з цього, синтаксис JSON не позбавлений обмежень.

Інформація, надана для ключів і значень, має відповідати певному формату. Наприклад, усі ключі мають бути рядками, написаними в подвійних лапках. Цей тип файлу забезпечує зручний для читання формат для зберігання та обробки даних, коли розробники створюють програмне забезпечення. Спочатку його було розроблено на основі нотації об'єктів Javascript, але згодом він став популярним, тому багато різних мов сумісні з даними JSON. Формат даних JSON — це відкритий стандартний файл (.json) і формат даних, який використовується для обміну даними за допомогою різних форм технологій. Найпоширенішим використанням даних і файлів JSON є читання даних із сервера для відображення на веб-сайті чи веб-додатку та зміна даних із відповідними дозволами. Але це не єдине, для чого він використовується. Комп'ютерні додатки, програми, мобільні програми та багато іншого використовують файли JSON. Він настільки універсальний, що можна стверджувати, що він використовується практично всюди. Багато сучасних торговельних майданчиків використовують JSON для обміну даними з покупцями та продавцями.

EDI — це старіший формат даних, який зазвичай використовувався в минулому для обміну діловими документами, такими як замовлення на купівлю та рахунки-фактури, між різними комп'ютерними системами. Незважаючи на те, що сьогодні вони менш поширені, деякі B2B-ринки все ще використовують EDI для обміну даними з постачальниками [8].

API (інтерфейс прикладного програмування) — не є форматом даних сам по собі, але це звичайний спосіб обміну даними на ринках з іншими системами. Інтерфейси програмного інтерфейсу (API) забезпечують стандартизований спосіб обміну даними між різними системами, що полегшує інтеграцію торгових майданчиків зі сторонніми програмними інструментами та службами.

Підсумовуючи, платформи використовують різні формати даних, включаючи XML, CSV, JSON, EDI та API, щоб керувати потоком інформації між користувачами. Конкретний формат даних, що використовується,

залежатиме від вимог інструменту та систем, з якими його потрібно інтегрувати.

1.3 Моделі даних

Моделі даних стосуються різних підходів і стратегій, які використовуються для управління та інтеграції даних на різних платформах, програмах і системах. У сучасному бізнес-середовищі дані часто зберігаються в різних місцях і в різних форматах, що ускладнює доступ до них і ефективне використання [11].

ETL (Extract, Transform, Load) означає процес міграції та інтеграції даних із кількох джерел у потрібну систему (рисунок 1.4). ETL — це поширений підхід, який використовується в програмах сховищ даних, бізнес-аналітики та аналітики.

Рисунок 1.4 – Схема ETL моделі

Першим кроком у процесі ETL є вилучення даних із вихідних систем. Це може включати запити до баз даних, доступ до файлів або підключення до

API для отримання даних. Витягнуті дані зазвичай зберігаються в проміжній області, яка служить місцем тимчасового зберігання даних [13].

Після вилучення даних наступним кроком є перетворення їх у формат, який може використовувати цільова система. Це може включати очищення та перевірку даних, виконання обчислень або агрегування та відображення даних у загальній моделі даних. Перетворення даних також може передбачати оновлення даних додатковою інформацією, такою як метадані або похідні атрибути.

Останнім кроком у процесі ETL є завантаження перетворених даних у потрібну систему. Це може передбачати вставлення даних у базу даних чи сховище даних або надсилання даних до хмарної служби зберігання. Потім завантажені дані стають доступними для використання подальшими програмами та процесами.

Одна з ключових переваг ETL полягає в тому, що він дозволяє організаціям інтегрувати дані з різних джерел в єдине уніфіковане подання. Це може надати більш повну та точну картину даних і дозволить організаціям приймати більш обґрунтовані рішення на основі інформації, отриманої з цих даних. ETL також може допомогти організаціям покращити якість даних шляхом виявлення та виправлення помилок, невідповідностей і дублікатів у даних [12].

Недоліки ETL:

− процеси ETL можуть бути складними, особливо при роботі з великими обсягами даних і кількома джерелами даних, ця складність може призвести до помилок і невідповідності даних;

− операції можуть бути ресурсомісткими, вимагаючи значного обсягу пам’яті, обчислювальної потужності;

− процеси ETL можуть вводити затримку в процес інтеграції даних, оскільки дані мають бути видобуті, перетворені та завантажені, перш ніж їх можна буде використовувати для аналізу чи інших цілей;

– ETL вимагає постійного обслуговування та керування, що може зайняти багато часу та коштувати;

– зазвичай ETL операції виконуються за розкладом, що означає, що вони можуть не підходити для потреб інтеграції даних у режимі реального часу.

Загалом ETL — це потужний підхід до міграції та інтеграції даних, який може дозволити організаціям використовувати весь потенціал своїх ресурсів. Розуміючи процес ETL і вибираючи правильні інструменти та методи ETL, організації можуть переконатися, що їхні дані точні, повні та своєчасні, і використовувати ці дані для досягнення кращих бізнес-результатів.

Модель об’єднання даних — це підхід до інтеграції даних, який дозволяє організаціям отримувати доступ та інтегрувати дані з багатьох джерел без фізичного переміщення даних (рисунок 1.5). У моделі об’єднання даних дані залишаються у вихідному місці, а методи віртуалізації використовуються для забезпечення уніфікованого перегляду даних [13].

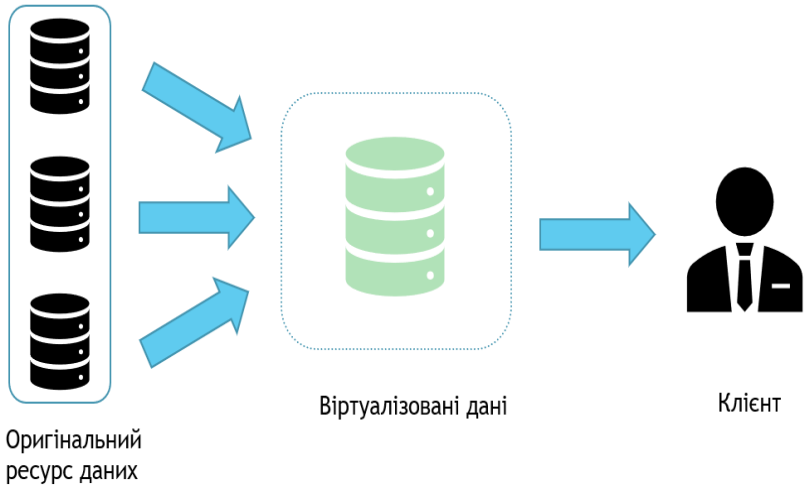

Рисунок 1.5 – Схема моделі об’єднання даних

Модель об’єднання даних часто використовується в ситуаціях, коли фізичне переміщення даних між системами є недоцільним або небажаним.

Наприклад, дані можуть зберігатися в застарілих системах, які важко перенести, або дані можуть підлягати нормативним обмеженням або обмеженням конфіденційності, які забороняють їх переміщення за межі вихідного середовища.

Для впровадження моделі об’єднання даних організації зазвичай використовують платформу віртуалізації даних. Платформи віртуалізації даних забезпечують рівень абстракції, який знаходиться між джерелами даних і програмами, які споживають дані. Цей рівень абстракції дозволяє організаціям отримувати доступ і інтегрувати дані з кількох джерел, ніби вони зберігаються в одному місці.

Платформи віртуалізації даних часто містять інструменти для виявлення та каталогізації джерел даних. Ці інструменти можна використовувати для ідентифікації та інвентаризації джерел даних, які необхідно інтегрувати. Інші інструменти можна використовувати для відображення структури та вмісту джерел даних у загальній моделі даних, що дозволяє програмам легше отримувати доступ до даних та інтегрувати їх. Також можна використовувати інструменти для оптимізації запитів до кількох джерел даних і для забезпечення максимально ефективного виконання запитів [13].

Недоліки моделі об'єднання даних:

− модель об’єднання даних може бути складною, особливо коли йдеться про великі обсяги різнорідних даних і кількох джерел даних;

− дана модель може спричинити проблеми з продуктивністю, оскільки дані мають бути доступні та об’єднані з кількох джерел у режимі реального часу;

− дані можуть бути непослідовними або неповними з кількох джерел, через що винукають проблеми з їх якістю;

− модель об’єднання даних може бути непридатною для офлайн-доступу до даних, оскільки потрібен доступ у режимі реального часу.

Використовуючи платформу віртуалізації даних і впроваджуючи правильні інструменти та методи, організації можуть гарантувати, що їхні дані інтегровані, доступні та придатні для виконання, незалежно від того, де вони зберігаються.

Модель реплікації даних — це модель даних, яка передбачає копіювання даних з одного місця в інше (рисунок 1.6). У моделі реплікації даних зміни до вихідних даних фіксуються та реплікуються в одне або кілька цільових місць, де репліковані дані можна використовувати для різноманітних цілей, наприклад для резервного копіювання та аварійного відновлення, розподіленого доступу до даних [14].

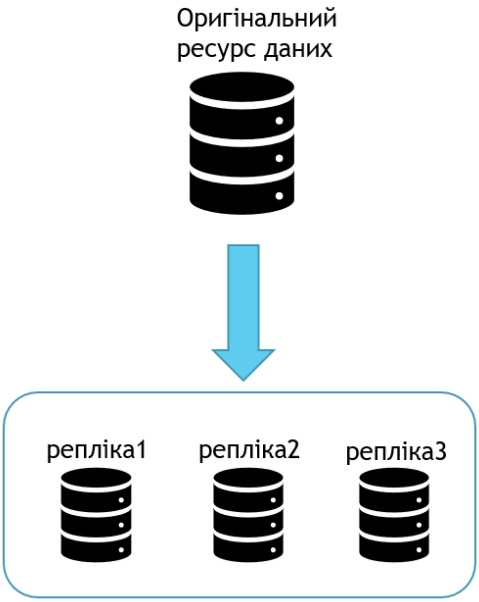

Рисунок 1.6 – Схема моделі реплікації даних

Існує кілька різних типів моделей реплікації даних, кожна з яких має свої переваги та обмеження.

Реплікації миттєвого знімка – копія вихідних даних робиться в певний момент часу та реплікується в одне або кілька цільових місць. Реплікація знімків часто використовується для цілей резервного копіювання та відновлення, а також для надання доступу до даних лише для читання.

Реплікація транзакцій – у реплікації транзакцій зміни у вихідних даних фіксуються та реплікуються майже в реальному часі в одне або кілька цільових місць. Реплікація транзакцій часто використовується для сценаріїв, коли важливо підтримувати синхронізацію даних у кількох системах, наприклад для ввімкнення розподілених програм або аналітики [13].

Реплікація злиттям: у реплікації злиттям зміни у вихідних даних реплікуються двонаправлено між вихідною та цільовою системами. Реплікація злиттям часто використовується в сценаріях, коли дані потрібно синхронізувати між кількома системами, які не завжди підключені до мережі.

Однорангова реплікація: у одноранговій реплікації дані реплікуються між кількома системами децентралізованим способом. Кожна система в одноранговій мережі є як джерелом, так і ціллю для реплікації, що дозволяє розподіляти та тиражувати дані в мережі.

Вибір моделі реплікації залежить від конкретних вимог сценарію використання, таких як обсяг і частота змін даних, вимоги до затримки для реплікації і потреби в узгодженості та цілісності даних.

Реплікацію даних можна реалізувати за допомогою різноманітних технологій та інструментів, включаючи інструменти реплікації бази даних, платформи проміжного програмного забезпечення та служби реплікації даних у хмарі. Вибір технології реплікації залежить від конкретних вимог сценарію використання, таких як обсяг і частота змін даних, вимоги до затримки для реплікації даних і потреби в узгодженості та цілісності даних.

Недоліки моделі реплікації даних:

− модель реплікації даних може запроваджувати дублювання даних, оскільки дані мають бути скопійовані з джерела в локальне сховище;

− можливе виникнення проблем узгодженості даних, оскільки дані можуть бути неузгодженими з кількох джерел, а оновлення не синхронізуватися;

− така модель вимагає синхронізації даних між вихідним джерелом і локальним сховищем, що може бути складним і трудомістким.

Модель реплікації даних є потужним підходом до керування даними, який дозволяє організаціям зберігати копії даних у кількох місцях, а також розподіляти дані для резервного копіювання, аварійного відновлення та сценаріїв розподіленої програми та аналітики. Вибравши правильну технологію реплікації та впровадивши правильну модель реплікації даних, організації можуть гарантувати, що їхні дані доступні та надійні, незалежно від того, де вони потрібні.

Ознайомившись із сферою діяльності платформ по управлінню проєктами та з форматами даних, які використовуються найчастіше, можна створити схему взаємодії системи управління форматами (Multiformat) з різними типами користувачів (реальні люди та інші системи) (рисунок 1.7).

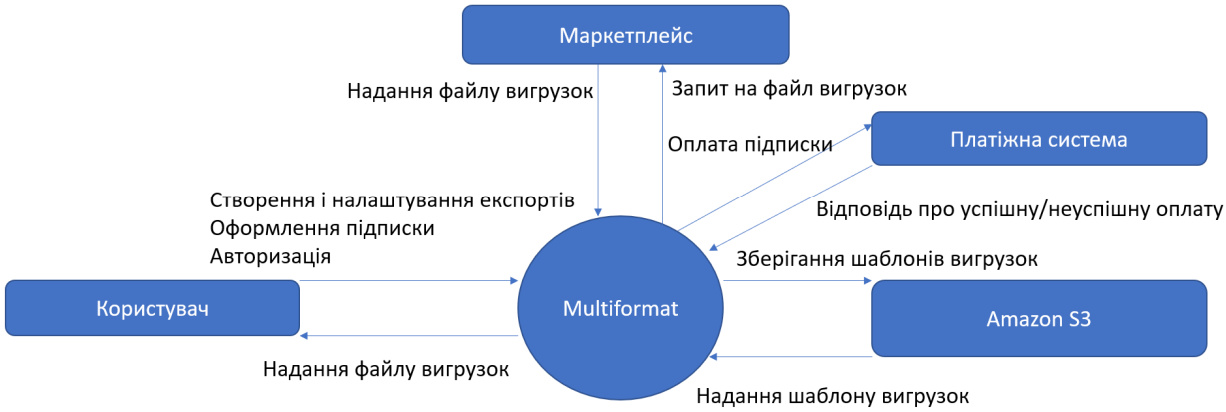

Рисунок 1.7 – Схема інтеграції системи управління форматами даних

Найпростішими операціями буде створення та авторизація користувача, адже це базовий набір функцій будь-якої системи [12]. Окрім створення свого облікового запису користувачі зможуть завантажувати свої формати даних до системи та експортувати їх до різних пптаформ або просто переглянути їх. Експортування та імпортування форматів даних буде реалізоване шляхом використання API, яке надається інструментами для використання їх функціональності. Так як користувачі можуть надавати дані про свої проекти в різних форматах, то цілком доречним є використання одного з хмарних сервісів Amazon – S3, який підходить для цієї задачі.

2 АРХІТЕКТУРА ТА ІТ-ІНФРАСТРУКТУРА МАРКЕТПЛЕЙСІВ

2.1 Архітектура маркетплейсу

Як було визначено, маркетплейс - це платформа e-commerce, онлайн-магазин електронної торгівлі, що надає інформацію про продукт або послугу третіх осіб, чиї операції обробляються його оператором [4]. Якщо пояснювати простіше, то маркетплейс є торговим майданчиком, який працює за принципом сполучної ланки між покупцями та продавцями (рисунок 2.1). Спочатку його легко переплутати з інтернет-магазином, але між ними – кардинальна різниця. У маркетплейсу непримітний дизайн, зате в каталозі сотні тисяч товарів. А кожен товар має десятки цін.

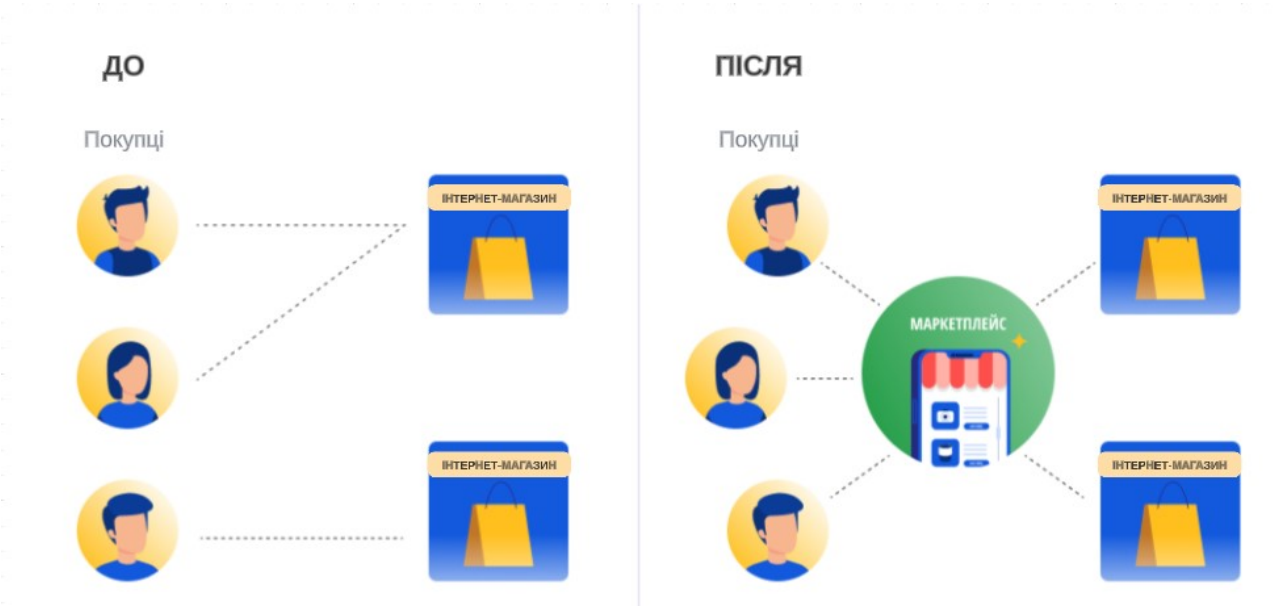

Рисунок 2.1 – Схема взаємодії на маркетплейсі

Принципи маркетплейсу:

- продає чужі товари (партнерів-постачальників);
- з кожного продажу залишає собі відсоток;
- легко підключає нових постачальників.

З цих принципів випливають приємні «побічні ефекти»: шалений

трафік і високі позиції в пошукових системах. Опинитися за запитом товару у видачі вище, ніж сайт виробника чи єдиного постачальника – норма для маркетплейсу.

Справа в тому, що маркетинг, продаж, доставка та інші послуги не менш важливі, ніж виробництво та сам товар. У сучасному світі все це ще дуже дорого. Причина популярності маркетплейсів суто економічна: виробникам та дрібним рітейлерам просто не під силу будувати власні торгові механізми.

У реальному світі дрібні магазинчики витісняються величезними супермаркетами, в онлайні маркетплейси повільно, але вірно знищують інтернет-магазини. Це загальносвітова тенденція [3].

Архітектура маркетплейсу включає наступні складові:

1. Каталог. Складається з трьох частин: дерева каталогу, переліку властивостей та товарів (рисунок 2.2).

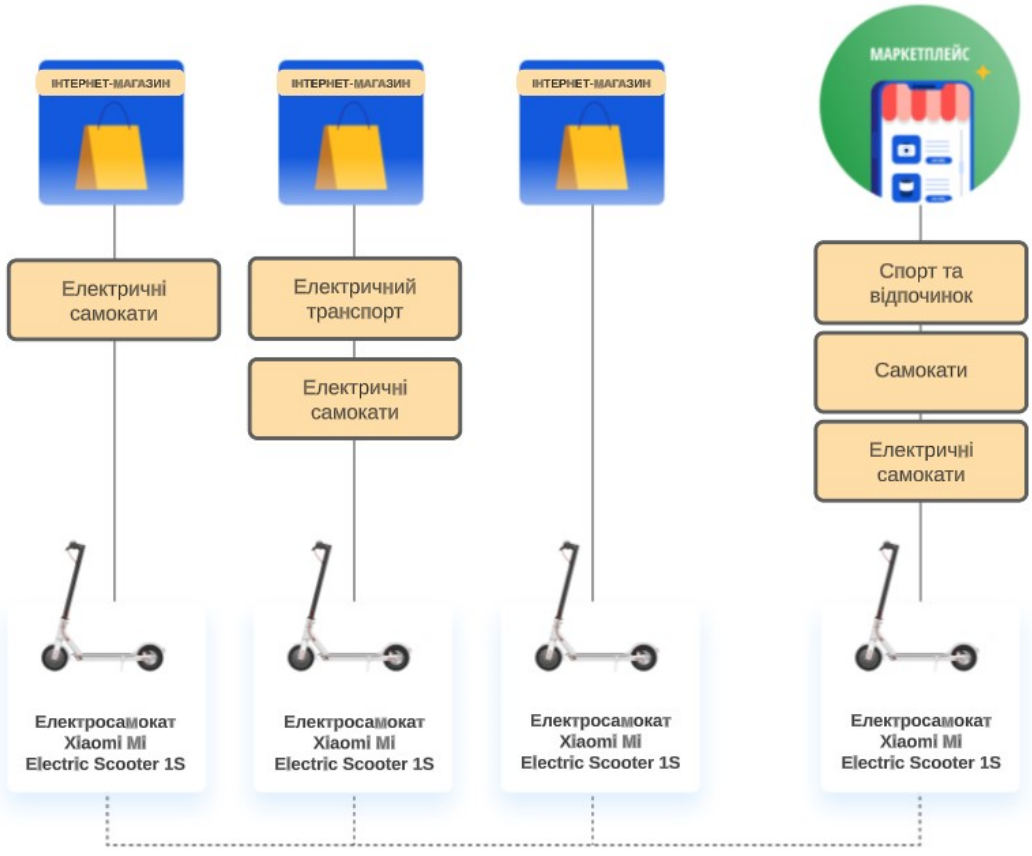

Рисунок 2.2 – Складові каталогу в маркетплейсі

Дерево каталогу складається нове, хоча допустимо надихатися найакуратнішим постачальником. Доробка дерева все одно знадобиться - асортимент маркетплейсу ширший, ніж у постачальників, та й спеціалісти SEO у кожного свої.

Перелік властивостей ґрунтується на переліках постачальників. Заснований, але не дорівнює «логічній сумі». Він у відмінності назв властивостей постачальників і «блукаючими» одиницями виміру. Чи можна об’єднати під «одним дахом» властивості «Вага», «Вага, кг» та «Маса»? Вирішувати має людина, а не ПЗ.

Кожен товар – перетин аналогічних товарів постачальників. Як і з переліком властивостей, виникає проблема різних найменувань. Після зіставлення продуктів і якостей потрібно отримати коректні значення якостей. "0.5 кг", "0,5" і "500 гр" – одне і теж для людини, але не для ПЗ.

Зіставлення властивостей та товарів – робота, яку не можна повністю автоматизувати. Але можна прискорити, розробивши для категорійного менеджера зручний інструмент, а ще краще, вбудувати в панель управління маркетплейсом.

2. Партнери. Маркетплейс не можливий без постачальників. Це відправна точка для коловороту Партнери → Товари → Користувачі → Замовлення → Нові Партнери (рисунок 2.3).

Щоб партнерів було багато, їм потрібні:

− вигідні умови співробітництва. Комісія за замовлення повинна окупатися оборотом, який ми даємо. Маркетплейс має збалансувати комісію замовлень партнера із довгостроковою вигодою від присутності його товарів на сайті;

− зрозуміла звітність. І маркетплейс, і партнер повинні відмінно та однаково розуміти, кому що належить. Стандартний звіт із продажу інтернет-магазину для такого не годиться, потрібно розробляти власний. І, зрозуміло, партнерам потрібен спосіб самостійно сформувати такий звіт у власному кабінеті на сайті;

– просте підключення. У маркетплейсі мають бути два-три типові механізми підключення (наприклад, інтерфейс в особистому кабінеті для ручного завантаження товарів та підтримка фіда YML). І має бути IT-команда (інхаус чи перевірений підрядник), щоб розробити інтеграцію з нуля (якщо одразу зрозуміло, що вигода від підключення партнера покриває витрати на розробку).

Рисунок 2.3 – Коловорот маркетплейсу

3. Монетизація. На етапі розробки маркетплейсу відразу вирішується, хто займатиметься обробкою платежів.

Існує 2 схеми, за якими приймаються платежі. Перша схема показує, що платежі приймає маркетплейс, а потім переводить гроші партнеру, залишаючи собі комісію (рисунок 2.4). Друга схема показує, що платежі приймають партнери, а потім самі виконують оплату комісії маркетплейсу (рисунок 2.5).

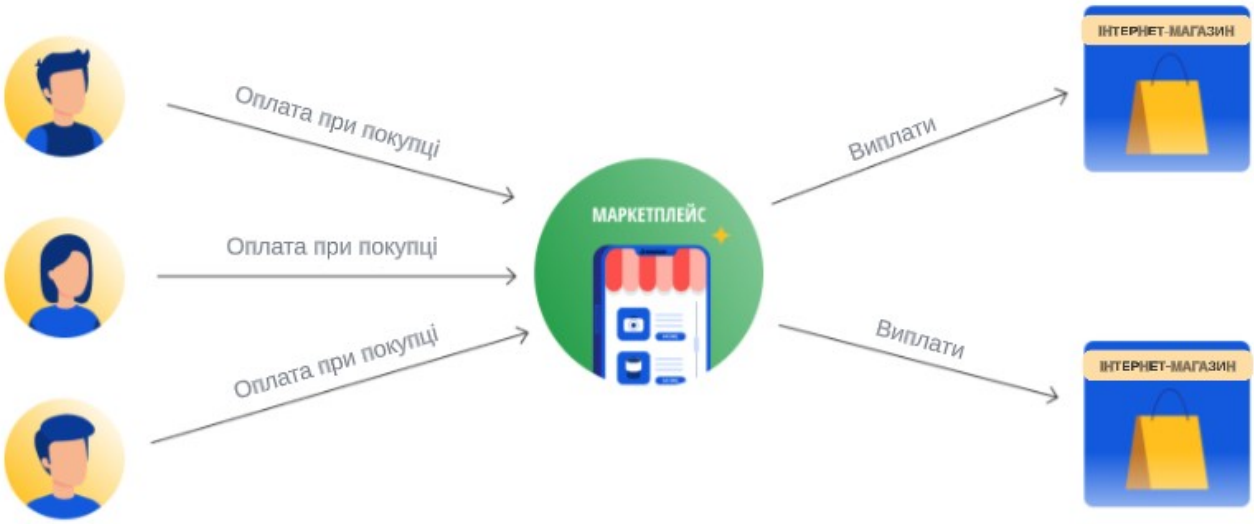

Рисунок 2.4 – Схема монетизації 1

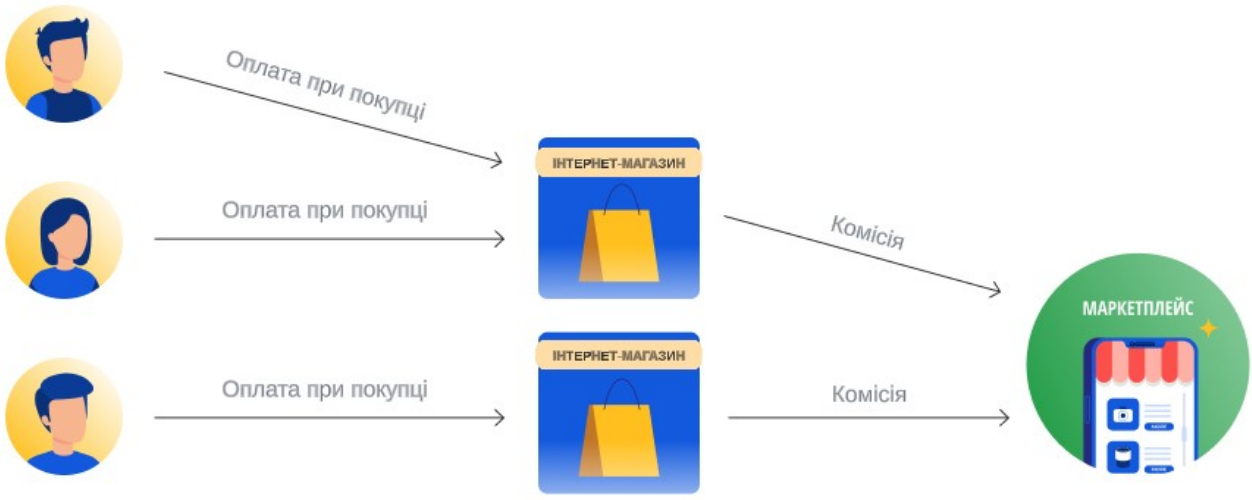

Рисунок 2.5 – Схема монетизації 2

Для 100% передоплати замовлення підходять обидві схеми. Приймати всі платежі найпростіше технічно та зручніше. Від маркетплейсу потрібно лише типова інтеграція з платіжним агрегатором.

Партнерам зручніше друга схема: гроші надходять відразу ж після покупки, а комісію треба платити колись потім. Якщо з пошуком нових партнерів проблеми, варто спробувати другий варіант взаєморозрахунків.

За будь-якої схеми болючим буде процес повернення платежу. Негатив у покупця буде у будь-якому разі, але маркетплейс нічого не може з цим вдіяти у другій схемі. Поверненням займається партнер, у всіх його помилках

покупець звинувачуватиме маркетплейс.

4. Доставка. На етапі розробки маркетплейсу відразу вирішується, хто займатиметься обробкою платежів.

Як і у випадку з оплатою, є дві протилежні схеми доставки:

- доставкою займається маркетплейс (рисунок 2.6);
- доставкою займається партнер (рисунок 2.7).

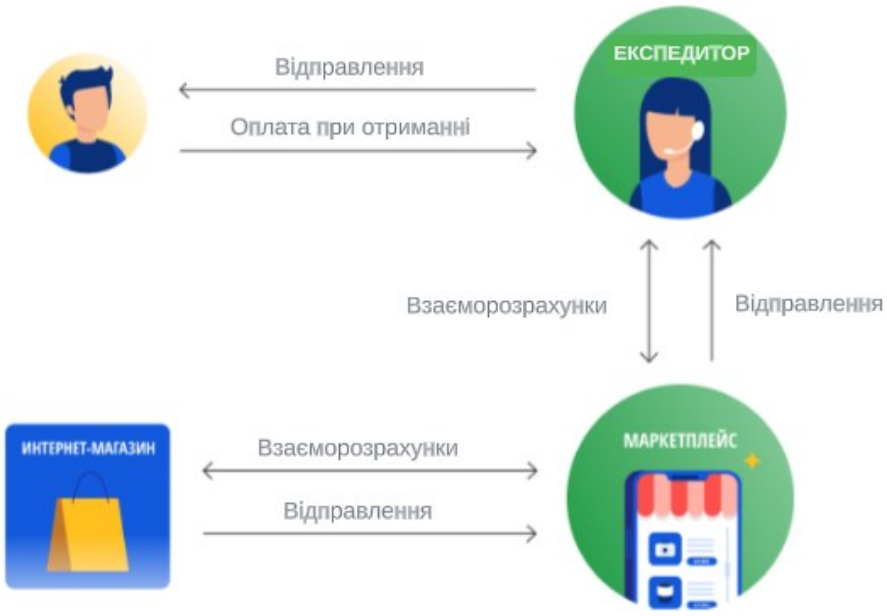

Рисунок 2.6 – Схема доставки 1

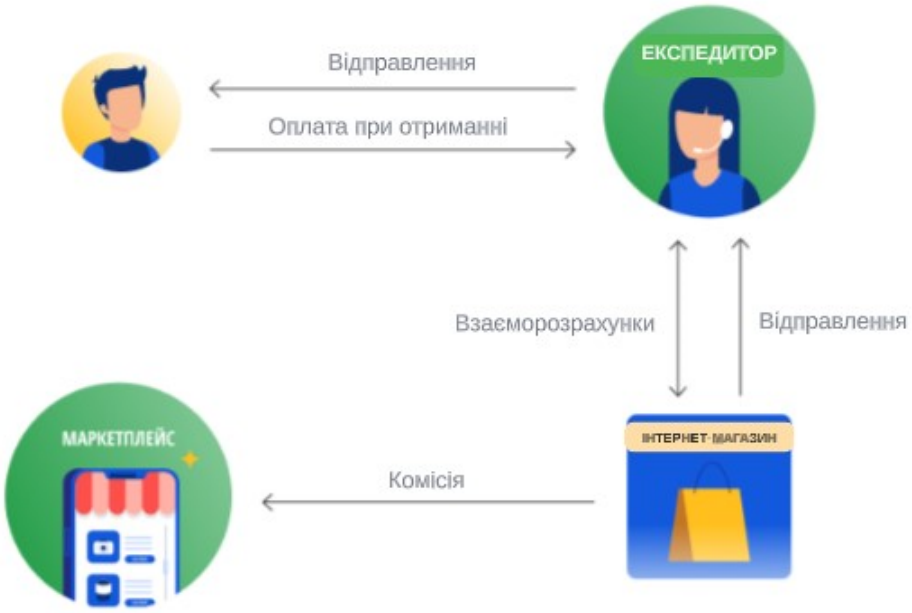

Рисунок 2.7 – Схема доставки 2

Якщо маркетплейс або партнер має власну службу доставки, зі схем просто пропадає посередник у вигляді служби доставки.

В обох схемах діє правило: той, хто взаємодіє зі службою доставки, той з нею і розраховується (інакше йшлося б про тристоронній договір, а це надто ускладнить життя як маркетплейсу, так і партнерам).

Якщо у маркетплейсу є власна служба доставки, яка готова до зростання кількості посилок, то йому більше підходить перша схема, в інших випадках краще залишити взаємодію з доставкою партнерам.

У другому випадку на сайті маркетплейсу має бути реалізована інтеграція з усіма можливими службами доставки всіх партнерів хоча б на рівні калькулятора цін та термінів.

2.2 IT-інфраструктура маркетплейсів

У запропонованій «типовій архітектурі маркетплейсу» (рисунок 2.8) враховано такі особливості:

− облікові системи партнерів не взаємодіють із сайтом маркетплейсу безпосередньо. Точкою входу для всіх є ESB (enterprise service bus – сервісна шина підприємства);

− ESB займається стандартизацією даних від партнерів, в управляючій системі (УС) та на сайт дані потрапляють вже у чистому вигляді;

− сайт взаємодіє з платіжними системами та службами доставки (як у схемі зі звичайним ІМ);

− сайт повинен бути готовий до винесення БД на окремий сервер та кластеризації [13].

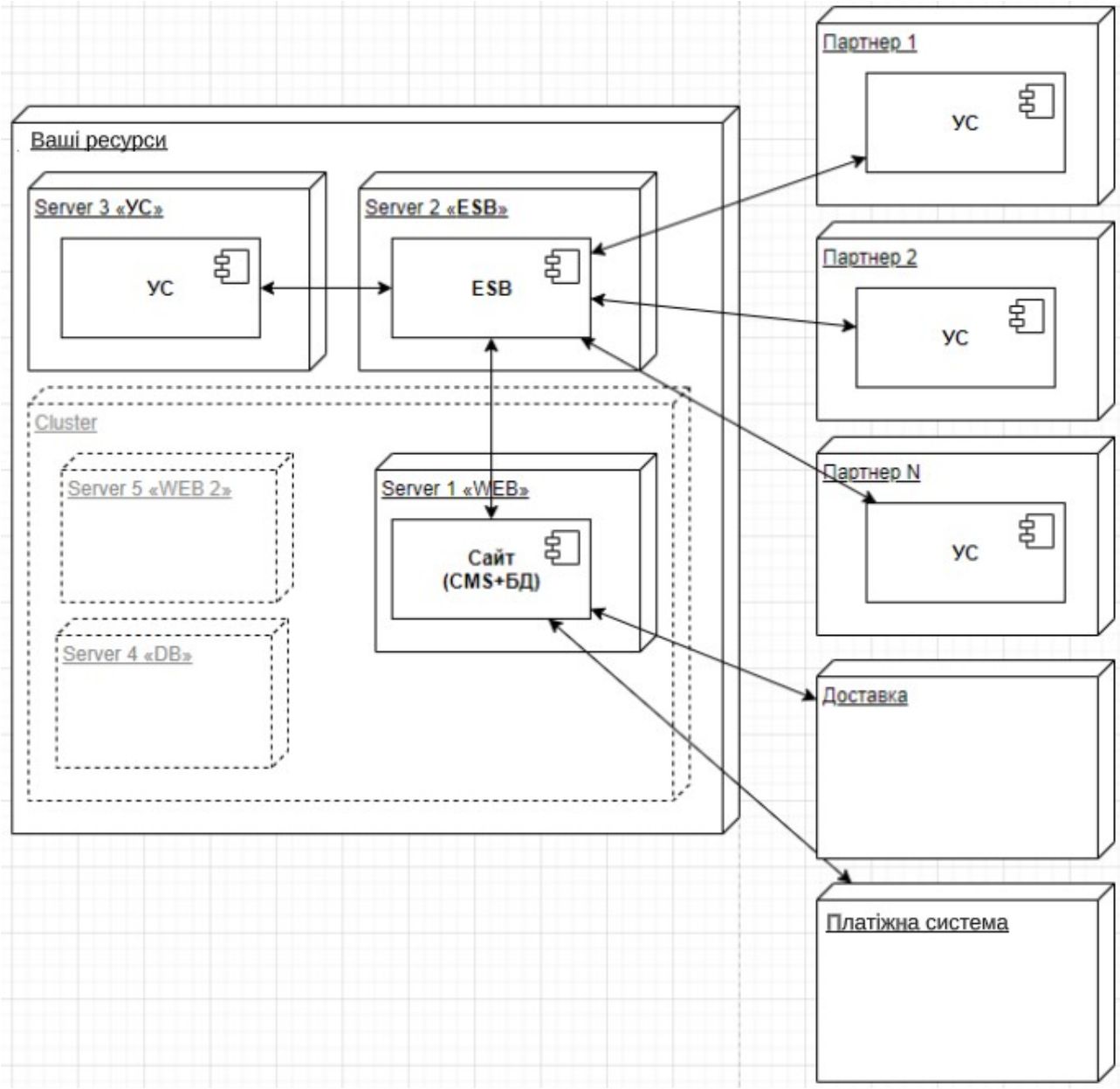

Рисунок 2.8 – IT-інфраструктура маркетплейсу

Типові рішення (наприклад, інтеграція сайту з платформами (рисунок 2.9)) добре справляються з передачею товарів та замовлень до якогось обсягу. Якщо у маркетплейсі більше 300000 товарів, то доведеться чекати кілька днів перш ніж номенклатура повністю хоча б один раз завантажиться на сайт. Весь цей час сервер оброблятиме записи з УС і відповідатиме на запити клієнтів — і в обох випадках швидкодія сайту критична. Прискорювати типовий однопотоковий обмін докупом ресурсів сервера до нескінченності не вдасться. Тому, якщо одразу відомо, що обсяг даних буде

колосальним, потрібно бути готовим рано чи пізно вирішувати навіть таке типове завдання, як обмін товарами/цінами/замовленнями між УС та сайтом [13].

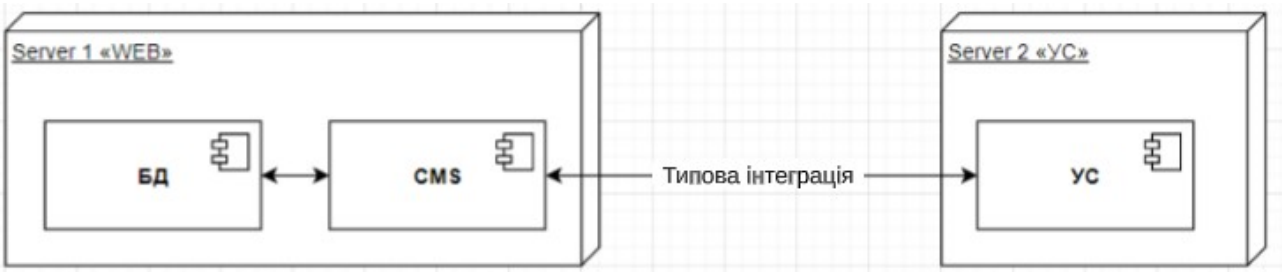

Рисунок 2.9 – Типові рішення маркетплейсу

Можна додати багатопоточність та сервер черг для надійності. Це збільшить навантаження на сервер ще більше, але можна отримати максимальну швидкість обміну даними. Варіанти інтеграції можна комбінувати: наприклад, передавати файли товарів через FTP, а проміжну БД записувати шляхи до файлів [12].

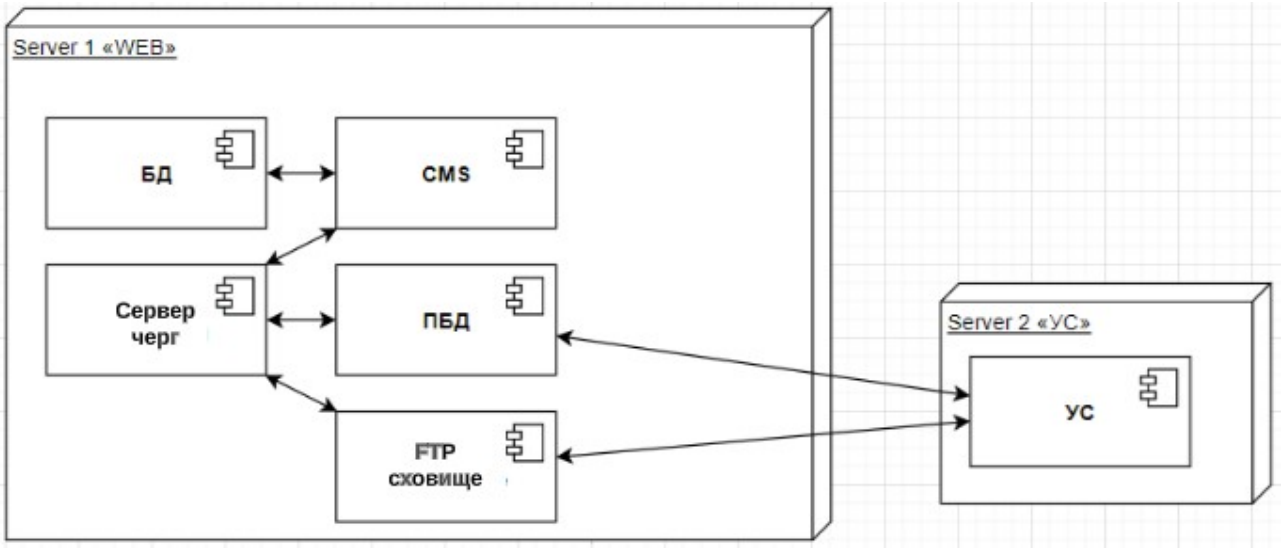

Рисунок 2.10 – Зв’язок між Server1 та Server2

Описана архітектура маркетплейсу та запропонована IT-інфраструктура не є повною та еталонною, але вони дозволяють побачити увесь спектр труднощів по організаціїх та роботі з маркетплейсами.

3 АРХІТЕКТУРНІ СТИЛІ РОЗРОБКИ ПЗ

3.1 Мікросервіси

«Мікросервіси» – це новий термін, який став дуже популярним в області архітектури програмного забезпечення [14]. Цей термін описує стиль розробки програмних систем, який набуває все більшої популярності. Архітектурний стиль мікросервісів означає розробку програмного додатку як набору невеликих сервісів, кожен з яких працює у власному процесі та взаємодіє з легкими механізмами, зазвичай, API ресурсів HTTP. Ці сервіси будуються навколо бізнес-функцій та можуть бути незалежно розгорнуті за допомогою автоматизованого механізму розгортання. Управління цими сервісами здійснюється мінімально централізовано та може використовувати різні технології зберігання даних, а також бути написаним на різних мовах програмування.

Наразі, архітектурному стилю мікросервісів приділяється велика увага, що відображається в статтях, блогах, соціальних мережах та презентаціях на конференціях. Якщо вам складно повірити, просто зверніться до Google Trends. Проте, у програмному співтоваристві є скептики, які не сприймають мікросервіси як щось нове та стверджують, що це просто ребрендинг SOA. Незважаючи на це, архітектура мікросервісів має суттєві переваги, особливо щодо можливості гнучкої розробки та доставки складних корпоративних програм. Ці переваги підтверджуються успіхами великих компаній, таких як Netflix, Amazon та інші, які розповіли про те, як вони успішно масштабують та полегшують постійне надання своїх послуг за допомогою архітектури мікросервісів [14].

Дизайн архітектури мікросервісів не тільки не є балаганом, який слід ігнорувати, але є головною перевагою таких нових стартапів, як Docker, CoreOS, Infrastructure as a Service (Cloud Computing) та інших. Ці продукти створені для спрощення розробки та розгортання додатків, побудованих на

основі мікросервісної архітектури. Docker – це технологія контейнерів з відкритим кодом, яка дозволяє розгортати кілька автономних ізольованих додатків (або служб) на одній ОС Linux. Це можливо завдяки тому, що кожен контейнер працює в своєму власному середовищі, ізольованому від інших..

3.2 Архітектура на базі моноліту

Щоб пояснити мікросервісну архітектуру, можна порівняти її з монолітним стилем, де додаток побудований як єдине ціле. Зазвичай, підприємницькі програми складаються з трьох основних компонентів: клієнтського інтерфейсу, бази даних та серверної програми. Серверна програма обробляє HTTP-запити, виконує бізнес-логіку, звертається до бази даних та формує відповідь відповідно до запиту. У монолітній архітектурі, серверний додаток єдиний та неможливо відрізнити його компоненти. Будь-які зміни в системі вимагають створення та розгортання нової версії серверної програми [14].

Такий монолітний сервер (рисунок 3.1) є природним способом підійти до побудови такої системи. Вся ваша логіка для обробки запиту працює в одному процесі, що дозволяє використовувати основні функції вашої мови для розподілу програми на класи, функції та простори імен. З певною обережністю ви можете запустити та протестувати додаток на ноутбуці розробника та скористатися конвеєром розгортання, щоб забезпечити належне тестування та впровадження змін у виробництво. Ви можете горизонтально масштабувати моноліт, запускаючи багато екземплярів за балансиром навантаження.

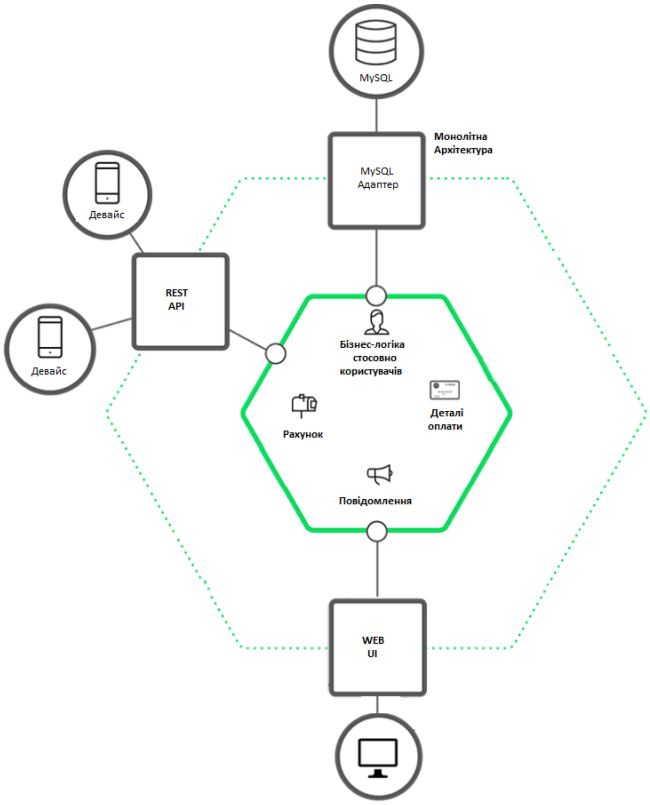

Рисунок 3.1 – Схема додатку з монолітною архітектурою

Хоча монолітні програми можуть бути успішними, все більше людей відчувають розчарування в їхньому використанні, особливо з розвитком хмарних технологій. Цикли змін пов'язані між собою, тому що навіть невелику зміну необхідно внести в усі компоненти програми, перед тим як перебудувати та розгорнути весь моноліт. З часом стає важко зберегти чітку модульну структуру, що ускладнює збереження змін, що повинні стосуватися лише одного модуля, в цьому модулі. Крім того, масштабування вимагає масштабування всієї програми, а не тільки її частин, що потребують більшої кількості ресурсів.

На жаль, цей простий підхід має величезні обмеження. Ці недоліки в результаті стали сильними сторонами мікросервісної архітектури.

Монолітні програми стають все більшими в розмірі, що ускладнює часту та просту випуск нових версій. За рахунок щільного зв'язку між компонентами, планування випуску забирає багато часу у людей з різних груп. Часті випуски також не рекомендуються, оскільки потрібно

переконатися, що програма не зламається через нещодавно випущену функцію.

Монолітні додатки також стикаються з проблемою надійності. Помилки в одному модулі можуть вплинути на роботу всього додатку, оскільки всі модулі працюють в одному процесі. Наприклад, витік пам'яті може викликати збій у всіх модулях і знизити надійність додатку. Крім того, оскільки всі екземпляри додатку однакові, одна помилка може спричинити проблеми з доступністю усього додатку.

Постійна робота програми може стати проблемою, особливо для більших монолітних додатків. Час розгортання може бути значно підвищений, особливо якщо для навіть найменших змін потрібно перебудувати всю програму. Це може ускладнити часте розгортання та зіткнення програми з проблемами безперервної роботи. Наприклад, це може бути особливо важливо для мобільних додатків, де користувачі очікують на постійні нові функції та покращення продукту.

Керування командою та проектом є важкою задачею при розробці монолітних додатків. Навіть якщо додаток розбито на модулі, вони все ще мають взаємозалежності щодо розгортання та випуску. Це призводить до значних затрат часу та ресурсів на планування випуску та керування взаємозалежними модульними розробками.

Монолітні додатки можуть мало ефективно масштабуватись, оскільки різні модулі можуть мати різні вимоги до ресурсів, що може виявитись проблематичним. Наприклад, один модуль може потребувати великої кількості процесорного часу, тоді як інший може вимагати значної кількості пам'яті. Якщо ці модулі розгортаються разом, то їх вимоги до ресурсів повинні бути згодовані. Це може призвести до вибору менш оптимального обладнання для обох модулів, що може знизити продуктивність додатку та збільшити вартість його розгортання.

В монолітних додатках технологічний стек зазвичай є єдиним і обмежує можливість використання спеціалізованих технологій для

конкретних вимог або задач. Наприклад, якщо потрібно обробити великі об'єми даних в реальному часі, може знадобитися використання спеціалізованих технологій обробки потоків даних. Однак, якщо така технологія не підтримується в обраному технологічному стеці монолітного додатку, то використання її стає неможливим. Це може знизити ефективність роботи додатку та зробити його менш конкурентоспроможним на ринку [14].

Заміна компонентів у монолітного додатку може бути вкрай складною задачею, особливо якщо змінюються ключові компоненти або технологічний стек в цілому. Це пов'язано з тим, що монолітні додатки мають глибоку взаємодію між компонентами та частинами коду, тому зміни в одному місці можуть мати вплив на багато інших частин коду. Також, моноліти часто не мають достатньої документації, що робить заміну компонентів ще складнішою.Звісно, можливо робити заміни компонентів поступово, шар за шаром, ретельно перевіряючи кожну зміну на правильність роботи системи в цілому або можна використовувати інструменти автоматизованого тестування для виявлення проблем відразу після змін компонентів, але всі ці варіанти потребують додаткових ресурсів.

Серйозною проблемою для монолітних додатків є складність написання коду. Зазвичай, коли розробники додають нові функції або виправляють помилки, вони можуть не бути повністю усвідомлені, як це впливає на решту додатку. Це може призвести до появи непередбачуваних багів та помилок, а також зростання технічного боргу. Крім того, зміни в одній частині коду можуть вплинути на інші частини додатку, що може призвести до неочікуваного поведінки. Наприклад, внесення змін до бази даних може вплинути на роботу іншої функції, що використовує цю базу даних.

3.3 Архітектура на базі мікросервісів

Компанії, такі як Netflix, eBay та Amazon вирішили зазначені у попередньому розділі проблеми, прийнявши шаблон архітектури мікросервісів [11]. Замість створення єдиного, складного моноліту, ідея полягає в тому, щоб розділити додаток на менші, взаємопов'язані сервіси. Крім того, що сервіси можуть розгортатися та масштабуватися незалежно один від одного, кожен сервіс також забезпечує чіткі межі модулів, навіть дозволяючи написання різних сервісів різними мовами програмування. Керування різними сервісами може здійснюватися різними командами.

Нормальною практикою в мікросервісній архітектурі є розробка міні-додатків, кожен з яких відповідає за свої функції, наприклад, управління замовленнями чи клієнтами (рисунок 3.2). Кожен мікросервіс має свою власну архітектуру, що складається з бізнес-логіки та адаптерів. Інколи мікросервіси надають API, який використовують інші мікросервіси або клієнти. Деякі мікросервіси можуть також реалізувати веб-інтерфейс. Під час виконання кожен екземпляр часто працює як хмарна віртуальна машина або контейнер Docker [14].

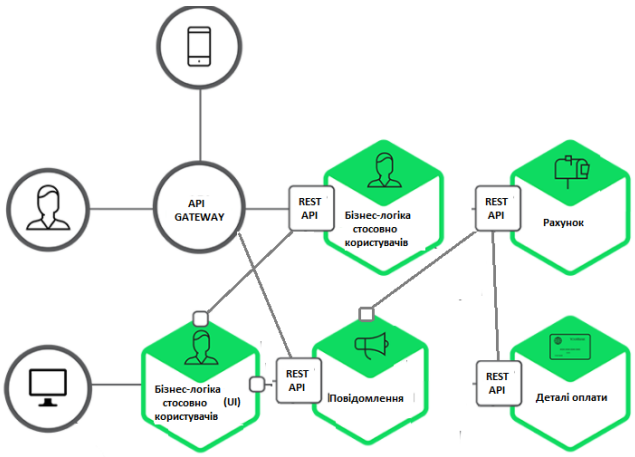

Рисунок 3.2 – Схема додатку з мікросервісною архітектурою

Кожен функціональний аспект програми тепер представлений в окремому мікросервісі, а веб-програма розбита на набір простих веб-додатків. Це полегшує впровадження окремих функцій для конкретних користувачів, пристроїв або спеціалізованих сценаріїв використання.

Кожен окремий мікросервіс у системі має свій власний REST API, а деякі сервіси використовують API, що надаються іншими сервісами, щоб здійснювати свої функції. Інтерфейсні сервіси використовуються іншими сервісами, щоб забезпечити відображення веб-сторінок для користувачів. Додатково, мікросервіси можуть використовувати асинхронний зв'язок, який базується на обміні повідомленнями, щоб забезпечити ефективну комунікацію між різними сервісами.

Деякі мобільні додатки взаємодіють з API REST, але прямий доступ до мікросервісів недоступний. Замість цього, спілкування здійснюється через посередника – API-шлюз. Шлюз API відповідає за керування завантаженням, кешування, контроль доступу, моніторинг та вимірювання API. Для ефективного виконання цих завдань часто використовують NGINX [14].

При виконанні функцій мікросервіс складається з кількох екземплярів, кожен з яких знаходиться в Docker-контейнері. Щоб забезпечити високу доступність, контейнери розгортаються на декількох хмарних віртуальних машинах. На передньому плані служб знаходиться балансувальник навантаження, який розподіляє запити між екземплярами, а також може вирішувати інші проблеми, такі як кешування, контроль доступу, вимірювання API та моніторинг.

Мікросервісна архітектура сильно впливає на взаємодію додатку з базою даних. Замість спільного використання однієї бази даних з іншими сервісами, кожен сервіс має свою власну схему бази даних. Цей підхід, з одного боку, суперечить загальнокорпоративній моделі даних та може призводити до дублювання певних даних. Однак, наявність окремої бази даних для кожної служби є дуже важливою, якщо ви прагнете досягти переваг мікросервісів, оскільки це забезпечує гнучке з'єднання. Крім того,

такий підхід дозволяє кожному сервісу використовувати базу даних, оптимізовану під його потреби, що полегшує масштабування та підтримку окремих сервісів.

3.4 Основні характеристики мікросервісних додатків

Можна виділити основні характеристики додатків на основі мікросервісної архітектури.

Говорячи про компоненти, обумовлюють, що компонент – це одиниця програмного забезпечення, яку можна самостійно замінити та оновити.

У мікросервісній архітектурі бібліотеки використовуються лише як один з можливих засобів компонування програмного забезпечення. Головним способом є розбиття на послуги. Бібліотеки, зазвичай, представляють собою компоненти, які зв'язані в програму і викликаються за допомогою функцій в пам'яті. У порівнянні з цим, мікросервіси – це компоненти, які взаємодіють з механізмами, такими як запити веб-служби або виклики віддаленої процедури. Використання мікросервісів забезпечує більш гнучкий та масштабований підхід до розробки програмного забезпечення, оскільки кожна послуга може бути розгорнута та масштабована окремо, незалежно від інших послуг [14].

Одним з ключових переваг використання мікросервісів є їх незалежність від інших компонентів. Замість того, щоб бути залежними від бібліотек, які використовуються в одному процесі, мікросервіси можуть бути розгорнуті окремо. Якщо бути точним, у разі внесення змін до окремого компонента в монолітній архітектурі ці зміни можуть вимагати передислокації всього додатка. Однак, якщо додаток складається з кількох окремих мікросервісів, зміна в одному сервісі призведе лише до його передислокації. Звичайно, деякі зміни можуть вимагати координації між

мікросервісами, зокрема зміни в інтерфейсах обслуговування. Однак, метою гарної архітектури мікросервісів є зменшення необхідності в такій координації шляхом встановлення чітких кордонів між послугами.

Використання мікросервісів як компонентів також призводить до більш чіткого визначення інтерфейсу компонентів. Це особливо важливо, оскільки більшість мов програмування не мають належних механізмів для визначення явного опублікованого інтерфейсу. Часто розробники мають лише документацію та власну дисципліну, які перешкоджають клієнтам порушувати інкапсуляцію компонента, що може призвести до надмірно щільного зв’язку між компонентами та збільшення складності програми. Однак сервіси дозволяють уникнути цього, використовуючи явні механізми віддаленого виклику, що надають чіткі кордони між компонентами та забезпечують більшу ізольованість між ними. Таким чином, розробники можуть бути впевнені, що компоненти не будуть порушувати інкапсуляцію та зберігати міцні зв'язки між собою.

У використанні мікросервісної архітектури є деякі недоліки, які необхідно враховувати. Один з них – це вартість віддаленого виклику, яка є дорожчою, ніж локальний виклик в межах одного процесу [14]. Це означає, що віддалені API повинні бути більш простими та менш детальними, що може бути незручним у використанні. Крім того, якщо виникає потреба змінити відносини між компонентами, переміщення функціоналу через границі процесів може бути складним і важким процесом.

Мікросервісний підхід до розподілу програмного забезпечення базується на створенні невеликих служб, організованих навколо конкретних можливостей бізнесу. Ці сервіси можуть включати повний набір компонентів, від користувальницького інтерфейсу до постійного сховища та зовнішньої інтеграції. Як результат, розробницькі команди, які працюють з мікросервісами, є багатофункціональними та мають у своєму складі спеціалістів з різних областей, таких як дизайн користувацького інтерфейсу, бази даних та управління проектами. Процес розробки додатків, побудованих

на основі мікросервісів, може забезпечити швидке внесення змін у продукт, а також спрощення розробки і підтримки окремих компонентів.

Спільнота мікросервісів у питанні комунікації та маршрутизації віддає перевагу альтернативному підходу: розумним точкам входу. Додатки, побудовані на основі мікросервісів, прагнуть бути максимально розв’язаними та якомога більш згуртованими – вони володіють власною логікою домену та діють більше як фільтри в класичному розумінні – отримуючи запит, застосовуючи логіку за необхідністю та виробляючи відповідь. Вони керуються та маршритизуються з використанням простих протоколів REST.

Найчастіше використовуються два протоколи – HTTP-запит-відповідь із API ресурсів та обміни повідомленнями. Найкраще вираження першого – це те, що мікросервіси використовують принципи та протоколи, на яких побудована всесвітня павутина. Ресурси, що часто використовуються, можуть бути кешовані дуже незначними зусиллями з боку розробників або операторів [14].

Другий загальновживаний підхід – це обмін повідомленнями через шину повідомлень. Вибрана інфраструктура, як правило, німа (німа, тобто у ролі лише маршрутизатора повідомлень) – прості реалізації, такі як RabbitMQ або ZeroMQ, не роблять набагато більше, ніж забезпечують надійну асинхронний механізм передачі повідомлень – розумні, наприклад Apache Kafka надають більше можливостей, таких як збереження повідомлень, додаткова обробка даних і т.д.

У моноліті компоненти виконуються в процесі, і зв'язок між ними здійснюється через виклик методу або виклик функції. Найбільша проблема в перетворенні моноліту на мікросервіси полягає у зміні схеми спілкування.

Одним із наслідків централізованого управління є тенденція до стандартизації на єдиних технологічних платформах. Децентралізація управління в мікросервісах надає вибір при побудові кожного з них. Можна використовувати Node.js для створення простої сторінки звітів, C++ для особливо складного компонента майже в реальному часі. При необхідності

поміняти інший варіант бази даних, який більше відповідає поведінці роботи одного компонента, не потребує великих зусиль. Такий підхід надає багато переваг. Наприклад, замість того, щоб використовувати набір визначених стандартів, записаних десь на папері, завжди є можливість створення корисних інструментів, якими інші розробники можуть користуватися для вирішення подібних проблем до тих, з якими вони зустрічаються. Ці інструменти, як правило, збирають із реалізацій та інколи діляться з ними більш широкою групою, але не виключно з використанням внутрішньої моделі з відкритим кодом. Зараз, коли git і github стали фактичною системою контролю версій, яку вибирають, практики з відкритим кодом стають все більш і більш поширеними у власному масштабі. Netflix є гарним прикладом організації, яка дотримується цієї філософії. Обмін корисним і, перш за все, випробуваним боєм кодом, оскільки бібліотеки заохочують інших розробників вирішувати подібні проблеми подібними способами, але залишає відкритими двері для вибору іншого підходу, якщо потрібно. Спільні бібліотеки, як правило, зосереджені на загальних проблемах зберігання даних, міжпроцесорного зв'язку та автоматизації інфраструктури [15].

Децентралізація управління даними представлена різними способами. На найбільш абстрактному рівні це означає, що концептуальна модель світу буде відрізнятися між системами. Це поширена проблема при інтеграції на великому підприємстві, погляд клієнта на продаж відрізнятиметься від подання підтримки. Деякі речі, які у вікні продажів називаються клієнтами, можуть взагалі не відображатися у вікні підтримки. Ті, що мають, можуть мати різні атрибути та (гірше) загальні атрибути з незначно різною семантикою.

DDD ділить складний домен на кілька обмежених контекстів і відображає взаємозв'язки між ними. Цей процес корисний як для монолітної архітектури, так і для мікросервісних архітектур, але існує природний взаємозв'язок між межами обслуговування та контексту [15].

Окрім децентралізації рішень щодо концептуальних моделей, мікросервіси також децентралізують рішення щодо зберігання даних. Хоча монолітні програми віддають перевагу єдиній логічній базі даних для постійних даних, підприємства часто віддають перевагу єдиній базі даних для цілого ряду програм – багато з цих рішень спираються на комерційні моделі постачальників щодо ліцензування. Мікросервіси воліють дозволяти кожній службі керувати власною базою даних, або різними екземплярами однієї і тієї ж технології баз даних, або зовсім різними системами баз даних.

Хоча архітектура мікросервісів має свої переваги, як і будь-яка інша технологія, вона також має свої недоліки. Одним з них є сама назва "мікросервіс", яка занадто акцентує увагу на розмірі послуги, тоді як важливо розуміти, що ці послуги є засобом досягнення мети, а не метою самою по собі. Мета мікросервісів полягає у розкладанні програми для полегшення розробки та розгортання додатків.

Іншим важливим недоліком мікросервісів є складність, що виникає через розподілену систему додатка. Розробники повинні вибирати та впроваджувати механізм міжпроцесорної комунікації на основі обміну повідомленнями та писати код для обробки часткових помилок. Це набагато складніше, ніж у монолітній програмі, де модулі викликають один одного за допомогою викликів методів/процедур на рівні мови [14].

Ще одним значним недоліком мікросервісів є архітектура розділених баз даних. У монолітній програмі транзакції можна легко виконувати, оскільки існує єдина база даних. У додатку на основі мікросервісів потрібно оновити кілька баз даних, що належать різним службам. Використання розподілених транзакцій не є можливим, тому розробники повинні використовувати підхід, який базується на послідовності, що є більш складним завданням для них. Нарешті, тестування програми мікросервісів є набагато складнішим завданням, ніж у монолітній програмі [15].

4 CLOUD-ТЕХНОЛОГІЇ

Cloud-технології зробили революцію стосовно методів зберігання, обробки та доступу до даних. Хмарні обчислення — це модель доставки обчислювальних ресурсів на вимогу через Інтернет. За допомогою хмарних обчислень ви можете використовувати мережу віддалених серверів, розміщених в Інтернеті, для зберігання, керування та обробки даних замість використання локальних серверів або персональних комп'ютерів [17].

Модель хмарних обчислень має три основні категорії (рисунок 4.1): інфраструктура як послуга (IaaS), платформа як послуга (PaaS) і програмне забезпечення як послуга (SaaS). Кожна категорія надає користувачеві різні рівні абстракції.

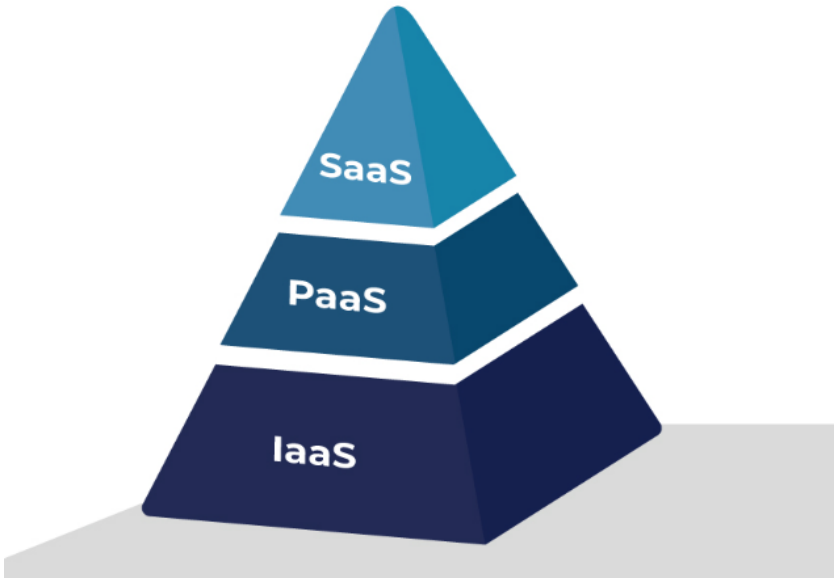

Рисунок 4.1 – Схема основних послуг хмарних технологій

Інфраструктура як послуга (IaaS) — це модель служби хмарних обчислень, яка дозволяє користувачам отримувати доступ і керувати обчислювальними ресурсами, такими як віртуальні машини, сховище та мережа, через Інтернет. Постачальники IaaS пропонують користувачам можливість надавати та керувати цими ресурсами на вимогу без необхідності

інвестувати в дороге обладнання чи інфраструктуру центру обробки даних [16].

За своєю суттю IaaS — це надання доступу до масштабованих обчислювальних ресурсів. Забезпечуючи віртуалізоване середовище, провайдери IaaS можуть надати користувачам доступ до практично необмежених ресурсів, які за потреби можна збільшувати або зменшувати (рисунок 4.2). Ця масштабованість є однією з ключових переваг IaaS, оскільки дозволяє підприємствам швидко та легко реагувати на мінливі вимоги, не турбуючись про обмеження фізичної інфраструктури [18].

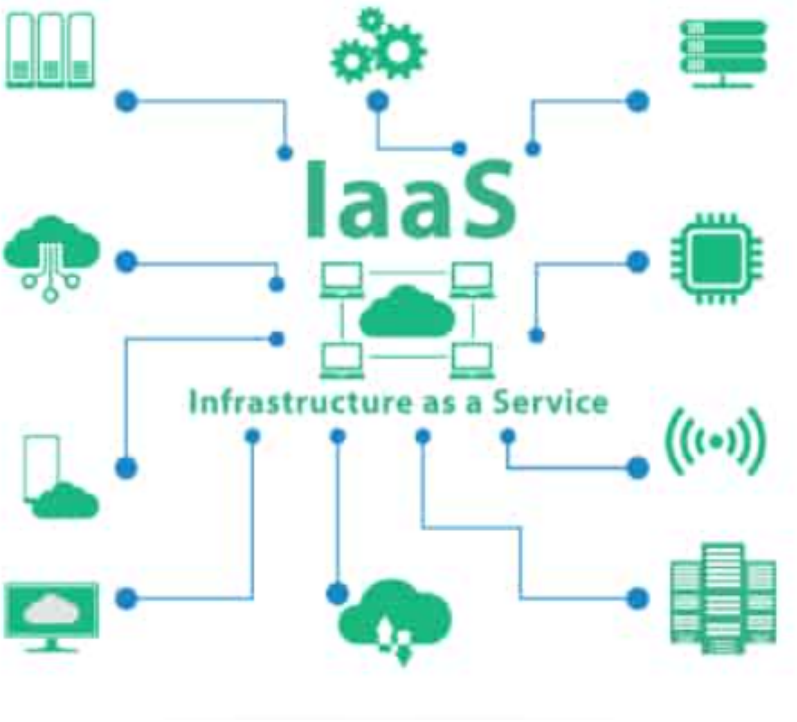

Рисунок 4.2 – IaaS

Однією з головних переваг IaaS є гнучкість, яку вона забезпечує. Користувачі можуть вибирати з безлічі обчислювальних ресурсів, включаючи віртуальні машини, сховище та мережу, і надавати їх за потреби. Це дозволяє компаніям створювати індивідуальні інфраструктурні рішення, які відповідають їхнім конкретним потребам, не турбуючись про витрати та складність обслуговування фізичного обладнання.

Окрім гнучкості, IaaS також пропонує користувачам високий рівень контролю над їх інфраструктурою. Користувачі можуть отримувати доступ до своїх ресурсів і керувати ними через веб-інтерфейс, а також можуть легко налаштовувати та керувати своїми віртуальними машинами, сховищем і мережею. Цей рівень контролю дозволяє компаніям оптимізувати свою інфраструктуру для продуктивності та ефективності, а також забезпечувати безпеку їхніх даних і відповідність галузевим нормам.

Ще однією ключовою перевагою IaaS є його економічна ефективність. Усунувши потребу у фізичній інфраструктурі, підприємства можуть заощадити на обладнанні, обслуговуванні та витратах на електроенергію. Крім того, постачальники IaaS зазвичай пропонують гнучкі моделі ціноутворення, такі як оплата за використання або ціноутворення на основі підписки, що дозволяє компаніям платити лише за ресурси, які вони використовують, не турбуючись про довгострокові зобов’язання чи дорогі початкові витрати [17].

Коли мова заходить про постачальників IaaS, на ринку є багато варіантів. Деякі з найпопулярніших постачальників включають Amazon Web Services (AWS), Microsoft Azure, Google Cloud Platform і IBM Cloud. Кожен постачальник пропонує низку послуг і функцій, і компаніям потрібно буде оцінити свої потреби та вимоги, щоб визначити, який постачальник їм найкраще підходить.

Одним із ключових факторів при виборі постачальника IaaS є рівень безпеки, який він пропонує. Оскільки IaaS передбачає зберігання та керування конфіденційними даними в хмарі, безпека є головним пріоритетом. Постачальники повинні пропонувати ряд функцій безпеки, таких як шифрування, брандмауери та засоби контролю доступу, щоб допомогти компаніям захистити свої дані та відповідати галузевим нормам.

Ще одним важливим моментом при виборі постачальника IaaS є рівень підтримки та послуг, які вони пропонують. Постачальники повинні пропонувати цілодобову підтримку та ряд професійних послуг, таких як

консультації, міграція та навчання, щоб допомогти підприємствам отримати максимальну віддачу від своєї інфраструктури.

Загалом IaaS — це потужна модель хмарних обчислень, яка надає підприємствам доступ до масштабованих, гнучких і економічно ефективних обчислювальних ресурсів. Використовуючи IaaS, компанії можуть створювати індивідуальні інфраструктурні рішення, які відповідають їхнім конкретним потребам, а також користуються перевагами високого рівня безпеки, контролю та підтримки. Оскільки популярність хмарних обчислень продовжує зростати, очевидно, що IaaS і надалі залишатиметься ключовим рушієм інновацій і зростання в найближчі роки.Платформа як послуга (PaaS) пропонує вищий рівень абстракції, ніж IaaS. PaaS надає користувачам платформу для розробки, запуску та керування програмами, не турбуючись про базову інфраструктуру. PaaS пропонує інструменти, проміжне програмне забезпечення та інші послуги, які допомагають розробникам створювати та розгортати свої програми швидко й ефективно [16-18].

Програмне забезпечення як послуга (SaaS) — це модель хмарних обчислень, яка доставляє програмні додатки через Інтернет, дозволяючи користувачам отримувати до них доступ і використовувати їх через веб-браузер або мобільний додаток (рисунок 4.3). У цій моделі програмне забезпечення розміщується та керується стороннім постачальником, який відповідає за його доступність, безпеку та обслуговування.

SaaS став дедалі популярнішим за останнє десятиліття, оскільки він надає підприємствам економічно ефективний і гнучкий спосіб доступу до програмних програм без потреби в локальній інфраструктурі чи IT-персоналі.

Постачальники SaaS пропонують різноманітні програмні додатки, починаючи від інструментів продуктивності, таких як електронна пошта та офісні пакети, до більш спеціалізованих програм, таких як управління взаємовідносинами з клієнтами (CRM) і програмне забезпечення для планування ресурсів підприємства (ERP). Ключова перевага використання SaaS полягає в тому, що компанії можуть отримати доступ до цих програм за

вимогою, не купуючи й обслуговуючи власну апаратну та програмну інфраструктуру [17].

Рисунок 4.3 – SaaS

Однією з головних переваг SaaS є його масштабованість. Програми SaaS можна легко збільшити або зменшити залежно від потреб бізнесу без додаткового апаратного чи програмного забезпечення. Це особливо корисно для компаній із мінливим попитом, оскільки вони можуть коригувати використання програми, не несучи зайвих витрат.

Ще однією ключовою перевагою SaaS є його економічна ефективність. Замість того, щоб інвестувати в дороге обладнання та програмне забезпечення, компанії можуть отримати доступ до SaaS-додатків через модель на основі підписки, сплачуючи лише за те, що їм потрібно, на основі кожного користувача або на місяць. Це означає, що компанії можуть уникнути початкових капітальних витрат, а також можуть заощадити гроші на поточному обслуговуванні та модернізації [18].

SaaS також пропонує підприємствам більшу гнучкість і мобільність. Оскільки програми SaaS доступні через веб-браузер або мобільний додаток, користувачі можуть отримати до них доступ з будь-якого місця, де є

підключення до Інтернету. Це полегшує компаніям керування віддаленими командами та дозволяє співробітникам працювати з дому чи в дорозі.

Безпека також є основною перевагою використання програм SaaS. Провайдери SaaS зазвичай вкладають значні кошти в заходи безпеки, щоб гарантувати, що їхні програми захищені від кіберзагроз, таких як витік даних і атаки зловмисного програмного забезпечення. Це означає, що підприємства можуть отримати вигоду від безпеки на рівні підприємства без необхідності інвестувати у власну інфраструктуру безпеки [16].

Одним із потенційних недоліків SaaS є втрата контролю над даними. Оскільки додаток розміщується та керується стороннім постачальником, підприємства можуть бути стурбовані безпекою та конфіденційністю своїх даних. Щоб зменшити цей ризик, постачальники SaaS зазвичай пропонують надійну політику безпеки та конфіденційності, а також дотримання галузевих стандартів і правил.

Підсумовуючи, SaaS революціонізував спосіб доступу компаній до програмного забезпечення та їх використання. Завдяки масштабованості, економічній ефективності, гнучкості, мобільності та перевагам безпеки SaaS став невід’ємним компонентом IT-стратегії багатьох компаній. Оскільки попит на хмарні програмні додатки продовжує зростати, цілком імовірно, що SaaS продовжить розвиватися та впроваджувати інновації, забезпечуючи підприємствам ще більшу цінність і переваги в найближчі роки.

PaaS — це модель служби хмарних обчислень, яка надає розробникам платформу для створення, розгортання та керування програмами (рисунок 4.4). На відміну від IaaS (інфраструктура як послуга), яка надає базові інфраструктурні ресурси, такі як сервери та сховище, і SaaS (програмне забезпечення як послуга), яка надає програмні додатки, PaaS пропонує повне середовище розробки для створення та розгортання веб-додатків [17].

Рисунок 4.4 – PaaS

PaaS надає середовище, розроблене таким чином, щоб полегшити розробникам створення та розгортання програм, не турбуючись про базову інфраструктуру. Постачальники PaaS пропонують ряд послуг, таких як сервери додатків, бази даних, проміжне програмне забезпечення та інструменти розробки, до яких розробники можуть отримати доступ і використовувати їх через веб-інтерфейс або API.

Основні переваги PaaS включають підвищення продуктивності, масштабованість і економію коштів. Надаючи розробникам готове середовище розробки, PaaS позбавляє розробників від необхідності витрачати час на встановлення та налаштування серверів, баз даних та інших компонентів інфраструктури. Це дозволяє розробникам зосередитися на написанні коду та створенні додатків, що може значно підвищити їх продуктивність [17].

PaaS також пропонує можливість швидко та легко масштабувати додатки, оскільки основною інфраструктурою керує постачальник. Це означає, що розробники можуть швидко додавати або видаляти ресурси за потреби, не турбуючись про базову інфраструктуру. Це може допомогти підприємствам заощадити кошти, оскільки вони платять лише за ресурси, які

використовують.

Ще одна перевага PaaS полягає в тому, що він забезпечує узгоджене середовище розробки протягом усього життєвого циклу розробки. Розробники можуть використовувати ті самі інструменти та технології від розробки до розгортання, що може допомогти оптимізувати процес розробки та зменшити кількість помилок.

PaaS можна використовувати для створення широкого діапазону додатків, від простих веб-додатків до складних корпоративних додатків. Постачальники PaaS пропонують низку мов програмування, фреймворків та інструментів, що дозволяє розробникам вибирати найкращий стек технологій для своєї програми.

Постачальники PaaS також пропонують низку варіантів розгортання, включаючи публічне, приватне та гібридне хмарне розгортання. Це дозволяє компаніям вибрати варіант розгортання, який найкраще відповідає їхнім потребам і бюджету.

Однак з PaaS також пов’язані деякі проблеми. Однією з головних проблем є прив’язаність до постачальника, оскільки компаніям може бути важко перейти до іншого постачальника PaaS після того, як вони інвестували в певну платформу. Крім того, компанії можуть зіткнутися з проблемами в інтеграції PaaS з існуючою IT-інфраструктурою, а також у забезпеченні безпеки та відповідності своїх програм.

Підсумовуючи, PaaS забезпечує повне середовище розробки для створення та розгортання веб-додатків, пропонуючи такі переваги, як підвищення продуктивності, масштабованість та економія коштів. PaaS можна використовувати для створення широкого спектру додатків і пропонує низку мов програмування, фреймворків та інструментів. Однак компанії повинні знати про виклики, пов’язані з PaaS, включаючи прив’язку до постачальника та інтеграцію з існуючою IT-інфраструктурою.

Хмарні обчислення пропонують кілька переваг порівняно з традиційними обчислювальними методами. Однією з головних переваг є

масштабованість. Хмарні ресурси можна збільшувати або зменшувати залежно від попиту. Це означає, що підприємства можуть уникнути необхідності інвестувати в дороге апаратне та програмне забезпечення, яке вони можуть використовувати не на повну потужність [16].

Cloud-технології також пропонують підвищену доступність. Завдяки хмарній технології користувачі можуть отримувати доступ до даних і програм з будь-якої точки світу, якщо у них є підключення до Інтернету. Це полегшує роботу компаній у всьому світі, а віддаленим співробітникам – доступ до ресурсів компанії.

Ще однією перевагою хмарних обчислень є економія коштів. Використовуючи хмарні ресурси, підприємства можуть заощадити гроші на апаратному забезпеченні, програмному забезпеченні та витратах на обслуговування. Хмарні постачальники зазвичай стягують плату з користувачів на основі використання, тому компанії платять лише за те, що вони використовують.

Безпека також є головною проблемою, коли йдеться про хмарні технології. Хмарні постачальники інвестують значні кошти в заходи безпеки для захисту даних своїх користувачів. Вони використовують шифрування, брандмауери та інші технології безпеки, щоб забезпечити безпеку даних і захист від несанкціонованого доступу.

Хмарні технології також уможливили розвиток нових технологій, таких як штучний інтелект (AI), машинне навчання (ML) та Інтернет речей (IoT). Ці технології значною мірою покладаються на обчислювальну потужність і масштабованість хмарних обчислень.

Загалом хмарна технологія революціонізувала спосіб зберігання, керування та доступу до даних для компаній і окремих осіб. Він пропонує ряд переваг, включаючи масштабованість, доступність, економію коштів і безпеку. Оскільки попит на хмарні технології продовжує зростати, ми можемо очікувати ще більшого прогресу в цій галузі в найближчі роки.

Хмарні постачальники — це компанії, які пропонують послуги

хмарних обчислень компаніям і окремим особам. Ці послуги можуть включати віртуальні сервери, сховище, бази даних і програмне забезпечення, серед іншого. Хмарні провайдери зазвичай стягують плату за свої послуги на основі оплати за використання, що означає, що клієнти платять лише за ті ресурси, які вони використовують, і можуть збільшувати чи зменшувати їх використання за потреби.

Існує багато хмарних провайдерів, але деякі з найбільших і найпопулярніших включають (рисунок 4.5):

- web-служби Amazon (AWS);
- Microsoft Azure;
- Google Cloud Platform (GCP).

Рисунок 4.5 – Основні провайдери Cloud-технологій

Кожен із цих хмарних провайдерів має свої сильні та слабкі сторони, і компанії та окремі особи можуть вибрати одного провайдера замість іншого залежно від своїх конкретних потреб і вподобань. Наприклад, AWS відома своїм широким спектром послуг і домінуванням на ринку, тоді як Microsoft Azure популярна серед підприємств, які використовують програмне забезпечення та інструменти Microsoft, а Google Cloud Platform відома своїм сильним фокусом на машинному навчанні та аналізі даних [17].

5 СТРУКТУРИЗАЦІЯ ПРОЕКТУ

5.1 Створення ієрархічної структури декомпозиції робіт

У світі управління проектами структура поділу робіт (WBS) є важливим інструментом, який використовується для поділу складного проекту на керовані частини. Цей інструмент дозволяє керівникам проектів визначати обсяг проекту, відстежувати прогрес і спілкуватися із зацікавленими сторонами. WBS служить основою для планування проекту, планування, оцінки витрат і розподілу ресурсів [6].

WBS — це ієрархічна декомпозиція проекту на більш дрібні, більш керовані компоненти. Кожен рівень WBS представляє більш детальну розбивку обсягу проекту, причому найнижчий рівень складається з окремих завдань або робочих пакетів, які можна призначити конкретному члену команди або групі. WBS надає візуальне представлення обсягу проекту, допомагаючи визначити основні результати проекту та організувати їх у структуровану структуру [7].

WBS зазвичай створюється на початку проекту та використовується протягом життєвого циклу проекту, щоб відстежувати прогрес і гарантувати, що всі результати виконані вчасно та в межах бюджету. Існує два типи WBS: орієнтована на продукт і орієнтована на процес. Орієнтована на продукт WBS зосереджується на результатах проекту, тоді як орієнтована на процес, зосереджується на діях, необхідних для отримання результатів. Орієнтована на продукт WBS використовується частіше, оскільки вона надає чіткішу картину обсягу проекту та результатів.

Мета WBS полягає в тому, щоб розбити складний проект на керовані компоненти, дозволяючи менеджерам проекту визначати обсяг проекту, відстежувати прогрес і спілкуватися із зацікавленими сторонами. WBS служить основою для планування проекту, планування, оцінки витрат і розподілу ресурсів. Він надає візуальне представлення обсягу проекту,

полегшуючи ідентифікацію основних результатів проекту та організацію їх у структуровану структуру. Розбиваючи проект на більш дрібні, більш керовані компоненти, WBS допомагає спростити процес управління проектом і зробити його більш ефективним [1].

WBS зазвичай створюється на початку проекту та використовується протягом життєвого циклу проекту, щоб відстежувати прогрес і гарантувати, що всі результати виконані вчасно та в межах бюджету. Це важливий інструмент для спілкування та співпраці між членами команди проекту, зацікавленими сторонами та клієнтами.

WBS можна створити за допомогою різноманітних методів, таких як мозковий штурм, інтерв'ю з експертами з відповідної тематики та посилання на існуючу проектну документацію. Його також можна створити за допомогою програмних засобів, які автоматизують процес створення та керування WBS. Програмні засоби WBS широко доступні та можуть допомогти автоматизувати процес створення та керування WBS. Ці інструменти зазвичай забезпечують візуальний інтерфейс для створення та редагування WBS, а також можуть містити функції для відстеження прогресу, керування ресурсами та створення звітів. Деякі популярні програмні засоби WBS включають Microsoft Project, Wrike і Trello. Результати та результати розділені на невеликі частини називаються «робочими пакетами». Робочий пакет має бути розроблений таким чином, щоб була можливість:

- оцініти вартість створеного робочого пакета;
- скласти графік роботи;
- контролювати робочий пакет.

Можна створити кілька можливих шаблонів WBS, незважаючи на те, що всі вони описують один і той же проект. Однак різні моделі WBS вимагають різних організаційних структур і стилів управління під час реалізації проекту. Таким чином, WBS-дизайнер вже має значний вплив на те, як керувати проектом на самому ранньому етапі подання пропозиції, іноді

навіть не підозрюючи про це. Невідповідність між WBS проекту, організаційною структурою та стилем управління менеджером проекту негативно впливає на ймовірність успішного завершення проекту. Хаос виникає, якщо різні сторони проекту виробляють різні WBS [6].

Що цікаво відзначити щодо WBS, його можна використовувати знову і знову для кількох проектів, якщо природа майбутніх проектів вимагає такого ж результату. Якщо характер проектів відрізняється від поточного, його також можна використовувати з деякими простими змінами.

Отже, для поточного IT-проекту була розроблена структура декомпозиції робіт WBS, що наведена на рис. 5.1.

Рисунок 5.1 – Структура декомпозиції робіт (WBS)

Є кілька переваг використання WBS, зокрема:

– WBS надає структуровану основу для планування проекту, допомагаючи керівникам проектів визначити основні результати проекту та

організувати їх у ієрархічну структуру;

− розбиваючи проект на більш дрібні, більш керовані компоненти, WBS допомагає визначити ресурси, необхідні для кожного завдання або пакету робіт, дозволяючи менеджерам проектів розподіляти ресурси ефективніше;

− полегшує спілкування із зацікавленими сторонами та членами команди про цілі, завдання та прогрес проекту;

− WBS допомагає спростити процес управління проектом і зробити його ефективнішим;

− WBS допомагає визначити потенційні ризики та зменшити їх до того, як вони стануть серйозними проблемами, визначаючи основні результати проекту та організовуючи їх у структуровану структуру.

Хоча WBS є цінним інструментом управління проектами, він має свої обмеження. По-перше, це може бути трудомістким і дорогим для створення, особливо для складних проектів. По-друге, може бути важко точно оцінити тривалість і вартість кожного пакету робіт. По-третє, змінами WBS може бути важко керувати, особливо якщо вони відбуваються на пізній стадії життєвого циклу проекту. По-четверте, WBS може не підходити для всіх типів проектів, особливо для тих, які є високоітеративними або покладаються на гнучкі методології.

Підсумовуючи, структура розподілу робіт (WBS) — це важливий інструмент управління проектом, який забезпечує ієрархічне декомпозоване представлення завдань і результатів проекту. Це основа для планування проекту, складання розкладу, оцінки витрат і розподілу ресурсів, а також корисна для визначення критичних дій на шляху, полегшення спілкування та запобігання розповзанню обсягу. Хоча WBS має свої обмеження, він залишається цінним інструментом для управління проектами, особливо коли використовується в поєднанні з програмними засобами, які можуть допомогти автоматизувати процес створення та керування WBS.

5.2 Створення організаційної структури виконавців

В управлінні проектами Структура розподілу організації (OBS) є цінним інструментом для організації та визначення ролей і обов’язків членів команди в організації [6]. OBS — це ієрархічна модель, яка розбиває організацію на менші частини або підрозділи, кожна з яких має власний набір ролей, обов’язків і структур звітності. У цьому документі надано детальний огляд OBS, включаючи його визначення, переваги та приклади того, як його можна використовувати в різних організаціях.

OBS — це графічне представлення, яке розбиває організацію на менші одиниці, причому кожна одиниця представляє іншу організаційну функцію або відділ. OBS зазвичай використовується в поєднанні з іншими інструментами управління проектами, такими як структура розподілу робіт (WBS), щоб допомогти визначити обсяг проекту та розподілити завдання та обов’язки між членами команди.

За своєю суттю OBS забезпечує основу для визначення ролей і обов’язків членів команди в організації. Розбиваючи організацію на більш дрібні підрозділи, OBS допомагає переконатися, що всі працюють на однакові цілі.

Є кілька переваг використання OBS в управлінні проектами. По-перше, OBS надає чітке та стисле уявлення про організацію, полегшуючи розуміння того, як різні частини організації пов’язані та як вони працюють разом. Це може бути особливо цінним у великих організаціях або складних проектах.

По-друге, OBS допомагає визначити ролі та обов’язки членів команди в організації, забезпечуючи, щоб усі були узгоджені та працювали над тими самими цілями. Це може допомогти покращити комунікацію та співпрацю в організації, оскільки кожен розуміє свою роль і те, як він вписується в загальну картину.

По-третє, OBS може допомогти виявити потенційні проблеми або проблеми в організації. Розбиваючи організацію на більш дрібні підрозділи, легше визначити сфери, де можуть бути збіги або прогалини у

відповідальності, і вжити заходів для вирішення цих проблем, перш ніж вони стануть серйознішими.

Створення OBS може бути складним процесом і вимагає ретельного планування та координації.

Спочатку потрібно визначити загальну структуру організації: перш ніж створювати OBS, важливо мати чітке розуміння загальної структури організації. Це включає визначення різних відділів або команд в організації, а також ролей і обов'язків кожної команди. Встановлення рівнів OBS: після визначення загальної структури наступним кроком є визначення рівнів OBS. Це передбачає поділ організації на більш дрібні підрозділи або департаменти та визначення відносин звітності між цими підрозділами.

Розподіл ролей і обов'язків: після визначення рівнів OBS наступним кроком буде розподіл ролей і обов'язків для кожного рівня. Це гарантує, що кожен в організації розуміє свої ролі та обов'язки, а також те, як вони вписуються в загальну картину.

Зв'язки звітності є важливою частиною OBS, оскільки вони визначають, як інформація перетікає між різними рівнями організації. Важливо розробити чіткі та лаконічні звітні стосунки, щоб гарантувати, що кожен в організації розуміє, як інформація повинна ділитися та передаватись.

Перегляд і вдосконалення OBS: після того, як OBS розроблено, важливо переглянути та вдосконалити його, щоб переконатися, що він точно відображає структуру та цілі організації. Це може передбачати внесення змін до рівнів OBS, коригування ролей і обов'язків або перегляд відносин звітності.

Структура розбивки організації об'єднує подібні заходи проекту або «робочі пакети» та пов'язує їх із структурою організації. OBS (також відома як організаційна структура розбивки) використовується для визначення відповідальності за управління проектом, звітність про витрати, виставлення рахунків, складання бюджету та контроль проекту. OBS забезпечує організаційну, а не задачну перспективу проекту. Ієрархічна структура OBS

дозволяє агрегувати (згортати) інформацію про проект на вищих рівнях. Коли обов'язки за проектом визначені та призначено роботу, OBS і WBS підключаються, надаючи аналітику потужну можливість вимірювати продуктивність проекту та робочу силу на дуже високому рівні (приклад продуктивності бізнес-підрозділу) або до деталей (приклад роботи користувача над завдання).

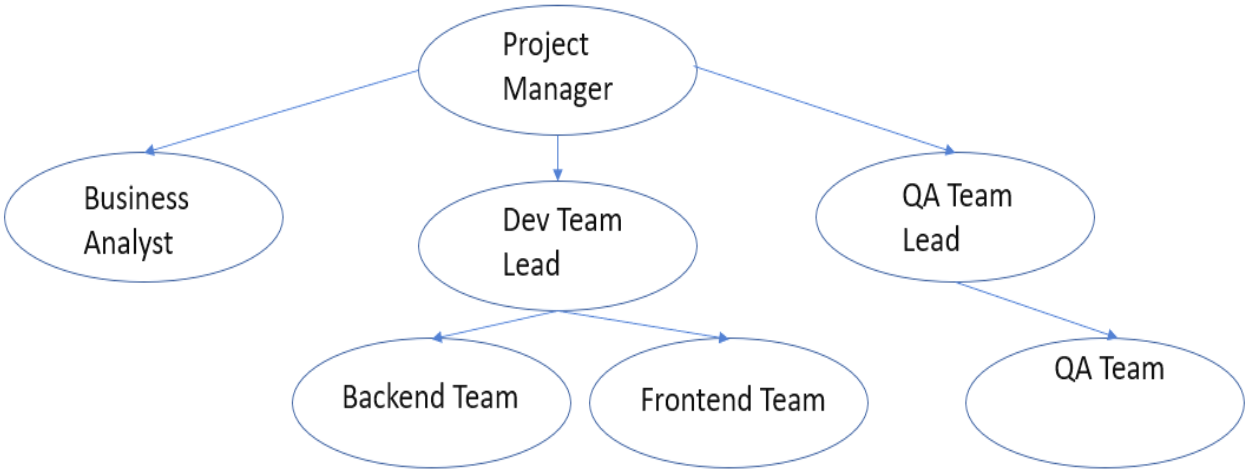

Рисунок 5.2 – Організаційна структура виконавців (OBS)

Для того, щоб можна було переглянути всі операції, призначені до виконання конкретною людиною, або відобразити всіх людей, які беруть участь у даній операції було розроблено матрицю відповідальності, яка називається діаграмою RACI (таблиця 5.1).

Таблиця 5.1 – Матриця відповідальності в форматі RACI (В – Відповідальний; П – Підзвітний; К – Консультант; О – Одержувач)

Діаграма RACI		Учасники проекту			
№	Операція	Project Manager	Business Analyst	Dev Team	QA Team
1	2	3	4	5	6
1	Аналіз продукту				
1.1	Аналіз ринку	О	В		
1.2	Встановлення бізнес задач	О	В		
2	Планування				
2.1	Розробка календарного плану	В	К	К	К
2.2	Визначення строків	В	К	К	К
2.3	Ухвалення ресурсів	В	К	К	К
3	Реалізація технічної частини				
3.1	Реалізація Frontend				
3.1.1	Створення шаблону	О		В	О
3.1.2	Створення сторінки авторизації	О		В	О
3.1.3	Створення сторінки реєстрації	О		В	О
3.1.4	Створення сторінки налаштувань аккаунта	О		В	О
3.1.5	Створення сторінки додавання вигрузок	О		В	О
3.1.6	Створення сторінки видалення вигрузок	О		В	О
3.1.7	Створення сторінки редагування вигрузок	О		В	О
3.1.8	Інтегрування з Backend REST API	О		В	О

ВИСНОВКИ

В результаті виконання кваліфікаційної роботи було розглянуто декілька платформ для маркетплейсів. Під час огляду платформ було порівняно формати даних, що найчастіше використовуються на сьогоднішній день. Також було досліджено та порівняно архітектурні стилі розробки ПЗ для проектування можливої реалізації системи управління різними форматами даних.

Під час виконання роботи були поставлені та виконані такі завдання:

– досліджено платформи, що використовуються для маркетплейсів

– виконано аналіз різних форматів даних, що використовуються різними платформами під час процесів їх роботи;

– виконано порівняння архітектурних стилів розробки ПЗ;

– встановлено можливі варіанти технологій, що можуть бути використані при реалізації системи

В кінцевому результаті можна зробити висновок, що необхідність такої системи зростає з кожним днем, тому було спроектовано модель системи управління даними.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. ДСТУ 3008:2015. Інформація та документація. Звіти у сфері науки і техніки. Структура і правила оформлювання. – Чинний від 22.06.2015. – Київ: ДП «УкрНДНЦ», 2016. – 31 с.

2. ДСТУ 8302:2015. Інформація та документація. Бібліографічні посилання. Загальні положення та правила складання. – Чинний від 04.03.2016. – Київ: ДП «УкрНДНЦ», 2016. – 20 с.

3. Катренко А.В. Управління IT-проектами. Стандарти, моделі та методи управління проектами. Підручник. Л: Новий Світ-2000, 2013. – 550с.

4. Маркетплейси – що це таке простими словами // https://elit-web.ua/ua/blog/chto-takoe-marketplejsy (date of access: 25.04.2023).

5. Методичні вказівки щодо розробки та оформлення магістерської кваліфікаційної роботи за спеціальністю 122 Комп'ютерні науки (освітня програма «Управління проєктами в галузі інформаційних технологій» освітньо-кваліфікаційного рівня «магістр» / Упоряд.: Петров К.Е., Левикін В.М., Чалий С.Ф., Євланов М.В., Саєнко В.І., Міхнов Д.К., Міхнова А.В., Чала О.В. – Харків: ХНУРЕ, 2021. – 28 с.

6. На який маркетплейс заходити інтернет-магазину: дві моделі заробітку в Україні // https://rau.ua/dosvid/na-kakoj-marketplace/ (date of access: 07.04.2023).

7. Операційна команда маркетплейса: структура, метрики, бюджети. Колонка CEO Hubber // https://retailers.ua/uk/news/mneniya/12381-operatsionnaya-komanda-marketpleysa-struktura-metriki-byudjetyi (date of access: 20.04.2023).

8. Прайс агрегатори та Маркетплейси України та світу // https://pricecontrol.biz/uk/monitorynh-tsin-na-marketplejsakh-i-prajs-ahrehatorakh-shcho-vybraty/ (date of access: 12.04.2023).

9. Приймак В. М. Управління проектами. Збірник кейсів. Навчальний посібник. К.: Київський національний університет імені Тараса Шевченка, 2020. – 220с.

10. Приймак В. М. Управління проектами. Навчальний посібник. К.: Київський національний університет імені Тараса Шевченка, 2017. – 464с.

11. Продажі через маркетплейси: з чого почати і як досягти успіху // https://ag.marketing/blog/prodazhi-cherez-marketplejsi/ (date of access: 10.04.2023).

12. Спробуй продати: як обрати маркетплейс для ведення бізнесу, їх «підводні камені» та переваги // https://ucap.io/sprobuj-prodaty-yak-obraty-marketplejs/ (date of access: 12.04.2023).

13. ТОП-10 ідей для нішевих маркетплейсів // https://ua.scallium.pro/marketplace-business-models (date of access: 30.03.2023).

14. Amazon, Ebay, Alibaba, OLX, Rozetka та інші: як вибрати ідеальний майданчик для продажів // https://www.epravda.com.ua/publications/2019/10/28/653012/ (date of access: 02.04.2023).

15. Erik Maier. Acquiring customers through online marketplaces? The effect of marketplace sales on sales in a retailer's own channels [Електронний ресурс] – Режим доступу до ресурсу: https://www.sciencedirect.com/science/article/abs/pii/ S0167811620300835

16. Sha Zhang The Impact of Adding Online-to-Offline Service Platform Channels on Firms' Offline and Total Sales and Profits [Електронний ресурс] – Режим доступу до ресурсу: https://doi.org/10.1016/j.intmar.2019.03.001

17. Francisco Freitas A methodology for refactoring ORM-based monolithic web applications into microservices [Електронний ресурс] – Режим доступу до ресурсу: https://doi.org/10.1016/j.cola.2023.101205

18. Simon Schneider Automatic extraction of security-rich dataflow diagrams for microservice applications written in Java [Електронний ресурс] – Режим доступу до ресурсу: https://doi.org/10.1016/j.jss.2023.111722

19. Mohd Naved Identifying the role of cloud computing technology in management of educational institutions [Електронний ресурс] – Режим доступу до ресурсу: https://doi.org/10.1016/j.matpr.2021.11.414

20. Fabio Palumbo Characterization and analysis of cloud-to-user latency: The case of Azure and AWS [Електронний ресурс] – Режим доступу до ресурсу: https://doi.org/10.1016/j.comnet.2020.107693

21. Alberto Artasanchez AWS for Solutions Architects: Design your cloud infrastructure by implementing DevOps, containers, and Amazon Web Services, PacktPublishing, 2021. – 454 с.