НАЦІОНАЛЬНИЙ УНІВЕРСТИТЕТ БІОРЕСУРСІВ
І ПРИРОДОКОРИСТУВАННЯ УКРАЇНИ

Факультет інформаційних технологій

УДК 004.9:631.16

«ПОГОДЖЕНО»
Декан факультету
Глазунова О.Г., д.п.н., професор
_____
_____ **202_ р.**

«ДОПУСКАЄТЬСЯ ДО ЗАХИСТУ»
Завідувач кафедри комп'ютерних наук інформаційних технологій
Голуб Б.Л., к.т.н., доцент
_____
_____ **202_ р.**

МАГІСТЕРСЬКА КВАЛІФІКАЦІЙНА РОБОТА

на тему _____Аналітична система обліку аграрного підприємства_____

Спеціальність ___122 ____Комп'ютерні науки_____
(код і назва)

Освітня програма _Інформаційні управляючі системи та технології____
(назва)

Орієнтація освітньої програми _освітньо-професійна_____
(освітньо-професійна або освітньо-наукова)

Гарант освітньої програми

_____ _____ _____
(науковий ступінь та вчене звання) (підпис) (ПІБ)

Керівник магістерської кваліфікаційної роботи

_____ _____ _____
(науковий ступінь та вчене звання) (підпис) (ПІБ)

Виконав _____ **Гожий П.П.**_____
(підпис) (ПІБ студента)

КИЇВ-2022

НАЦІОНАЛЬНИЙ УНІВЕРСТИТЕТ БІОРЕСУРСІВ І ПРИРОДОКОРИСТУВАННЯ УКРАЇНИ

Факультет (ННІ) _____

ЗАТВЕРДЖУЮ

Завідувач кафедри_____

_____

_____ _____ _____

(науковий ступінь, вчене звання) (підпис) (ПІБ)

“ ” _____ 20 ___ року

З А В Д А Н Н Я

ДО ВИКОНАННЯ МАГІСТЕРСЬКОЇ КВАЛІФІКАЦІЙНОЇ РОБОТИ СТУДЕНТУ

_____

(прізвище, ім'я, по батькові)

Спеціальність _____

(код і назва)

Освітня програма _____

(назва)

Орієнтація освітньої програми_____

(освітньо-професійна або освітньо-наукова)

Тема магістерської кваліфікаційної роботи_____

_____

затверджена наказом ректора НУБіП України від “ ” _____ 20 ___ р. №_____

Термін подання завершеної роботи на кафедру _____

(рік, місяць, число)

Вихідні дані до магістерської кваліфікаційної роботи _____

_____

_____

Перелік питань, що підлягають дослідженню:

1. _____
2. _____
3. _____

Перелік графічного матеріалу (за потреби) _____

_____

Дата видачі завдання “ ” _____ 20 ___ р.

Керівник магістерської кваліфікаційної роботи _____ _____

(підпис) (прізвище та ініціали)

Завдання прийняв до виконання _____ _____

(підпис) (прізвище та ініціали студента)

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

1. Рис. – рисунок
2. Табл. – таблиця
3. СЕД – система електронного документообігу.
4. СКП – Система комп'ютерного прогнозування
5. НП – населений пункт
6. BI - Business Intelligence
7. ER – Entity Relation
8. КП – комп'ютерне прогнозування
9. ВВП – внутрішній валовий продукт
10. DSV – Data Source View
11. СУБД – система управління базою даних
12. HOLAP – Hybrid OLAP.
13. СД – сховище даних
14. KPI – Key Performance Indicator
15. MOLAP – Multidimensional OLAP
16. OLAP – On-Line Analytical Processing.
17. ROLAP – Relational OLAP
18. SSAS – SQL Server Analysis Services.
19. SQL – Structured Query Language
20. UUID – Universally unique identifier
21. JSON – JavaScript Object Notation
22. SSRS – SQL Server Reporting Services.
23. XML -Extensible Markup Language
24. ANSI – American National Standards Institute
25. АС ППР – автоматизована система претензійно-позовної роботи
26. ISO – International Organization for Standardization

27. ACID – Atomicity, Consistency, Isolation, Durability
28. PL – Procedural Language
29. REST – Representational State Transfer
30. API – Application Programming Interface
31. URI – Uniform Resource Identifier
32. HTTP – Hyper Text Transfer Protocol
33. JDBC – Java Database Connection
34. IDE – Integrated development environment
35. HTML – Hyper Text Markup Language
36. CSS – Cascading Style Sheets
37. ООП – об'єктно-орієнтоване програмування
38. SOLID – single responsibility, open-closed, Liskov substitution principle, interface segregation, dependency inversion
39. AJAX – Asynchronous JavaScript and XML.
40. ORM – Object Relation Mapping
41. EE – Enterprise Edition
42. ELK – Elastic, Logstash, Kibana

ВСТУП

Аграрна галузь завжди займала одне з провідних місць в економіці України. Це і не дивно, важко представити країну, яка б не мала власного аграрного сектору, залежність всього продовольства лише на імпорт має величезний спектр проблем. Не менш важливою є інформаційна частина самою галузі, яка може збільшити продуктивність підприємства в геометричність прогресії.

У кожній країні, у будь-якому суспільстві сільське господарство є життєво-необхідною галуззю господарства, оскільки зачіпає інтереси буквально кожної люди. Аграрна галузь в Україні на даний момент є провідною, є фундаментом нашої економіки.

Робота в аграрному секторі є нелегкою та потребує від людини певних фахових знань і досвіду, а також особливого ставлення до землі. У процесі аграрного розвитку тільки в Україні задіяно мільйони людей. Це і агрономи, і агроінженери, і економісти, і бухгалтера, і багато інших професій. Насправді, бути причасним до агро-процесу можна навіть не знаючи про це. В цій сфері задіяна навіть така несподівана професія, як програміст.

Насправді, роботи для програміста в цій галузі дуже багато. Від автоматизації процесів збору урожаю до систем обліку і прогнозування, такі різні, але від цього не менш важливі програми.

Актуальність даної роботи полягає в тому, що якщо будь-яке підприємство, а тим паче аграрне, хоче мати можливість вести свою діяльність і робити це ефективно, то потрібно мати таку систему, яка б могла задовольнити будь-які потреби обліку, аналітики та фінансів.

Метою дослідження є перевірка доцільності використання технологій Data Mining і загалом інформаційних систем для введення діяльності аграрного підприємства. Які можливі способи оптимізації процесів, яка архітектурні рішення більше підходять для введення діяльноеті такого роду.

Завданням даної роботи можна визначити опис технічних вимог, побудова архітектури системи дослідження та розробка рішення на основі цього. Після цього потрібно буде виконати збір даних, їх обробка та виведення результати, які цілі були досягнені. На основі цього можна порівняти, які покращення надала розробка аналітичної системи. Для проведення аналізу даних та підтвердження гіпотез, розрахунку показників ефективності та прикладних параметрів використовується інструменти програмного обчислення. Розглядаються типові задачі підприємницької діяльності, та час їх виконання за допомогою аналітичної системи та іншої системи, окремо визначено ліквідність розробки. Запропоновано сукупність технологій реляційних, не-реляційних баз даних, програмних інструментів та інструментів фронт-розробки для перевірки доцільності використання цих інструментів у аналітичний діяльності аграрного підприємства, було запропоновано використання різних видів баз даних для перевірки удосконалення роботи системи.

1. СИСТЕМНИЙ АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Опис предметної області

На сьогоднішній день перед людством гостро постають проблеми пов'язані з вирощуванням сільськогосподарських культур. Пшениця, кукурудза, соняшник – це лише одні з основних пропозицій на світовій біржі зерна, які у великій кількості імпортуються з території України, яка є одним з провідних імпортерів сільськогосподарських культур. В основному, це пов'язано з площею країни та родючістю ґрунтів, на яких вирощується зерно. Велика кількість країн просто не може собі дозволити вирощувати таку кількість зерно, а деяким можливо просто вигідніше просто закуповувати його, за рахунок прибутку з інших галузей, що можуть принести більший прибуток, не потребують природних факторів, таких як родючість ґрунту, великі поля. Такий підхід, як правило, досягаються за рахунок збуту продукції, або переробки сировини в зерна в продукцію. Тож, можна підвести підсумок про те, що для аграрного підприємства не найменш значущу роль відіграє логістика, при чому дуже глобальна логістики трансферів між різними країнами, материками, з використанням усіх можливих видів транспорту: вантажівки далеких дистанцій, залізничний транспорт, морський транспорт і можливо навіть літаки, які є найдорожчим видом транспортування і основна перевага яких є швидкість транспортування на далекі дистанції. Але якщо розглядати залізничний транспорт і морський – то вони мають дуже рівні позиції в логістиці. З цього слідує, що система повинна надавати можливість вирахувати найдоцільніший вид доставки продукції.

Об'єктом даної предметної області є аналіз діяльності аграрного підприємства. Тож, предметом - аналітична система обліку діяльності аграрного

підприємства. А метою є визначити ключові показники ефективності аграрного підприємства.

Якщо вже розробляти масштабну аграрну систему як підприємство, яке відігравало б роль в макроекономічних масштабах, то слід розглянути аграрне підприємство не тільки як виробника, а як і реалізатора продукції, при чому не тільки своєї, а й інших фермерів. Наприклад, якщо ми маємо декілька фермерів, які працюють окремо один від одного, але вирощують одну й ту саму культуру, наприклад зерно і одне підприємство, більш масштабне, яке може викупити продукцію і продати її потім великою партію дорожче, то в системі потрібно мати функцію, яка дозволяла вести такий облік.

1.2 Визначення вимог

Вимоги до системи є доволі комплексні і їх буде доцільно розділити на дві складові: функціонал підприємницької діяльності аграрного підприємства та аналітична складова підприємства.

До функціоналу діяльності аграрного підприємства можна віднести:

- Облік для постання продукції (як зовнішні постачальники, так і власне виробництво);
- Облік складу (кількість продукції, вид, опис, ресурс постачання, час прибуття, час відбуття);
- Облік клієнтів (контактна інформація клієнт, інформації про транзакції);
- Бухгалтерський облік (бухгалтерські рахунки та транзакції між цими рахунками).

До аналітичної складової можна віднести:

- Аналіз продуктивності працівників;
- Аналіз логістичних можливостей підприємства та їх рентабельність;

- Ключові показники ефективності підприємства та окремих працівників;
- Розрахунок можливого виробництва продукції на основі наявних інструментів та потужності підприємства;

Тепер, розглянемо складові більш детально.

Функціонал діяльності аграрного підприємства – це те, чим займається саме підприємство, а саме вирощення, закупівлі, продажі, бухгалтерський облік, облік тієї продукції, що пов'язана з цим підприємством.

Вирощення – це один з ключових процесів у роботі підприємства. Вирощення відноситься як до рослинництва, так і до тваринництва. Для початку треба добути ресурс з якого можна виростити худобу або рослину. Це може бути як власний збір насіння або розмноження тварин(наприклад, курчат), так і закупівля. Система повинна дозволяти ввести облік таких речей і зберігати всі потрібні параметри для цього. Після того, як, наприклад, насіння посадили, це повинно бути помічено в системі. І останнє, це збір урожаю, який теж зберігається в системі.

Закупівля передбачає як покупку вже готової продукції, для можливого подальшого збуту (арбітраж трафіку), так і покупка сировини для подальшого вирощення.

Продаж – це реалізація продукції, яка зберігається на складі. Є клієнти зі своєю контактною інформацією, яка дозволяє вести продажі й є окремі сутності для фіксування проведених транзакцій в системі.

Бухгалтерський облік відповідає за типову систему обліку фінансів підприємства за системою дебет-кредит, яка складається всього лиш з двох сутностей, сутність рахунків, між якими переходять фінанси, це саме бухгалтерські рахунки, які не мають нічого спільного між використанням банківських рахунків. А транзакції – це сутність грошових переказів між

рахунками, яка має поля вхідного рахунку, вихідного рахунку, суму транзакції, час транзакції та призначення транзакції.

Обік продукції передбачає зберігання її на складі, та збереження все можливих назв продукції, її опис, категорії, одиницю вимірювання товару.

Аналітична складова є в першу чергу залежною від основної діяльності аграрного підприємства, але не є від цього менш важливо – бізнес, що хоче розвиватись, повинен проводити апроксимацію наявних даних для прогнозування майбутньої ситуації на ринку, які потім будуть використанні для прийняття рішень. Крім того, важливим я загальна оцінка користі працівників, яку роботу було виконано, які знання працівник підвищив.

Також до аналітики можна віднести те прокладання логістичний шляхів, так наприклад, маємо адрес, продавця товару і маємо локацію складів, можна скласти можливий маршрут, який транспортом буде дешевше доставити продукцію. Так само це працює з клієнт, якому треба доставити товар. Ця функція не завжди може бути корисною, так як є ситуації, коли доставкою займається клієнт або продавець, або окрема логістична компанія. Але все одно, дуже приємно мати додаткову можливість, яка може допомогти в роботі і збільшити прибуток(зменшити витрати).

І останнє, що можна відзначити до аналітичних можливостей в розроблюваній аналітичній системі обліку діяльності аграрного підприємства. Програмний засіб може мати доступ до наявних територій, на яких можна висадити рослини, яка кількість насіння є для вирощення, скільки людей може зайнятись цим, скідьки техніки є для цього, яку врожайність можна отримати у підсумку. Так само і з тваринництвом, я хліви, є кількість корму, її види, так як різні корма підходять для різної худоби, є сама худоба, я люди, які можуть зайнятись цією худобою, збором молока, яєць, м’яса. Якщо збір виконується автоматично, то це теж можна використати як незалежну зміну у цьому виразі.

Окремо для рослинництва може буде доцільно вираховувати використання тепличного методу вирощування.

1.3 Аналіз вимог до програмної та апаратної системи

Дослідивши основні вимоги до самої системи, її функціональності, можна перейти до не менш важливого пункт, а саме, вимоги до програмної системи. Важливість даного пункту полягає в новизні самої магістерської роботи, а саме використання різних видів баз даних, для вирішення різних задач, що виникають при використанні аналітичної системи обліку аграрного підприємства.

Вимоги до програмної системи містять деякий набір інструментів для серверної та клієнтської. Такий підхід зумовлений клієнт-серверною архітектурою системи, яка була використана при розробці аналітичної системи.

Для клієнту буде важливо мати браузер з підтримкою нових версій JavaScript. На даний момент останній реліз довгої підтримки є ECMAScript 2020. Крім того, наявність мінімум 4 гігабайтів пам’яті була би достатньою для користуванням сайту. Майже вся логіка виконується на стороні серверу, тож не потрібно мати потужний процес, буде достатньо майже будь-якого процесору, що підтримують сучасні систем – проблеми скоріш виникнуть з встановлення та відкриття браузеру, чим відправлення запиту для отримання документу сайту.

Що ж відноситься до вимог серверної частини, то тут все набагато складніше. Серед інструментів, що буде потрібно, в першу чергу, це засіб для запуску коду Java. Крім цього архів скомпільованого коду повинен бути запущений на сервері додатків. В ході розробки був використаний сервер додатків Tomcat. Серед програмних засобів також варто відмити бази даних. Серед тих, що потрібні в ході цієї розробки, можна відзначити PostgreSQL, Apache Kafka, Redis, MongoDB. Вибір цих інструментів буде розглянуто в наступних розділах з описом їх плюсів і мінусів, а також областями використання.

1.4 Аналіз наявних рішень

Існуючі рішень для введення аналітичної діяльності аграрного підприємства можна розділити на дві категорії: електронні інформаційні системи, що є найкращим рішенням і системи паперового обліку і аналітики, які з кожним роком відходять на другий план. Це й логічно, насправді немає причин використовувати цей підхід, тож ми розглянемо тільки чому варто не використовувати це рішення.

Серед проблем аналітичних систем обліку є їх застарілість та функціональність, якої недостатньо для вирішення сучасних задач. В цьому випадку одна проблема випливає з іншої – велика кодова база, що потребує підтримки просто не має її. Можливість розробки просто стає неможливою. Нова функціональність коштує бізнесу занадто багато, розробити можливо тільки той функціонал, що має підтверджену бізнес-вартість.

Якщо розглядати з точки зору найбільш подібного, то найбільш підходящим буде порівняти аналітичну систему з ERP системами. ERP – це організована стратегія інтеграції виробництва і операцій, управління ресурсами праці, фінансового менеджменту і управління активами, орієнтованого на безперервне балансування і оптимізацію ресурсів підприємства за рахунок спеціалізованого інтегрованого пакету прикладного програмного забезпечення, який надає загальну модель даних і процесів для всіх сфер діяльності. З цього виходить, що ERP-система, це конкретний прикладний пакет, який реалізує стратегію ERP.

Серед найпопулярніших розробників ERP-систем зараз можна виділити SAP, Oracle, Microsoft, які на 2010 рік мають частки ринку 24%, 18% та 11% відсоток відповідно. Розглянемо всі ці системи окремо.

SAP була одною з перших ERP-систем, розроблених на початку 1970-х років. Таким чином, у розробників було багато часу допрацювати розширення функціональних можливостей системи. Перевагами SAP є те, що вона має дуже

багато модулів і може легко підтримувати прості і складні бізнес-процеси, в тому числі:

- бухгалтерський облік;
- ланцюг поставок;
- виробництво;
- управління замовленнями на продаж;
- управління складом;
- виконання;
- планування виробництва;
- управління якістю.

Це лише деякі з усіх бізнес-можливостей SAP. Але без мінусів теж не обійшлося. SAP дуже складний, тому що він покладається на так звані «коди транзакцій» для доступу до різних функцій програмного забезпечення. Хоча з останніми версіями все стало трохи краще, але система все одно є дуже складною і потребує доволі довгого навчання користувачів для використання її. Крім того, система дорожче за інші рішення, а програмне забезпечення потребує довгого тестування перед розгортанням. З цього можна зробити підсумок, SAP – хороший вибір для великих підприємств або компаній середнього розміру з важкими бізнес-процесами, високими цілями, або і тим, і іншим.

Корпорація Oracle була створена в 1977 році, майже ж тоді, коли SAP. Вона починала як розробник програмного забезпечення для управління реляційними базами даних і після цього використовувала свою технологію в якості основи для ERP-рішень. Для багатьох великих бізнесів вибір дистриб'ютора ERP зводиться тільки до Oracle ERP і SAP. Це і логічно, так як вони контролюють більшу частину ринку для великого бізнесу. Але Oracle ERP може бути використана не тільки для великого бізнесу. Деякі невеликі компанії можуть такого розгорнути її. Серед плюсів використання системи є те, що система була високо оцінена за свою гнучкість і широкий спектр

вузькоспеціалізованих компонентів, які роблять її фаворитом на таких ринках. Ця гнучкість означає, що більшість клієнтів можуть реалізувати тільки ті функції, що їм потрібно, щоб уникнути розгортання величезної кількості модулів, які не використовуються. Цей варіант дає явну перевагу перед іншими системами. Інша сторона цього всього є те, що велика кількість компонентів може збити з пантелику. Іноді не ясно, які з модулів потрібні твоєму бізнесу. Тим не менш, добре спроєктована система, що може бути досягнути за рахунок досвідчених партнерів, що розгортають та налаштовують систему може допомогти вибрати правильний набір продуктів. Ціна налаштування може бути великою при великій кількості використаних компонентів. Тим не менш, це дешевше за SAP, якщо вибір модулів був правильним. В підсумок можна сказати, що більшість користувачів задоволені системою, яку вони купили і збираються використовувати її далі. Система є хорошим вибором для організацій великого і середнього розміру, яким потрібна велика гнучкість і функціональність для підтримки конкретних бізнес-процесів.

Ще одним існуючим рішенням є Microsoft Dynamics, яка є не окремою ERP-системою, а цілим брендом для різноманітних систем, що підходять для різних задач. Компанія Microsoft продавала дані системи окремі одна від одної протягам декількох років. Але нещодавно був проведений ребрендинг пакету даних рішень і кожна система в пакеті мала свою чітку направленість.

- Microsoft Dynamics 365 Finance і Microsoft Dynamics 365 Supply Chain Management, ці два додатки орієнтовані на управління фінансами і операціями дистрибуції відповідно. Вони можуть бути розгорнуті як окремо, так й інтегровані як комплексна система;
- Microsoft Dynamics 365 Business Central – базова ERP-система для малого та середнього бізнесу, яка має в собі множину модулів для фінансів, операцій, продажу та маркетингу.

- Microsoft Dynamics GP – більш складна, комплексна система, чим попередня, вона включає ще більше модулів, серед яких управління запасами, персонал, бізнес-аналітикою і звітністю.

Всі ці варіанти систем можуть бути розгорнуті як локально, так і в хмарному середовищі, а також можливо гібридне розгортання, комбінацію яких може вибрати для себе кожен клієнт окремо.

Серед плюсів можна відзначити, що ця система має інтерфейс, який інтуїтивно зрозумілий порівняно з іншими системи. Дизайн є типовим для Microsoft і багато схожих елементів, які є в офісному пакеті програм, таких як Word, Excel, PowerPoint та інші. Хоча користувачам все одно треба деякий час для навчання користуванням цими додатками для повсякденних операцій, їм не потрібно вивчати нові правила користування і термінологію інтерфейсу.

Серед мінусів, можна зазначити, що підприємства зі складними або розширеними бізнес-процесами можуть виявити, що продукти Microsoft Dynamics ERP не є повними рішеннями без широких налаштуваннях або складної інтеграції з існуючими системами. Ті, хто розглядає Microsoft Dynamics, повинні враховувати ці елементи у вартості розгортання.

Розглядаючи витрати на саму ж систему, продукти Microsoft Dynamics ERP мають більш низьку ціну ліцензування, чим SAP або Oracle, але їх розгортання може зайняти більше часу. Більш тривалі цикли реалізації збільшують витрати на запуск, і, було зазначено вище, налаштування та інтеграція можуть зайняти ще більше збільшити ці затрати.

Дана система є хороший рішенням для малого і середнього бізнесу, які мають прості бізнес-процеси і потребують систему, можливості якої виходять за рамки їх поточних інструментів.

Враховуючи дані фактори, можна сказати, що використання SAP і Oracle важко порівняти з використанням системи, що розроблюється в даній роботі, так як в цих систем трохи інші цілі і ціна і функціонал набагато вище того, що було

розроблено. Але якщо розглянути Microsoft Dynamics, то тут все набагато цікавіше. Дана систем виграє в багатьох параметрах. Тож, розглянувши найпопулярніші рішення на ринку, можна сказати, що найбільш схожою системою є продукт від Microsoft. Він дозволяє робити те саме і навіть більше і коштує найменше серед цих систем. Такі параметри досить важливі, і роблять цю системою ідеальною. Але в ході даного проєкту є важливий нюанс, який виграє у даної системи – використання різних баз даних для вирішення різних задач. В системі від Microsoft є деякі інструменти, що дозволяють робити те саме, але логічним є те, найкраща інтеграція буде саме з іншими продуктами від Microsoft. А саме в цьому дослідженні були використані такі бази даних як PostgreSQL, Redis, MongoDB та Apache Kafka. Саме це основна задача роботи, тому і важливо використовувати саме ці інструменти.

1.4 Формулювання технічного завдання

Після того, як було розглянуто і повністю описано предметну область системи, було розглянуто програмні та технічні вимоги та порівняно з вже існуючими рішеннями, прийшов час створити завдання для технічного спеціаліста, який буде розробляти цю системи. Для цього нам допоможе технічне завдання. Задачею такого завдання є дати розуміння програмісту, як повинна виглядати система, яку потрібно розробити.

Для правильної коректної розробки додатку треба створити чітке технічне завдання, яким буде користуватись програміст під час розробки системи. Хороше технічне завдання не викликає запитанням у технічного спеціаліста, але на практиці, питання виникають завжди, тож ідеальне технічне завдання не існує, а до цього треба прагнути. Будь-який продукт не може бути розроблений без технічного завдання – це основа для моделювання системи, вибору архітектурних рішень і в подальшому розробки безпосередньо системи. Не слід також забувати, що окрім технічної складової можуть бути інші характеристики системи, наприклад, особливості дизайну.

Почнемо розгляд з першого і основного етапу, як для програміста – технічні характеристики. В цій частини буде дано відповіді на питання: якою повинна бути система? Які функції повинна виконувати? На які особливості розробки потрібно звернути увагу? Відповідь на ці питання дасть розуміння того, що очікувати від системи, та з якими труднощами можна зіткнутись. Розуміючи майбутні проблеми можна завчасно до них підготуватись спроєктувавши систему правильно.

Облікова система в першу чергу повинна займатись обліком. В аграрному підприємстві є декілька таких аспектів, серед яких фінанси, бо ціль будь-якого бізнесу – отримувати прибуток. Щоб знати який прибуток, які витрати, потрібно реалізувати можливість бухгалтерського обліку. Для отримання прибутки слід мати клієнтів, тобто ринок збуту своєї продукції, товару. Якщо є товар, то його треба десь зберігати, для цього потрібно ввести облік складу на якому зберігається продукція. Для товару також знадобиться довідник товарів – множина назв товарів з їх детальним описом. Крім того, аграрне підприємство може не тільки вирощувати самостійно рослини або тварин, а й закуповувати для подальшого збуту. Ці всі бізнес-вимоги формують деякий набір сутностей, що треба реалізувати в системі, так звану об’єктну модель. Від цієї моделі в першу чергу буде залежати набір сутностей у базі даних. Крім того, усім цим не може займатись одна людина, тож є декілька акторів, що означає, що на сайті повинна бути авторизація.

Склавши все разом можна виділити такі функціональні характеристики системи:

- Бухгалтерський облік;
- Облік дистрибуції товару;
- Облік отримання товару(закупівлі або самостійне вирощення);
- Облік складу
- Довідник інформацію про товари

Підсумувавши бізнес-процеси, що є в програмі, можна перейти до наступного етапу введення бізнесу, а саме аналітика. Без аналітики того, що відбувається в систему практично неможливо вести підприємницьку діяльність. Тож які аналітичні дані можна отримати, маючи такий функціонал? Перше і саме основне, це розрахунок отриманого прибутку за звітній період, потрібно мати аналітичний інтерфейс, який дозволив би виконати вибрати період часу, в який було проведено продажі і порахувати їх суму. Крім того, було би корисно отримати порівняння з минулим звітнім періодом. Іншим видом аналітичною діяльності можна визначити попит на деякий товар. Потрібна функція отримання інформації за декілька товарів, порівняння їх між собою та взагалі, який прибуток було отримано з цих товарів. Серед інших звітів, можна відзначити, наприклад, кількість не реалізованого товару – яка кількість ще знаходиться на складу, яка з нього, можливо, зіпсувалась через тривале зберігання. Це дає важливу інформацію для бізнесу, в якому напрямку рухатись, як краще будувати свій план.

Між іншим, є інші аналітичні дані, які було би корисно визначити, але на даних момент система не має таких функцій, але вже деякі сутності, які з цим пов’язані. Для прикладу можна взяти коефіцієнт корисності співробітників, які займаються продажами. Можна взяти всіх співробітників, порахувати їх суму продажів, скільки прибутку вони принесли компанії, та вирахувати, хто виконав зазначений план, хто його перевищив, а хто не зміг його досягти. Такі аспекти справді допоможуть для збільшити мотивацію найкращих співробітників, наприклад, завдяки системі бонусів. Така аналітична діяльність потребує окремої розробки пакету, для управління персоналом, але система за рахунок своєї гнучкості дозволяє це робити без порушення одного з головних принципів об’єктно-орієнтованого проєктування – код повинен бути відкритий для додання, але закритий для змін.

Розробляючи систему треба звернути уваги на гіпотезу, яка була виведена при написані даної роботи, а саме: використання різних інструментів зберігання даних(баз даних) дозволяє покращити роботу і навіть збільшити її функціонал в майбутньому. Повинно бути одне основне сховище, де і будуть знаходить дані. Для кожної угоди будуть створювати документи. Для цього було корисно використати документно-орієнтовану базу даних. Для швидкої роботи та зменшення навантаження можна використати сервер кешування, а для обрки транзакцій що приходять на бухгалтерський облік потрібно реалізувати чергу, як могла деякий час містити в собі дані перед їх безпосередньою обробки. Крім того, ці інструменти було би корисно використати в майбутньому для інших функціональних цілей, тож розгортання даних рішень було би дуже корисно.

2. МОДЕЛЮВАННЯ СИСТЕМИ

2.1 Діаграма прецедентів

Використання засобів UML для опису та представлення предметної області надає змогу розглянути реальний стан галузі, перспективи розвитку та покращення розроблювальної системи, поглибити свої знання в конкретній сфері, візуалізувати взаємозв'язки між об'єктами, представити функції та обов'язки діючих осіб та уже інтегрованих інформаційних систем.

Діаграма прецедентів призначена для візуалізації різноманітних сценаріїв взаємодії між акторами (користувачами) і прецедентами (випадками використання); опису функціональних аспектів системи. Діаграма відіграє важливу роль не тільки у комунікації між збирачами вимог до проекту, а і потенційними користувачами.

У роботі аналітичної системи обліку аграрного підприємства задіяні багато різних осіб, що здійснюють виконання закупок, продажів, облік товарів, здійснення управління безпосередньо системи. Розглянемо діаграму прецедентів для виконавчого провадження на комунальному підприємстві рисунку 1.

Основні актори діаграми прецедентів:

1. Бухгалтер – працівник, що веде бухгалтерський облік підприємства.
2. Директор підприємства - займається переглядом звітом і прийняттям рішень на їх базі.
3. Комірник – працівник підприємства, що веде облік на складі, зберігання, завантаження і вивантаження.
4. Менеджер – займається закупівлями, продажами, звітністю і прийняттям рішень.
5. Аналітик – представник підприємства, що займається веденням статистики та аналізом даних.

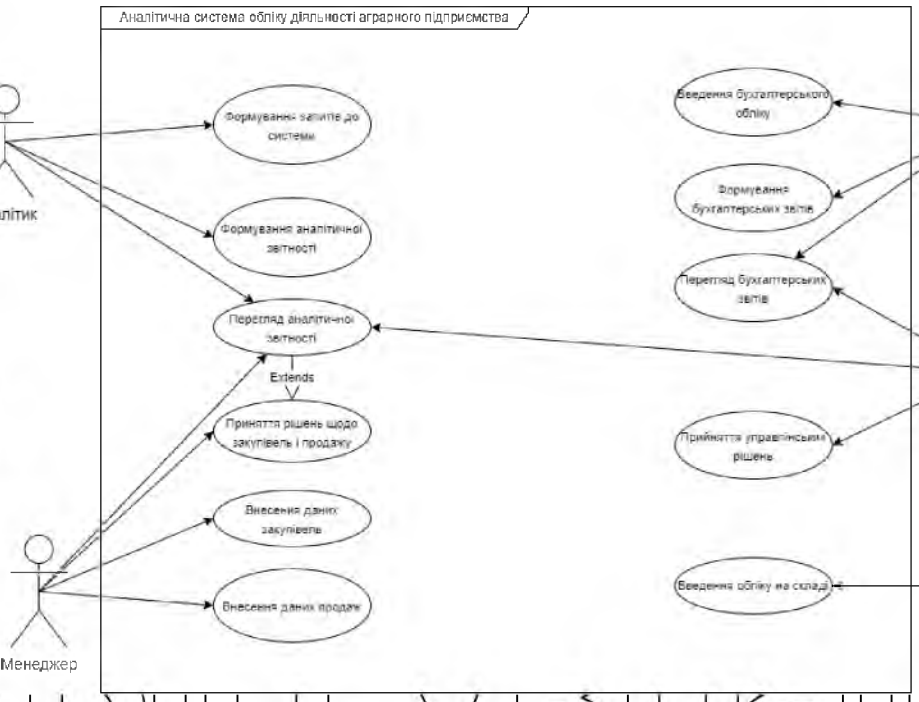


Рис. 1 Діаграмма прецидентів

Розглянемо кілька прецедентів ближче.

Назва прецедента використання: «Введення обліку на складі».

Мета прецеденту використання: Облік всіх операцій на складі, щоб точно знати, що зараз знаходиться на складі.

Оптимістичний сценарій:

А. На склад приходить якийсь товар.

Б. Комірник заносить ці дані в систему.

В. Товар був проданий.

Г. Товар потрібно забрати зі складу

Д. Комірник вносить дані в систему.

Е. Товар вивантажують зі складу.

Прагматичний сценарій:

Умова 1. У товару вийшов термін придатності.

Умова 2. Товару на складі недостатньо для вивантаження.

Умова 3. На складі недостатньо місця.

2.2 Постановка завдання

Система має забезпечувати наступні функції:

1. Формування та перегляд аналітичної звітності
2. Введення бухгалтерського обліку
3. Введення обліку на складі
4. Внесення даних закупів
5. Внесення даних продаж
6. Формування звітів

2.3 Архітектура системи

Система має клієнт-серверну архітектуру, адже це дозволяє забезпечити такі вимоги:

- віддалений доступ;
- забезпечення одночасної роботи великого числа користувачів;
- масштабованість;
- модульне оновлення програмного забезпечення.

На рисунку 2 представлено архітектуру розроблюваної системи, вона містить такі основні вузли:

- Робоча станція аналітика містить модуль, призначений для роботи саме зі сховищем даних.
- Робоча станція бухгалтера, комірника, менеджера.
- Сервер, що містить два основних об'єкта – БД та СД, а також модуль обробки даних, що забезпечує наповнення сховища даних інформацією з оперативної бази даних.

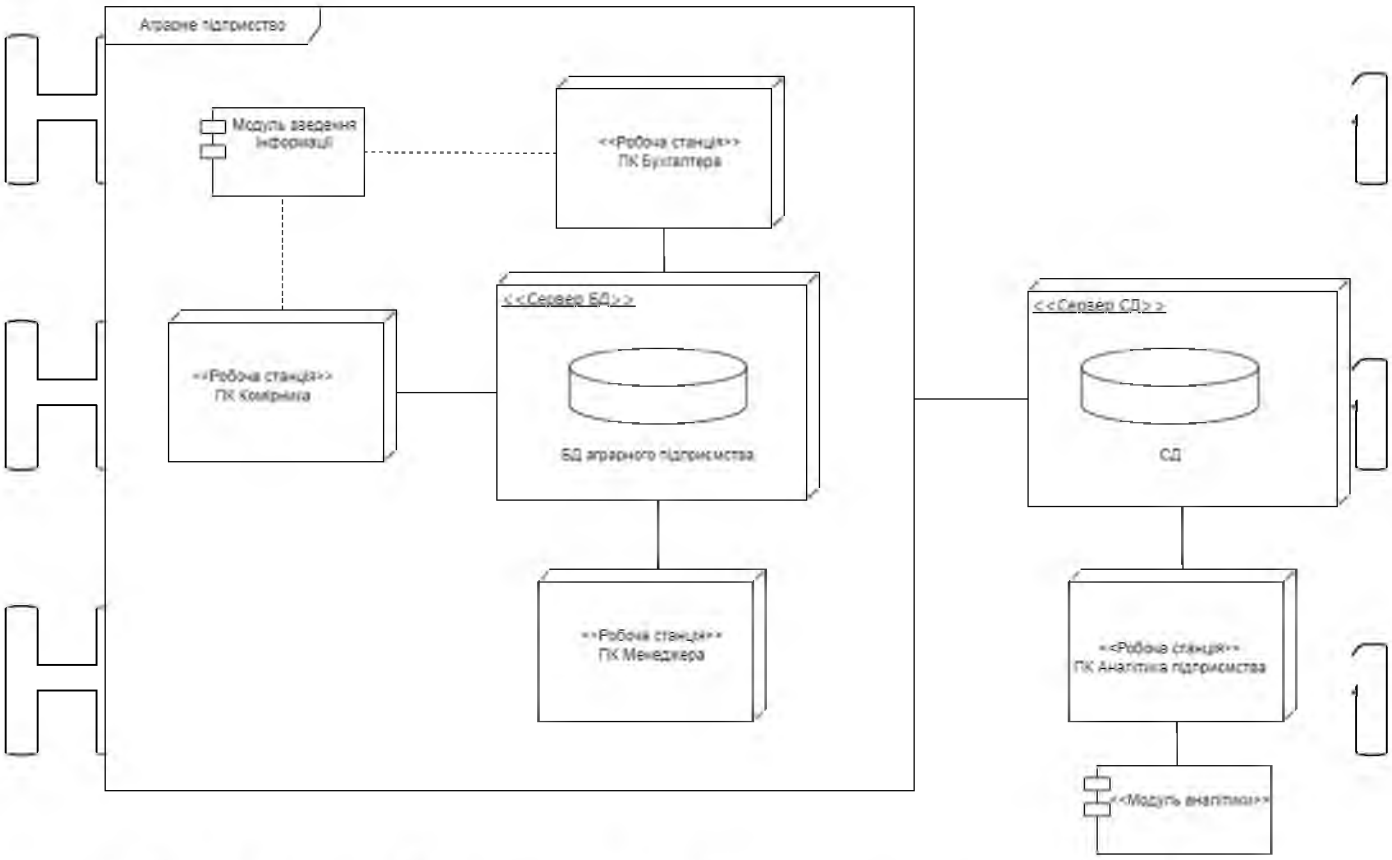


Рис. 2 Топологія системи

2.4 Опис джерела даних

На рисунку 3 представлена схематична структура оперативної бази даних.

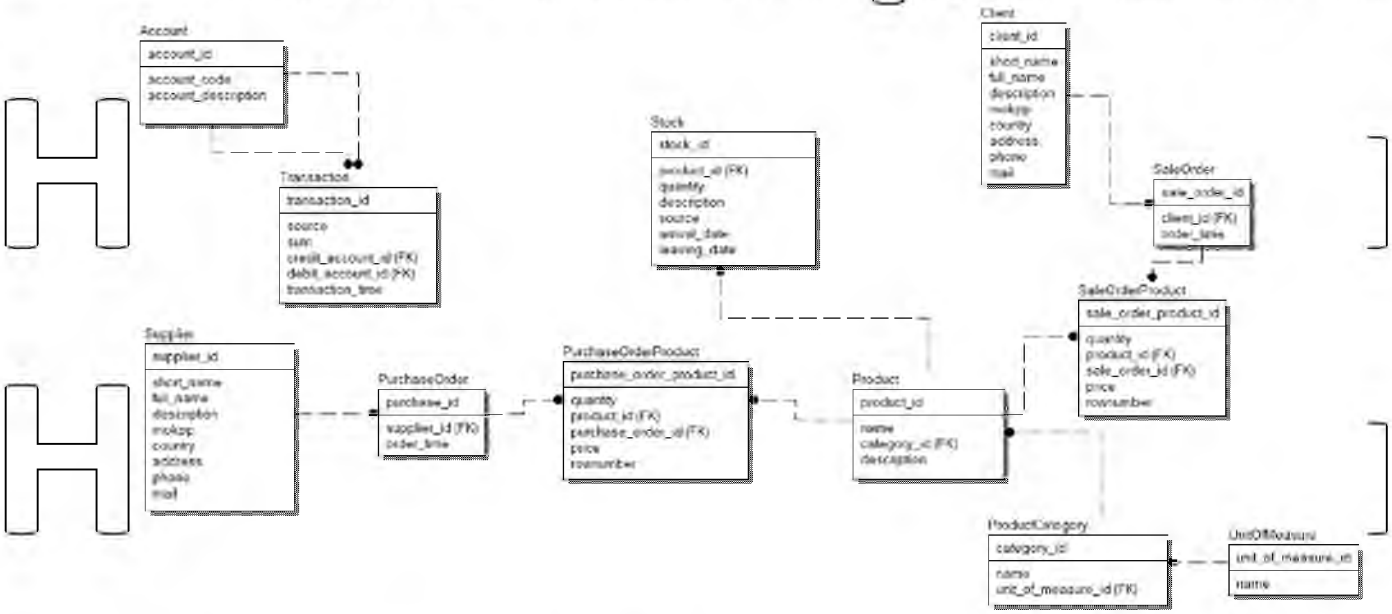


Рис. 3 Оперативна база даних

Опис сутностей наведено нижче

Supplier – сутність містить дані про постачальників:

- Коротке ім’я
- Повне ім’я
- Опис
- Ідентифікаційний код
- Країна
- Адрес
- Телефон
- Пошта

PurchaseOrder – сутність замовлення на постачання:

- Час замовлення

PurchaseOrderProduct – сутність окремого рядка замовлення

- Кількість товару
- Ціна
- Порядковий номер

Product – сутність продукту/товару:

- Назва
- Опис

Client – сутність клієнта:

- Коротке ім’я
- Повне ім’я
- Опис
- Ідентифікаційний код
- Країна
- Адрес
- Телефон

- Пошта

SaleOrder – сутність замовлення на продаж:

- Час замовлення

SaleOrderProduct – сутність окремого рядка замовлення на продаж:

- Кількість товару
- Ціна
- Порядковий номер

Stock – сутність запису на складі:

- Кількість товару
- Опис
- Ресурс
- Час завантаження на склад
- Час вивантаження зі складу

ProductCategory – сутність категорії товару:

- Назва

UnitOfMeasure – сутність одиниць вимірювання товару:

- Назва

Account – сутність бухгалтерського рахунку:

- Код рахунку
- Опис рахунку

Transaction – сутність операції між рахунками:

- Призначення
- Сума
- Кредитний рахунок
- Дебетовий рахунок
- Час операції

3.РОЗРОБКА СИСТЕМИ

3.1 Організаційна структура програмного забезпечення

Комп'ютеризована система фінансового прогнозування складається з надання доступу до джерел даних, концепцій бізнес-планування, обробки запитів і надсилання відповідей, інтерфейсів користувача та концепцій представлення даних в інтерфейсах користувача.

Відповідно до потреб розробленої системи, проста у використанні клієнт-серверна архітектура може розділити систему на два модулі: програма, що відповідає за зберігання на сервері, інтерфейс користувача на стороні клієнта, а оренда даних частина сторони клієнта та сервера. Вона має бути спільною для обох, але основна увага приділяється конфігурації сервера.

Веб-дизайн побудований за моделлю REST. З точки зору дизайну, можна підкреслити, що кожна ідея та представлення даних різні. Отже, є зовнішній і внутрішній сервери.

Важливість моделі REST виникла в 2000-х роках. У той час виникла так звана «проблема Інтернету», коли існуюча модель розробки програмного забезпечення в Інтернеті не відповідала потребам ринку. Системи мають бути розділені, щоб різні розробники могли розвиватися та продовжувати працювати разом. Також необхідно прибрати залежності компонентів програми та слідувати одному з основних принципів об'єктно-орієнтованого програмування. Код має бути відкритий для розширення та закритий для модифікації. Веб-програми обслуговують запити з інших веб-сайтів або програм. Клієнтські програми використовують інтерфейси прикладного програмування (API) для зв'язку з веб-сервером. Загалом, API дозволяють комп'ютерним програмам вибирати дані та служби для зв'язку та обміну інформацією. Веб-інтерфейси API — це «обличчя» Інтернету, яке слухає запити клієнтів і відповідає на них. З REST API ця функція

називається «RESTful». REST API — це набір інтеграційних служб. Список таких речей називається моделлю підтримки.

API REST використовують URI для відповідності доступних даних. Основне правило для створення такого URI можна виразити у вигляді рівняння (1).

URI=схема "://" ресурс "/" шлях ["?" запит] [поле "#"], (1)

Правило генерації ідентифікації продукту можна описати так:

Слеші повинні використовуватися для ідентифікації зв'язків успадкування та не повинні включатися в шлях з будь-якої іншої причини.

Необхідно використовувати пробіли, щоб зробити питання читабельним.

Не використовуйте підкреслення.

Питання повинні бути написані малими літерами.

Файли розширення не слід використовувати для запису.

URI не слід використовувати для вказівки на те, що виконується операція CRUD.

Наразі REST API використовує HTTP/1.1, включаючи методи запитів, коди відповідей і заголовки повідомлень.

Кожен метод застосування має власні символи та унікальну семантику для доступу до моделей ресурсів. Так, ви можете використовувати метод GET для отримання стану ресурсу, HEAD використовується для отримання метаданих функції, а PUT слід використовувати для додавання або оновлення нового ресурсу. DELETE видаляє об'єкт із його батьківського об'єкта. POST використовується для створення нового ресурсу в кластері та ініціалізації контролера. Є правила, яких слід дотримуватися.

GET і POST не слід використовувати для відстеження інших запитів.

Ви повинні використовувати GET, щоб перевірити ресурс.

HEADER загалом схожий на GET, але повинен повертати заголовки відповіді без тіла.

PUT слід використовувати для вставки та зміни архівів.

POST потрібно встановити метод для класу керування.

DELETE видаляє базу даних.

Повертає діючу опцію.

Ще однією особливістю цього шаблону є номер відповіді. Їх багато, але їх можна розділити на п'ять основних груп:

інформація (100-199);

успіх (200-299);

Транспорт (300-399);

Скарги клієнтів (400-499);

Помилка сервера (500-599).

І остання велика частина REST API полягає в тому, що ви не можете використовувати цей шаблон без заголовків. Він може бути довгим, але якщо він занадто короткий, вам знадобляться такі заголовки: Тип елемента, довжина елемента та остання зміна.

Не менш важливою чистиною даного протоколу, є обмеження доступу до даних користувачем. Для цього система використовує авторизацію на рівні веб-сервера. Кожен клієнт виконує авторизацію на сервері. З даними аутентифікації, логіном і паролем клієнтська система надсилає запит на сервер. Якщо дані в запиті правильні, клієнт отримує у відповідь токен сесії і використовує його для подальшого доступу до серверу ресурсу.

Важливо знати, що представлення ресурсів здійснюється за допомогою кількох рішень, переважно у форматі XML та JSON.

Розроблений додаток серверу був створений відповідно до принципів принципів об'єктно-орієнтованого програмування. Сервіс розділений на три основні компоненти: контролер — рівень обробки веб-запитів, сервіс — рівень обробки бізнес-логіки, сховище — рі вень обробки доступу до даних,

включаючи базу даних. Крім основних елементів сервісу, існують також моделі, конфігурації та класи додатків для виконання обчислень.

База даних отримує запити через обробник підключення до бази даних Java. Процедури та тригери були створені тому, що так зручніше виконувати деякі операції на рівні бази даних. Однак ця система не реалізує суто процедурний підхід. Для цього є кілька причин, одна з яких полягає в тому, що код на стороні сервера легше підтримувати, і він не є життєздатним з точки зору архітектури.

З іншого боку, фронт система має лише логіку презентації. Тобто він обробляє правильну сторінку (розділяє набір даних на кілька елементів, сторінок і обробляє відповіді від сервера).

3.2 Вибір інструментарію для створення програмного забезпечення

Одним з найважливіших факторів розробки програмного забезпечення є середовище розробки, яке значною мірою впливає на швидкість розробки. Вибір неправильного набору інструментів розробки та тестування може багаторазово уповільнити процес розробки.

В ході дослідження даної роботи було розроблено зовнішню систему за допомогою WebStorm. Це одне з провідних середовищ розробки з усіма аспектами, необхідними для розробки. Позитивним моментом є те, що ви можете визначити перевірки синтаксису, які не допускають помилок аналізу та компіляції, підтримку інтеграції з системами контролю версій, включаючи Git, інструменти рефакторингу коду та мінімальне тестування. Цього достатньо для поступової розробки, але не для повного тестування. Крім того, ви можете додавати примітки щодо конфігурації до програми, працювати з GraphQL і створювати проекти з шаблонів. Таким чином ви не витрачаєте час на вже виконану роботу щоразу. Також є безкоштовна версія, але можна відзначити

недолік, що програма поширюється тільки за платною підпискою, і її можливостей недостатньо.

Для розробки сервера використовувалося середовище розробки IntelliJ IDEA. Розташування цього середовища розробки є зручним і ергономічним, і з цим важко посперечатися. Основні мови програмування, які використовуються - Java, Scala, Groovy і Kotlin. Корисність цієї програми полягає в її інтелекті, вона може автоматично генерувати код шаблону. Доступний великий набір плагінів. Якщо цього недостатньо, ви можете створити свій власний. За статистикою, 72% розробників Java використовують це середовище розробки. Його використовують такі відомі компанії в галузі інформаційних технологій, як Samsung, Pivotal, Netflix, Twitter, Volkswagen.

Основні причини використання цього середовища розробки:

- Інструмент для складання проекту;
- контроль версій;
- корисний HTTP-клієнт для тестування;
- засоби профілювання;
- декомпілятор коду;
- бази даних та засоби SQL.
- підтримка мов програмування, які працюють у віртуальній машині Java.
- підтримка всіх сучасних фреймворків: Spring, Java EE, Jakarta, Quarkus, Micronaut;
- мінімальна підтримка веб-розробки: JavaScript, HTML, CSS, Angular, React, Vue.
- підтримка мобільної розробки;
- інтеграція з іншими продуктами JetBrains. Це інструменти всіх кольорів і смаків;

- постійна підтримка та впровадження нових технологій.

Для розробки бази даних було отримано інструмент pgAdmin, описаний у пункті 2.2.

Java: для розробки серверів, JavaScript: для веб-розробки, SQL: для розробки баз даних.

На даний момент Java є однією з найпопулярніших мов програмування і не має аналогів у своїй сфері використання. Для розробки серверів у нас є C#, який синтаксично дуже схожий, але все ще далекий від Java за популярністю. Java зазнала великого розвитку, включаючи практично всі банківські системи. Переписувати все іншою мовою програмування принаймні економічно не вигідно. Спільнота Java підтримує і завжди готова прийти на допомогу. Мова C є швидшою, але ці мови мають дуже різні сфери використання та не підтримують Java. як, включаючи об'єктно-орієнтоване програмування.

Maven був створений для створення проекту. Це настільки популярно, що Gradle є альтернативою. Але це унікальна справа смаку. Просто виберіть потрібні залежності в Інтернеті та додайте їх до файлу конфігурації. Отже, вам потрібні Spring, Hibernate, Lombok, Jackson, Hikari, драйвер PostgreSQL, Junit, log4j, Spring Security і Spring Data для розробки вашого проекту, вам слід подумати, як ви їх вибрали.

Spring — це програмна основа, яка реалізує оболонку інверсії залежностей. воно вже мало багато нюансів і максимально ускладнювало розвиток. 2002 і не мав нічого, крім реалізації інверсії залежностей. За свою 18-річну історію кодова база зазнала багато змін, і відмінності між версіями 2015 і 2020 років є разючими. Більшість розробників оцінюють розвиток фреймворку позитивно. Наразі для цього інструменту немає розчисток. Існує досить багато Micronaut з іншим підходом до Spring, який робить це в робочому процесі за допомогою відображення, а Micronaut змінює байт-код у процесі збирання.

Сам по собі Spring має доволі маленький розмір. Його загальна сила полягає в кодовій базі, сформульованій навколо нього, тому доцільно розглянути, що саме було зроблено, щоб зробити цю структуру настільки потужною та впізнаваною. Тобто інтегрувати цю спільноту з іншими фреймворками. Spring Boot нещодавно був створений у процесі розробки фреймворку. Це абстракція, яка додатково спрощує процес розробки програми, надаючи повний набір інструментів із коробки. По-перше, розглянемо Spring Boot Web, який надає зручний інтерфейс для веб-розробки. Під капотом все працює в базовій версії Java у звичайному класі типу HttpServlet. Крім того, існує повна спеціальна обробка сортування запитів і відповідей. Найкраща частина полягає в тому, що він повністю настроюється. Якщо ви хочете самостійно налаштувати обробку, напишіть власну реалізацію та використовуйте її. Це дуже зручно і повністю відповідає принципам парадигми SOLID. Також зауважте, що фреймворк має готову реалізацію сервера Tomcat. Це позбавляє від необхідності встановлювати та налаштовувати окремий веб-сервер. Просто запустіть, і все запрацює.

Іншою частиною більшої інфраструктури Spring Boot є Spring Data. Завдяки цьому ви можете використовувати доступ до даних, як завгодно. Це звичайне виконання запитів безпосередньо до бази даних і реалізація викликів процедур і ORM (Object Relational Mapping). Можна сказати, що все це вже зроблено в Java без використання будь-якого фреймворку, але за допомогою Spring Boot Data ми можемо зробити більше з меншим кодом. те саме свято) і баланс між ними, а також підтримка транзитивності на рівні сервера та обробки помилок бази даних. Останній дійсно потужний. Це пов’язано з тим, що будь-яка помилка створює виняткову ситуацію SQLException, яку неможливо обробити стандартною обробкою помилок. Spring також дозволяє створювати винятки, які можна обробляти у відповідь на помилки, що виникають. Знову ж таки, це досить просто і є одним із основних принципів Spring. Для простоти ви також можете

впровадити базу даних безпосередньо у свою програму. Це не дуже підходить для виробничих середовищ, але це значно спрощує процес розробки.

Jackson — бібліотека для роботи з документами JSON і XML. Кілька років тому підтримка XML не була настільки важливою. Java повністю підтримувала цей процес. Підтримка XML була дуже складною і було багато непотрібних речей, але вона працювала і працювала дуже добре. Але починаючи з Java 11 він перестав працювати і був просто видалений. Насправді це дуже неприємний елемент. Нам довелося змінити існуючий код, оскільки перехід нашого сервісу на нову версію мови програмування був поганим, але ми нічого не могли з цим вдіяти. Джексон зробив налаштування кращим, швидшим, зрозумілішим і зручнішим. Але у мене не було обробки JSON, тому цей фреймворк мені дуже допоміг.

Lombok — це бібліотека, яка спрощує розробку, генеруючи конструктори, методи перевірки ідентичності об'єкта, хеш-функції, геттери та сетери. І тут все не так зрозуміло. З одного боку, це спрощує роботу, робить код чистішим, усуває шаблонний код. Є проблема.

Log4j — це бібліотека, яка дозволяє маніпулювати журналами обслуговування. Показано різні інтерфейси для реєстрації як простих файлів, так і більш складних файлів, таких як стеки ELK. Загалом, можна зробити перший і другий тип колод більш надійними. Ви також можете налаштувати шаблони журналів і фільтри. Наприклад, якщо ви хочете зберігати лише помилки та вказати загальний час запису в наносекундах. Elastic — це пошукова система, орієнтована на JSON, яка дозволяє виконувати пошук. Logstash — це так зване сховище даних, де журнали зберігаються та індексуються.

Junit — це платформа автоматизації тестування з усіма інструментами, необхідними для тестування.

Решта — це драйвер PostgreSQL і Hikari. Обидва інструменти необхідні для роботи з базою даних. Перший — драйвер бази даних PostgreSQL. Hikari — дуже легкий, але потужний інструмент. краще за інших.

Виконання коду на рівні байт-коду.

Мікрооптимізації, які зазвичай прискорюють виконання.

Розумне використання колекцій.

Тому набір технологій, що використовуються для розробки серверів, може задовольнити будь-які потреби бізнесу, які можуть виникнути в процесі розробки додатків.

Веб-розробка видалила технології JavaScript, HTML, CSS, Vue.js і Axios.

Для веб-розробки було обрано архітектурне рішення під назвою SPA (Single Page Application), яке має багато переваг перед MPA.

Для швидкості потрібно отримати менше даних, сторінка завантажується лише один раз, а на сервер надсилаються лише асинхронні запити.

SPA можна легко налагодити за допомогою браузера Chrome, який надає повний набір інструментів.

Це спрощує процес розробки для мобільних пристроїв, оскільки логіка сервера відокремлена від логіки презентації та може повторно використовуватися в інших системах.

Це спрощує сам процес розробки.

SPA можуть ефективно зберігати локальні кеші, а деякі функції програми працюють без доступу до Інтернету.

Також зауважте, що цей підхід має деякі недоліки, про які слід знати.

Просунути свій сайт в пошуку складніше.

Перший дзвінок може бути дуже довгим.

Потрібна підтримка браузера JavaScript.

Односторінкові програми вважаються менш безпечними, оскільки вони можуть запускати сценарії, які можуть отримувати дані з сервера, але вони покладаються на розробника для забезпечення безпеки сайту.

JavaScript має певні проблеми з пам’яттю, тому вам може знадобитися більше оперативної пам’яті та потужності ЦП на вашій клієнтській машині.

Vue.js – це засіб для розробки SPA додатків, молодий фреймворк, один з для веб-розробки, наймолодший серед популярних інструментів розробки, який стрімко набирає популярність. Основними його супротивниками, альтернативними рішення є Angular і React. Angular – це величезна екосистема для розробки фронт систем, інструмент, який використовується для розробки великих систем, він має все, або по крайній мірі все. Порівнюючи з іншими інструменти, можна сказати, що його популярність також зумовлена підтримкою IT-гігантів, таких як Microsoft, а так окремої мови програмування TypeScript, яка додає у фронт-розробку принципи ООП та дозволяє бути коду більш структурним. React – суб’єктивно є не таким інтуїтивно зрозумілим і в багатьох сучасних проектах надають перевагу Vue замість нього. Основна перевага фреймворка – він швидкий і зручний у розробці, дуже легкий. Можна сказати, що він проаналізував помилки своїх конкурентів і зробив з цими висновками і насправді це прекрасно.

Charts.js — це бібліотека візуалізації даних JavaScript. З кожним роком світ комп'ютерних технологій все більше заповнює глобальну мережу. Мова розмітки HTML5 захоплює світ штурмом, і наразі немає жодних ознак припинення її розвитку та впровадження. Ключові переваги:

- Графіка рендериться на стороні клієнту. Чому? Тому що його можна змінити на клієнті;
- Створити діаграму легко. Для створення діаграми не потрібно велика кількість коду;

- Працює у всіх браузерах. Також сумісний з мобільними пристроями;
- Також можна додати, що ця бібліотека - це не тільки графіка, а й багато інших елементів, значення яких можна оцінити візуально.

Axios наразі є найпопулярнішою бібліотекою для надсилання HTTP-запитів. Підтримуються всі методи запиту, заголовки, в тому числі авторизації, та обробка відповідей сервера. Реалізація бібліотеки є абстракцією стандартних об'єктів JavaScript, інструменти Socket та HttpXmlRequest, але, як і більшість бібліотек, точніше як їх основне завдання, вона максимально спрощує доступ до даних. Крім того, ви можете визначити свої запити AJAX у бібліотеці jQuery, але це гірший варіант з двох причин. По-перше, jQuery давно не використовувався в реальних проектах, використання цієї бібліотеки може бути зумовлено тільки якоюсь ненавистю до сучасних інструментів, або якщо проєкт був написати доволі давно, якщо йому більше 5 років, коли ще можна було зрозуміти використання даного інструменту і використання бібліотеки для однієї функції не є розумним варіантом.

3.3 Алгоритмізація та програмування програмних модулів

У цій системі вся обробка даних виконується на стороні сервера. Якщо ми говоримо про фронтенд системи, то тут немає іншої логіки, крім логіки подання. Логіку сервера можна розділити на дві логічні частини: логіку пошуку даних і логіку обробки даних.

Спочатку розглянемо як запит обробляється на стороні клієнта. По-перше, клієнт може лише перейти до сторінки підтвердження особистої, немає можливості перейти до іншої сторінки. Ви не зможете отримати доступ до своїх даних, доки не завершиться автентифікація. Після того, як клієнт вводить дані (наприклад, ім'я користувача та пароль), на сервер надсилається запит на автентифікацію, і якщо користувач ввів усе правильно, сервер повертає маркер доступу та ролі, надані цьому користувачеві. Залежно від своєї ролі клієнт

бачитиме різні сторінки: Користувач, Сторінка прогнозів, Адміністратор, Сторінка адміністратора. Маркер запиту також використовується клієнтом для надсилання запитів на сервер. Вам не потрібно вводити дані автентифікації або отримувати контекстні відповіді. Методи, реалізовані на рівні сервера, описані нижче. Тому для інтерфейсних систем краще починати обговорення з боку сервера. Це тому, що це лише відповідь у формі документа, як показано користувачам у шаблоні.

Звернувши концепцію отримання даних в системі, важливо почати з рядка підключення JDBC і пояснити, що потрібно зробити для отримання цих даних. Насправді всі бази даних різні. В нашому випадку база даних PostgreSQL такий вид рядка для підключення через JDBC драйвер(2).

jdbc:postgresql://host:post/database?properties,(2)

Крім того, необхідно вказати ім'я користувача та пароль користувача бази даних. Крім того, він показує максимальну кількість підключень, мінімальну кількість підключень, тестові запити на підключення, автоматичне підтвердження кожного запиту, максимальний час підключення до бази даних і максимальний час обробки запиту. Для зберігання всіх цих констант було створено окремий файл. Цей файл можна оновити без перекомпіляції коду. Надалі ви можете продовжувати використовувати сервер конфігурації. Це дозволяє зберігати ваші налаштування в одному місці та запобігати їх використанню іншими користувачами.

Розглянемо принцип підключення даних через JDBC більш детально. Java Database Connectivity API надає універсальний доступ до даних з мови програмування Java. Використовуючи даного функціонального інтерфейсу, дозволяє вам віртуально отримати доступ до будь-якої бази даних, починаючи з реляційних бази даних, закінчуючи електронних таблиць та просто файлів. Технологія також надає стандарт для кожного інструменту і може бути розроблений альтернативний інтерфейс. Тобто, підсумовуючи, ми маємо

інструмент для доступу до бази даних, який по суті просто інтерфейс, який можна реалізувати як завгодно, якщо наприклад поточна реалізація не подобається, або якщо хтось розробив свою базу даних і захотів створити драйвер для Java може використати цей інструмент.

Розгляд файлу конфігурації показано на рис 3.3.1:

```
spring.datasource.url=jdbc:postgresql://localhost:5432/postgres
spring.datasource.hikari.username=postgres
spring.datasource.hikari.password=admin
spring.datasource.hikari.maximum-pool-size=10
spring.datasource.hikari.minimum-idle=2
spring.datasource.hikari.connection-test-query=select 1 from dual
spring.datasource.hikari.auto-commit=true
spring.datasource.hikari.connection-timeout=30000
spring.datasource.hikari.idle-timeout=15000
```

Рис.3.3.1 Файл конфігурації

Для використання доступу до бази даних через засоби Spring можна використати JdbcTemplate. Є і інші інструмент, наприклад різні ORM, але для вирішення поточних задач, що виконуються в даному дослідження буде достатньо і цього. Реалізацію можна побачити на рис 3.3.2:

```
@Configuration
public class AppConfig {

    @Autowired
    private Environment env;

    @Bean
    public JdbcTemplate jdbcTemplate() {
        HikariConfig configuration = new HikariConfig();
        configuration.setJdbcUrl(env.getProperty("spring.datasource.url"));
        configuration.setUsername(env.getProperty("spring.datasource.hikari.username"));
        configuration.setPassword(env.getProperty("spring.datasource.hikari.password"));
        configuration.setMaximumPoolSize(Integer.parseInt(Objects.requireNonNull(env.getProperty("spring.datasource.hikari.maximum-pool-size"))));
        configuration.setMinimumIdle(Integer.parseInt(Objects.requireNonNull(env.getProperty("spring.datasource.hikari.minimum-idle"))));
        configuration.setConnectionTestQuery(env.getProperty("spring.datasource.hikari.connection-test-query"));
        configuration.setAutoCommit(Boolean.parseBoolean(env.getProperty("spring.datasource.hikari.auto-commit")));
        configuration.setConnectionTimeout(Long.parseLong(Objects.requireNonNull(env.getProperty("spring.datasource.hikari.connection-timeout"))));
        configuration.setIdleTimeout(Long.parseLong(Objects.requireNonNull(env.getProperty("spring.datasource.hikari.idle-timeout"))));
        HikariDataSource dataSource = new HikariDataSource(configuration);
        return new JdbcTemplate(dataSource);
    }
}
```

Рис. 3.3.2 Реалізація використання Java для підключення до бази даних через Jdbc Template

3.4 Вибір баз даних для тестування

Загалом, для виконання даного проєкту, як було зазначено раніше, буде використано різні типи реляційних і нереляційних баз даних. Сама по собі база даних говорить сама за себе – її рол зберігати даних. В найпримітивнішому виді, така базою даних може бути навіть звичайних файл, наприклад тексту. Але слід наголосити, що крім цього, важливим чинником для бази даних є, що це не просто зберігання даних, а саме структуроване зберігання даних. На основі даного принципу було сформовано гіпотези даної роботи – використання такого підходу в теорії повинно збільшити функціональний потенціал системи, а також швидкість обробки запитів, що в свою чергу збільшує швидкість роботи працівників підприємства. Тож, розглянемо використані інструменти більш детально.

Для початку ознайомимось з, так званою, основною базою даних, де зберігається найбільша кількість даних, а саме реляційною базою даних PostgreSQL. Основна відмінність реляційних та нереляційних баз даних в тому, що реляційні дані мають чітку структуру, яка формується на основі строгих правил, серед яких те, що в базі даних є сутності, які мають атрибути, серед яких я первинні ключі та зовнішні ключі. Ці сутності можуть бути пов’язані між собою зв’язками, які бувають трьох типів: один до одного, один до багатьох та багато бо багатьох. Такий підхід є досить зручним для добре структурованої системи, яка одночасно дозволяє користуватися неймовірним функціоналом, а також дає логічний опис, самої системи, її структури, роду діяльності. Така база даних при правильній структуризації, може дуже швидко бути розширеною, хоча й на перший погляд складніше, чим бази даних, що будуть розглянуті нижче. Таке відчуття може виникати насамперед із-за строгої типізації, яка в свою чергу навпаки спрошує розробки, особливо, коли систему потрібно масштабувати. Тож, повернемось до одного з найпопулярніших представників бази даних. PostgreSQL називає себе найдосконалішою у світі базою даних з відкритим

кодом. PostgreSQL — це надзвичайно потужне програмне забезпечення, яке пропонує функції, яких ви, можливо, не бачили раніше. Деякі функції також прикутні в інших добре відомих механізмах баз даних, але під іншими назвами. PostgreSQL — це система керування реляційними базами даних корпоративного класу, яка не поступається найкращим пропрієтарним системам баз даних: Oracle, Microsoft SQL Server і IBM DB2. PostgreSQL особливий тим, що це не просто база даних: це ще й платформа додатків, до того ж вражаюча. PostgreSQL швидкий. У контрольних тестах PostgreSQL або перевершує, або відповідає продуктивності багатьох інших баз даних, як відкритих, так і пропрієтарних. PostgreSQL запрошує вас писати збережені процедури та функції численними мовами програмування. На додаток до готових мов C, SQL і PL/pgSQL, ви можете легко ввімкнути підтримку додаткових мов, таких як RL/Perl, PL/Python, PL/V8 (така ж PL/JavaScript), PL/Ruby та PL /P. Ця підтримка широкого спектру мов дає змогу вибрати потрібну мову з конструкціями, які можуть найкращим чином вирішити поставлену проблему. Наприклад, використовуйте R для статистики та побудови графіків, Python для виклику веб-сервісів, бібліотеку Python SciPy для наукових обчислень і PL/V8 для перевірки даних, обробки рядків і боротьби з даними JSON. Ще простіше: знайдіть потрібну вам функцію у вільному доступі, знайдіть мову, якою вона написана, увімкніть цю конкретну мову в PostgreSQL і скопіюйте код. Ніхто не думатиме про вас менше.

Більшість продуктів баз даних обмежують вас попередньо визначеним набором типів даних: цілі числа, тексти, логічні значення тощо. PostgreSQL не лише має більший вбудований набір, ніж більшість, але ви можете визначити додаткові типи даних відповідно до ваших потреб. Потрібні комплексні числа? Створіть складений тип із двох плаваючих елементів. Маєте бажання створити трикутник? Створіть тип координат, а потім створіть тип трикутника, що складається з трьох пар координат. Тип з дванадцяти чисел? Створіть свій власний дванадцятковий тип. Інноваційні типи корисні настільки, наскільки

оператори та функції їх підтримують. Отже, коли ви створили спеціальні типи чисел, не забудьте визначити для них основні арифметичні операції. Так, PostgreSQL дозволить вам налаштувати значення символів (+,-,/,*). Щоразу, коли ви створюєте тип, PostgreSQL автоматично створює допоміжний тип масиву для вас. Якщо ви створили тип комплексного числа, масиви комплексних чисел доступні вам без додаткової роботи.

PostgreSQL також автоматично створює попередньо визначені таблиці. Наприклад, створіть таблицю собак зі стовпцями породи, миловидності тощо. За лаштунками PostgreSQL керує типом даних dog. Цей надзвичайно корисний міст між анімованими світами та об'єктами. Світ означає, що елементи даних можна обробляти зручним способом Що робити далі Ви можете створювати функції, які працюють з одним об'єктом за раз. Функції, що діють одночасно з набором об'єктів Для PostgreSQL використовуйте спеціальні типи для підвищення продуктивності. Доменно-спеціальні конструкції для коротшого коду, який зручніше підтримувати. Функції, які стають дедалі важливіщими.

Все ж таки, якби добре все не було, для об'єктивної оцінки, яку потребує будь-яка поважаюча себе наукова робота. Потрібно визначити і недоліки PostgreSQL. Тож давайте розглянемо, чому ж не PostgreSQL? Типовий розмір встановлення для PostgreSQL без розширень більше 100 МБ або більше. Це не включає використання PostgreSQL для баз даних на невеликих пристроях. Як просте кеш-сховище. Існує багато легких баз даних. Задовольняйте свої потреби, не розширюючи площу. Враховуючи статус компанії, PostgreSQL не економить на безпеці. Якщо я розробляю легку програму, яка керує безпекою на прикладному рівні безпека PostgreSQL і керування дозволами може бути надмірним. Якщо ви розглядаєте одного користувача, ви можете запускати бази даних, такі як SQLite, або бази даних, такі як Firebird. як клієнт-сервер або в однокористувацькому вбудованому режимі. Однак зазвичай PostgreSQL поєднують з іншими типами баз даних. Однією з поширених комбінацій є

використання Redis або Memcache для кешування результатів його запитів PostgreSQL. Як інший приклад, якщо PostgreSQL є основною базою даних вашої програми, ви можете використовувати SQLite для зберігання відключених наборів даних для офлайн-запитів.

Далі буде розглянуто нереляційні бази, які було використано в процесі розробки. І почати потрібно з бази даних MongoDB - це потужна, гнучка та масштабована база даних загального призначення. Він поєднує можливість масштабування з такими функціями, як вторинні індекси, запити діапазонів, сортування, агрегації та геопросторові індекси. MongoDB — це документно-орієнтована база даних, а не реляційна база даних. Основною причиною відмови від реляційної моделі є полегшення масштабування, але є й інші переваги. Документно-орієнтовані бази даних замінюють концепцію «рядків» більш гнучкою моделлю «документи». Дозволяючи вбудовані документи та масиви, документоорієнтований підхід дозволяє одному запису представляти складні ієрархічні зв'язки. Це природно вписується в те, як сучасні розробники об'єктно-орієнтованої мови думають про дані. Також не визначено жодної схеми. Ключі та значення документа не фіксовані, як тип і розмір. Без фіксованої схеми легше додавати або видаляти поля за потреби. Загалом це пришвидшує розробку, дозволяючи розробникам швидко виконувати ітерації. Експериментувати легко. Розробники можуть випробувати десятки моделей зі своїми даними, перш ніж вибрати найкращу. Розміри наборів даних додатків зростають із загрозливою швидкістю. Підвищена доступність, пропускна здатність і дешеве сховище дозволяють використовувати невеликі програми-репліки, що повинні зберігати більше даних, ніж багато баз даних призначені для обробки. Терабайт колись безпрецедентна кількість даних тепер є нормою. Неймовірна продуктивність є головною метою MongoDB і значною мірою сформувала його дизайн. MongoDB додає динамічні доповнення до документів і попередньо виділяє файли даних для додаткових транзакцій. Використовуйте простір для стабільної продуктивності.

Використовуйте якомога більше оперативної пам'яті як кеш, який намагається автоматично вибрати правильний індекс для запиту. Усі аспекти MongoDB розроблені для підтримки високої продуктивності. MongoDB призначена як база даних загального призначення, тому окрім створення, читання, оновлення та видалення даних, він надає список унікальних функцій, що постійно зростає, серед яких:

- Індексація - MongoDB підтримує загальні вторинні індекси, дозволяючи різноманітні швидкі запити і надає унікальні можливості складеного, геопросторового та повнотекстового індексування.
- Агрегація - MongoDB підтримує «конвеєр агрегації», який дозволяє створювати комплекс агрегації з простих частин і дозволяє базі даних оптимізувати її.
- Спеціальні типи колекцій - MongoDB підтримує колекції даних, термін дії яких закінчується в певний час, наприклад, сесії. Він також підтримує колекції фіксованого розміру, які корисні для зберігання кінцевих даних, наприклад журналів.
- Файлове сховище – MongoDB підтримує простий у використані протокол, що дозволяє зберігати величезні файли та файли метаданих.

Тож підсумовуючи, можна сказати – що ця база добре підходить для невеликих проєктів, який не обов’язково мати строгу типізацію даних і структуру самої бази даних. Це в деяких випадках пришвидшує розробку і дозволяє виконувати запити дуже швидко. Це є і плюсом і мінусом, так як коли маєш велику базу зі складними бізнес процесами, то хочеться повністю контролювати те, що відбується в системі.

Наступним кроком буде розглянуто базу даних, які використовуються більше не для постійно зберігання даних, а як оптимізація роботи самої системи.

І одна із-за таких баз даних Apache Kafka. Зараз самий розквіт використання даної бази даних. Kafka використовується десятками тисяч організацій, у тому числі понад третиною компаній зі списку Fortune500(список найбільших IT-гігантів світу). Це один з найбільш швидкозростаючих проектів з відкритим кодом, який створив величезну екосистему навколо нього. Це в основі руху до керування та обробки потоків даних. Kafka розпочався як внутрішня інфраструктурна система, яка була побудована в LinkedIn. Ціль була дуже проста: було багато баз даних та інших систем, створених для зберігання даних, але в нашій архітектурі бракувало чогось, що могло б допомогти щоб обробляти безперервний потік даних. Перш ніж створити Kafka, проводились експерименти з усіма готовими варіантами; від систем обміну повідомленнями до агрегації журналів і інструментів ETL, але жоден із них не дав того, що було потрібно.

Kafka схожа на систему обміну повідомленнями, оскільки ви можете публікувати та підписуватися на потоки повідомлень. Таким чином, вона схожа на такі продукти, як ActiveMQ, RabbitMQ і IBM MQSeries. Але навіть при цій схожості Кафка має деякі особливості. Це зовсім інша тварина, тому що вона радикально відрізняється від традиційних систем обміну повідомленнями. Тут є три великі відмінності. По-перше, вона працює як кластер і служить сучасною розподіленою системою, яка може масштабуватися для обробки всіх програм для найбільших підприємств. Це дозволяє мати центральну платформу, яку можна пружно масштабувати для обробки всіх потоків даних у вашому підприємстві, а не запускати десятки окремих брокерів обміну повідомленнями, підключених вручну до різних програм. По-друге, Kafka — це справжня система зберігання, створена для зберігання скільки завгодно даних. Це велика перевага для використання як рівня підключення, оскільки це забезпечує реальні гарантії доставки. Дані тиражуються, зберігаються та можуть зберігатися скільки завгодно довго. Нарешті, світ обробки потоків пропонує набагато вищий рівень абстракції. Системи обміну повідомленнями здебільшого просто доставляють

повідомлення. Можливості потокової обробки Kafka дозволяють динамічно обчислювати потоки та набори даних, отримані з потоків, із значно меншим кодом. Ці відмінності роблять Кафку настільки унікальним, що насправді немає сенсу думати про нього як про «ще одну чергу».

Інший погляд на Kafka полягав у тому, щоб думати про нього як про версію Hadoop у реальному часі. Hadoop дозволяє зберігати та періодично обробляти дані файлів у дуже великому масштабі. Kafka дозволяє зберігати та безперервно обробляти потоки даних, навіть у великих масштабах. На технічному рівні, безперечно, є схожість, і багато людей бачать, що нова область обробки потоків є надмножиною того типу пакетної обробки, яку люди робили з Hadoop та його різними рівнями обробки. Це порівняння втрачає те, що випадки використання, які відкриває безперервна обробка з низькою затримкою, суттєво відрізняються від тих, які природно трапляються в системі пакетної обробки. У той час як Hadoop і великі дані орієнтовані на аналітичні додатки, часто в просторі сховищ даних, низька затримка Kafka робить її придатною для тих основних програм, які безпосередньо живлять бізнес. Це має сенс: події в бізнесі відбуваються постійно, і здатність реагувати на них, коли вони відбуваються, значно полегшує створення послуг, які безпосередньо забезпечують роботу бізнесу, повертаються до досвіду клієнтів і так далі.

Остання сфера, з якою порівнюють Kafka, це ETL(Extract, Transform, Load) або інструменти інтеграції даних. Зрештою, ці інструменти переміщують дані, а Кафка переміщує дані. У цьому також є певна достовірність, але я думаю, що основна відмінність полягає в тому, що Кафка перевернув проблему. Замість інструменту для вилучення даних з однієї системи та вставлення їх в іншу, Kafka — це платформа, орієнтована на потоки подій у реальному часі. Це означає, що він може не тільки підключати готові додатки та системи даних, але й запускати власні додатки, створені для запуску тих самих потоків даних. Тож, що ця архітектура, зосереджена навколо потоків подій, є справді важливою річчю.

Певним чином ці потоки даних є найважливішим аспектом сучасної цифрової компанії, таким же важливим, як і грошові потоки, які ви бачите у фінансовому звіті.

Можливість поєднати ці три сфери — об'єднати всі потоки даних у всіх варіантах використання — ось що робить ідею потокової платформи такою привабливою для людей.

І остання база даних, яка буде розглянута, це теж не реляційна база даних, яка використовується в більшості для кешування даних Redis - це сервер структури даних із набором даних у пам'яті для збільшення швидкості. Його називають сервером структур даних, а не просто сховищем значень ключів, оскільки Redis реалізує структури даних, які дозволяють ключам містити двійкові безпечні рядки, хеші, набори та відсортовані набори, а також списки. Це поєднання гнучкості та швидкості робить Redis ідеальним інструментом для багатьох програм. Сьогодні Redis використовується великими та малими компаніями, які виконують як великі, так і малі завдання. Такі компанії, як Engine Yard, Github, Craigslist, Disqus, Digg і Blizzard, є частиною зростаючого списку користувачів Redis. Хоча дану структуру даних важко назвати традиційною базою даних, але насправді вона маю свою структуру і зберігає дані, то фактично – це і є база даних.

Серед задач, що може виконати Redis – це кешування даних. Розглянемо на прикладі. У компанії є якийсь сайт, який працює з великою кількістю запитів, запити працюють REST API, які можуть перевантажувати систему, більш запитів є типовими і користувачу не потрібно отримувати саме ті, що знаходяться в базі даних саме в цю секунду - дані які були пів хвилини назад в базі даних задовольняють потреби користувача. Для цього і можна використати Redis – він буде в собі зберігати дані в структурі ключ-значення і повертати саме те, що потрібно користувачу і навантаження буде набагато меншим. Сучасну систему важко представити без кешування даних, окрім випадків, коли актуальна

інформація потрібна в цей же момент. Серед таким систем може бути, наприклад, процесінговий центр в банківській сфері, у якій все зав'язано на актуальність інформації, але для безперебійною роботи таких сервісів використовуються зовсім інші підходи моделювання архітектури системи.

3.5 Вибір інструментів тестування

Для повноцінного дослідження потрібні інструменти, що не тільки зможуть підтвердити правильність роботи системи, а й інструменти, що зможуть визначити час обробки даних, який так важливий у дослідженні, яке передбачає визначити чи є взагалі сенс використовувати набір баз даних для аналітичної системи обліку аграрного підприємства чи можливо варто залишити все як є. Серед таких інструментів, нам знадобиться два схожих за функціоналом інструменти, але різним за цілю, для яких їх використовують: Postman та Apache JMeter. Далі буде розглянуто їх окремо та порівняно в кінці.

Postman, що він з себе представляє? Це інструмент для роботи з API, який дозволяє тестувальнику відправляти запити до сервіс і працювати з їх відповідями. З його допомогою можна протестувати бекенд та впевнитись, що все працює коректно. Інструментарію зі схожим функціонал існує доволі багато. Вибір на Postman пав із-за того, що він є найпопулярнішим і крім того у нього є інші плюси, серед яких:

- Інтуїтивно-зрозумілий і простий у користуванні, не потребує якогось складного налаштування чи знання мов програмування;
- Безкоштовний, окрім окремих функцій, для більшості задач буде досить його безкоштовної версії;
- Підтримую різні види API (REST, SOAP, GraphQL);
- Розширюється під будь-які потреби за допомогою Postman API;
- Легко інтегрується в CI/CD за допомогою Newman – консольна програма для запуску тестів;

- Запускається на будь-якій ОС;
- Підтримує ручне та автоматичне тестування;
- Зібрав навколо себе велику спільноту, де можна знайти відповідь на будь-яке питання.

Тестувальнику цей інструмент дозволяє:

- Відправляти запити і отримувати відповіді і отримувати відповіді;
- Зберігати запити в папки і колекції;
- Змінювати параметри запитів;
- Змінювати середовище(dev, test, production);
- Виконувати автотести, використовуючи Collections runner, в тому числі по графіку;
- Імпортувати або експортувати колекції запитів і наборів тестів, щоб обмінюватись даними з колегами.

Далі розгляпемо Apache JMeter, який схожий за функціоналом до Postman, але має зовсім інші цілі. Apache JMeter – це програма з відкритим кодом, яка на 100% є чистою Java, яка була розроблена для завантаження тестів функціональної поведінки, як і Postman, і вимірювання продуктивності. З самого початку була створена для тестування Веб-додатків, але зараз може бути використана для інші тестових функцій.

Apache JMeter можна використовувати для тестування продуктивності як на статичних, так і на динамічних ресурсах, динамічних веб-додатках. Його можна використовувати для імітації великого навантаження на сервер, групу серверів, мережу чи об’єкт, щоб перевірити його міцність або проаналізувати загальну продуктивність за різних типів навантаження. Функції JMeter включають:

- Можливість завантажувати та тестувати продуктивність багатьох різних типів програм/серверів/протоколів:
  - Веб – HTTP, HTTPS(Java, NodeJS, Golang, …),

- FTP;
- Бази даних через JDBC;
- LDAP;
- Проміжне ПЗ, що орієнтоване на повідомлення(JMS);
- Поштові протоколи – SMTP, POP3 і IMAP;
- Нативні команди і скрипти;
- TCP;
- Java об’єкти.

- Повнофункціональна тестова IDE, яка дозволяє швидко записувати план тестування (з браузерів або нативних програм), створювати та налагоджувати;
- Режим CLI (режим командного рядка для завантаження тесту з будь-якої ОС, сумісної з Java;
- Повний і готовий до представлення динамічний HTML-звіт;
- Легка кореляція завдяки можливості отримувати дані з найбільш популярних форматів відповідей, HTML, JSON, XML або будь-якого текстового формату;
- Повна багатопотокова структура дозволяє одночасну вибірку багатьма потоками та одночасну вибірку різних функцій окремими групами потоків;
- Кешування та автономний аналіз/відтворення результатів тесту.

Підсумовуючи все вище описано, можна сказати, що функції і правда дуже схожі з Postman, але першого ціль протестувати правильності роботи сервісу, а JMeter в першу чергу – це робота з великою кількістю потоків задля того, щоб визначити продуктивність система, яка можна залежати від багатьох факторів, наприклад: потужність заліза, ефективність написаного коду, обмеження самої мови програмування, швидкість інтернету(у випадка Web тестування).

4. РЕЗУЛЬТАТИ ДОСЛІДЖЕННЯ

4.1 Результати розробки системи обліку

4.2 Результати використання MongoDB

Для отримання результату дослідження, почнемо зі створення потрібних нам елементів. На рисунку 4.2.1 можна побачити створену базу даних і колекцію в середовищі MongoDB. Для розгортання сервісу було використано хмарне середовище Atlas, в якому і було створено базу даних:

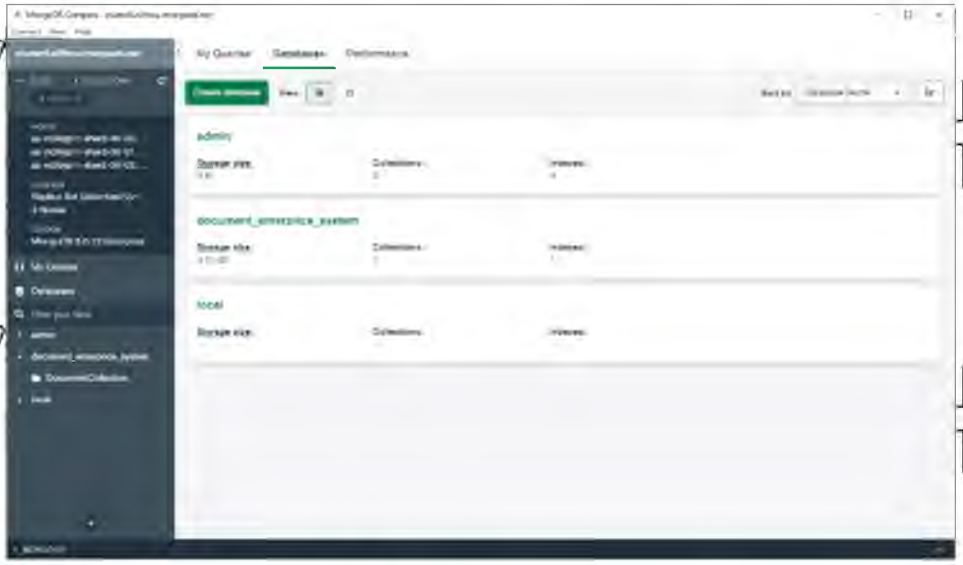


Рис. 4.2.1 Створення бази даних і колекції в MongoDB

Після створення бази даних, наповнимо її даними. В нашому випадку, буде використано JSON-документ, який своїй структурі має 2 поля (якщо не брати до уваги _id, який є обов'язковим): ідентифікатор документи, а сам документ, який конвертований у Base64. Ідентифікатор документа, це по суті ідентифікатор покупки або продажі в базі даних, звідти він і береться і після цього система відправляє запит в MongoDB. На малюнку 4.2.2 можна почати базу даних, яка наповнена даними:

Рис. 4.2.2 Колекція, яка наповнена даними

Після створення бази даних, потрібно розробити API для з базою даних. Після того, як це було зроблено, перевіряємо запит у середовищі тестування Postman, як це показано на рисунку 4.2.3

Рис. 4.2.3 Тестування в середовищі Postman

Упевнившись, що все працює коректно, починаємо будувати запит в середовищі JMeter, як це показано на рисунку 4.2.4.

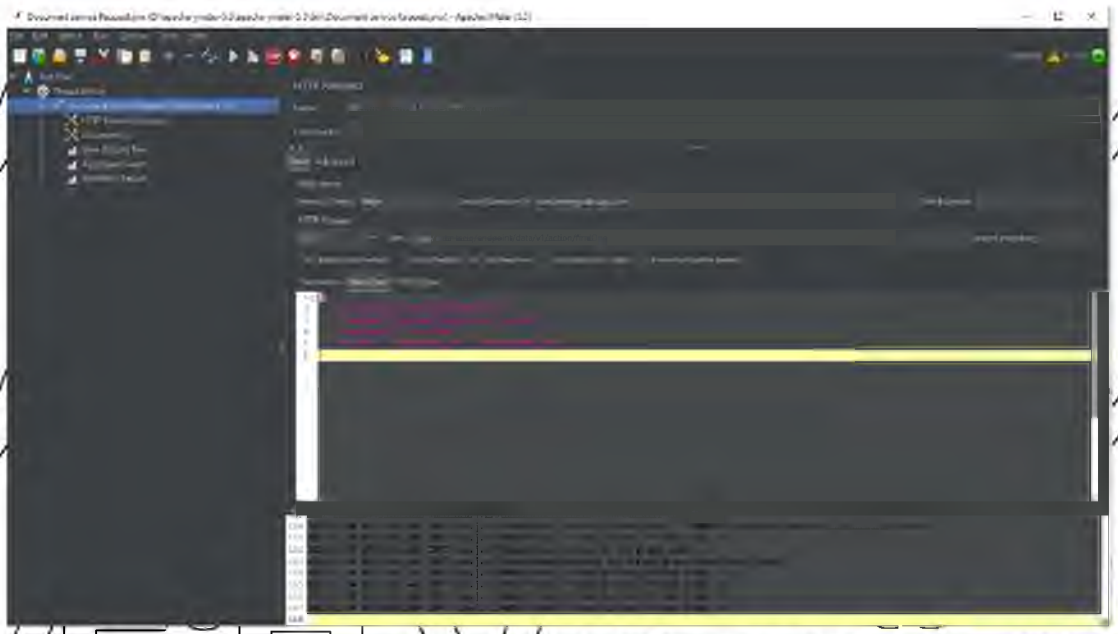


Рис. 4.2.4 Формування запиту.

Після того, як запит був сформований, починаємо тестування, для початку Запити в 5 потоках на вибірку різних документів і будемо чекати результат. Через деякий час ми можемо вже отримати деяких результат(рис. 4.2.5):

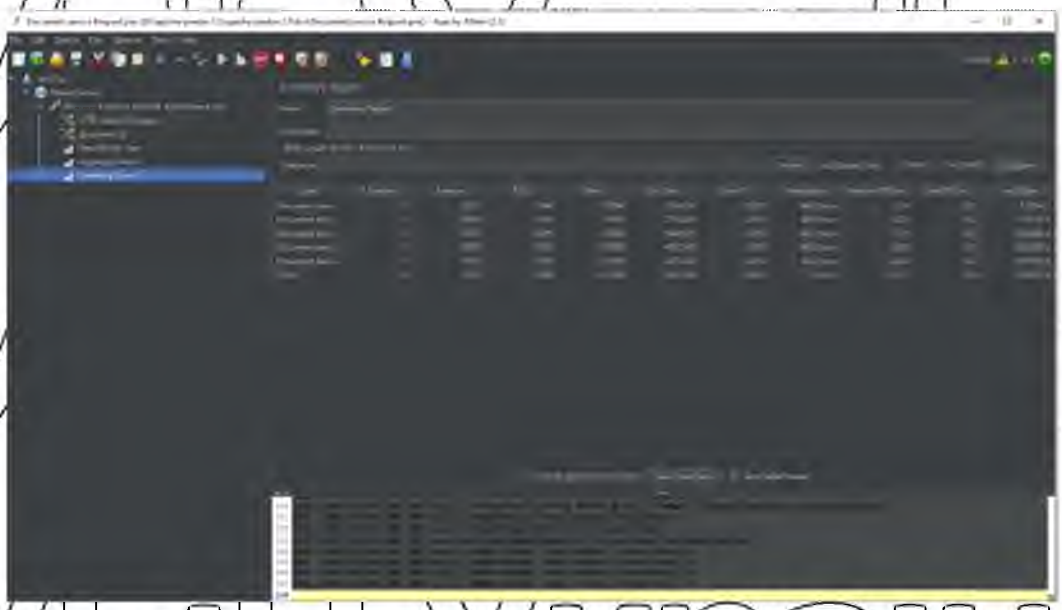


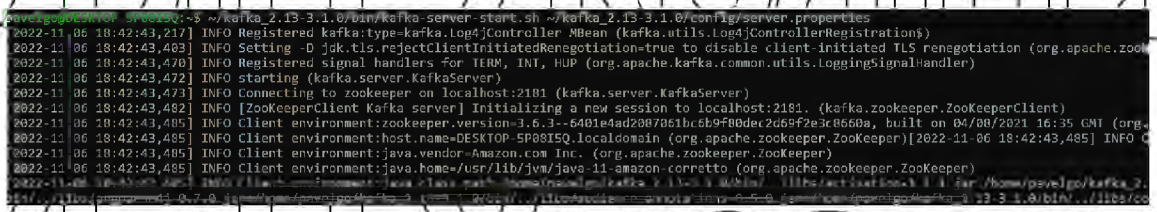
Рис. 4.2.4 Формування запиту.

Спостерігаючи результати можна побачити, що швидкість і справді більше, і це навіть не беручі дані, коли PostgreSQL має навантаження від інших сервісів.

4.3 Результати використання Apache Kafka

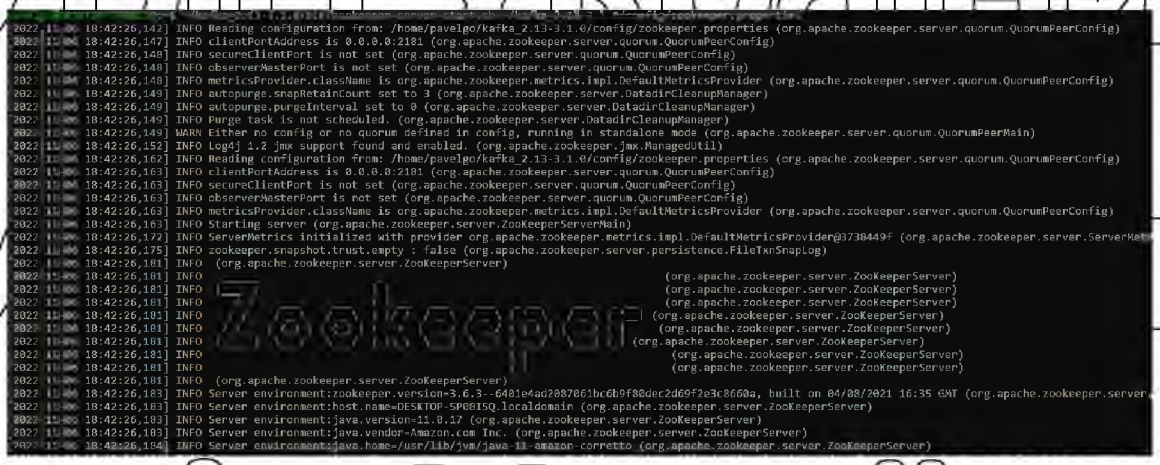
У даному дослідженні було перевірено нові функціональні можливості, що надаєте використання Apache Kafka – зберігання даних, навіть якщо ресурс бази даних, або сервер додатку не доступний. А також порівняємо швидкість обробки даних, швидкість обробки скоріш за все не збільшиться, зате навантаження на систему стане більш стабільним – система в піковий час буде обробляти лише стільки, скільки зможе, а все інше буде оброблено в більш напруженні години.

Тож, почнемо з функціональних можливостей. Для початку встановимо і запустимо Kafka. Для роботи Kafka потрібно також налаштувати Zookeeper – менеджер об'єктів всередині Kafka. На рис. 4.3.1 та 4.3.2 можна побачити запуск:



```
pavelgo@DESKTOP-5P08I5Q:~$ ~/kafka_2.13-3.1.0/bin/kafka-server-start.sh ~/kafka_2.13-3.1.0/config/server.properties
[2022-11-06 18:42:43,217] INFO Registered kafka:type=kafka.Log4jController MBean (kafka.utils.Log4jControllerRegistration$)
[2022-11-06 18:42:43,403] INFO Setting -D jdk.tls.rejectClientInitiatedRenegotiation=true to disable client-initiated TLS renegotiation (org.apache.zoo
[2022-11-06 18:42:43,470] INFO Registered signal handlers for TERM, INT, HUP (org.apache.kafka.common.utils.LoggingSignalHandler)
[2022-11-06 18:42:43,472] INFO starting (kafka.server.KafkaServer)
[2022-11-06 18:42:43,473] INFO Connecting to zookeeper on localhost:2181 (kafka.server.KafkaServer)
[2022-11-06 18:42:43,482] INFO [ZooKeeperClient Kafka server] Initializing a new session to localhost:2181. (kafka.zookeeper.ZooKeeperClient)
[2022-11-06 18:42:43,485] INFO Client environment:zookeeper.version=3.6.3--6401e4ad2087061bc6b9f80dec2d69f2e3c8660a, built on 04/08/2021 16:35 GMT (org.
[2022-11-06 18:42:43,485] INFO Client environment:host.name=DESKTOP-5P08I5Q.localdomain (org.apache.zookeeper.ZooKeeper)[2022-11-06 18:42:43,485] INFO C
[2022-11-06 18:42:43,485] INFO Client environment:java.vendor=Amazon.com Inc. (org.apache.zookeeper.ZooKeeper)
[2022-11-06 18:42:43,485] INFO Client environment:java.home=/usr/lib/jvm/java-11-amazon-corretto (org.apache.zookeeper.ZooKeeper)
```

Рис. 4.3.1 Запуск Apache Kafka



```
[2022-11-06 18:42:26,142] INFO Reading configuration from: /home/pavelgo/kafka_2.13-3.1.0/config/zookeeper.properties (org.apache.zookeeper.server.quorum.QuorumPeerConfig)
[2022-11-06 18:42:26,147] INFO clientPortAddress is 0.0.0.0:2181 (org.apache.zookeeper.server.quorum.QuorumPeerConfig)
[2022-11-06 18:42:26,148] INFO secureClientPort is not set (org.apache.zookeeper.server.quorum.QuorumPeerConfig)
[2022-11-06 18:42:26,148] INFO observerMasterPort is not set (org.apache.zookeeper.server.quorum.QuorumPeerConfig)
[2022-11-06 18:42:26,148] INFO metricsProvider.className is org.apache.zookeeper.metrics.impl.DefaultMetricsProvider (org.apache.zookeeper.server.quorum.QuorumPeerConfig)
[2022-11-06 18:42:26,149] INFO autopurge.snapRetainCount set to 3 (org.apache.zookeeper.server.DatadirCleanupManager)
[2022-11-06 18:42:26,149] INFO autopurge.purgeInterval set to 0 (org.apache.zookeeper.server.DatadirCleanupManager)
[2022-11-06 18:42:26,149] INFO Purge task is not scheduled. (org.apache.zookeeper.server.DatadirCleanupManager)
[2022-11-06 18:42:26,149] WARN Either no config or no quorum defined in config, running in standalone mode (org.apache.zookeeper.server.quorum.QuorumPeerMain)
[2022-11-06 18:42:26,152] INFO Log4j 1.2 jmx support found and enabled. (org.apache.zookeeper.jmx.ManagedUtil)
[2022-11-06 18:42:26,162] INFO Reading configuration from: /home/pavelgo/kafka_2.13-3.1.0/config/zookeeper.properties (org.apache.zookeeper.server.quorum.QuorumPeerConfig)
[2022-11-06 18:42:26,163] INFO clientPortAddress is 0.0.0.0:2181 (org.apache.zookeeper.server.quorum.QuorumPeerConfig)
[2022-11-06 18:42:26,163] INFO secureClientPort is not set (org.apache.zookeeper.server.quorum.QuorumPeerConfig)
[2022-11-06 18:42:26,163] INFO observerMasterPort is not set (org.apache.zookeeper.server.quorum.QuorumPeerConfig)
[2022-11-06 18:42:26,163] INFO metricsProvider.className is org.apache.zookeeper.metrics.impl.DefaultMetricsProvider (org.apache.zookeeper.server.quorum.QuorumPeerConfig)
[2022-11-06 18:42:26,163] INFO Starting server (org.apache.zookeeper.server.ZooKeeperServerMain)
[2022-11-06 18:42:26,172] INFO ServerMetrics initialized with provider org.apache.zookeeper.metrics.impl.DefaultMetricsProvider@3738449f (org.apache.zookeeper.server.ServerMet
[2022-11-06 18:42:26,175] INFO zookeeper.snapshot.trust.empty : false (org.apache.zookeeper.server.persistence.FileTxnSnapLog)
[2022-11-06 18:42:26,181] INFO  (org.apache.zookeeper.server.ZooKeeperServer)
[2022-11-06 18:42:26,181] INFO                                                    (org.apache.zookeeper.server.ZooKeeperServer)
[2022-11-06 18:42:26,181] INFO                                                    (org.apache.zookeeper.server.ZooKeeperServer)
[2022-11-06 18:42:26,181] INFO                                                    (org.apache.zookeeper.server.ZooKeeperServer)
[2022-11-06 18:42:26,181] INFO                                                  (org.apache.zookeeper.server.ZooKeeperServer)
[2022-11-06 18:42:26,181] INFO                                                   (org.apache.zookeeper.server.ZooKeeperServer)
[2022-11-06 18:42:26,181] INFO                                                 (org.apache.zookeeper.server.ZooKeeperServer)
[2022-11-06 18:42:26,181] INFO                                                     (org.apache.zookeeper.server.ZooKeeperServer)
[2022-11-06 18:42:26,181] INFO                                                     (org.apache.zookeeper.server.ZooKeeperServer)
[2022-11-06 18:42:26,181] INFO  (org.apache.zookeeper.server.ZooKeeperServer)
[2022-11-06 18:42:26,183] INFO Server environment:zookeeper.version=3.6.3--6401e4ad2087061bc6b9f80dec2d69f2e3c8660a, built on 04/08/2021 16:35 GMT (org.apache.zookeeper.server.
[2022-11-06 18:42:26,183] INFO Server environment:host.name=DESKTOP-5P08I5Q.localdomain (org.apache.zookeeper.server.ZooKeeperServer)
[2022-11-06 18:42:26,183] INFO Server environment:java.version=11.0.17 (org.apache.zookeeper.server.ZooKeeperServer)
[2022-11-06 18:42:26,183] INFO Server environment:java.vendor=Amazon.com Inc. (org.apache.zookeeper.server.ZooKeeperServer)
[2022-11-06 18:42:26,184] INFO Server environment:java.home=/usr/lib/jvm/java-11-amazon-corretto (org.apache.zookeeper.server.ZooKeeperServer)
```

Рис. 4.3.2 Запуск Zookeeper

Після запуску розроблює додаток Java, ціль якого буде відправляти дані та збирати дані з Kafka. Для відправки запити приходить по HTTP-протоколу. А після збору будуть відправлені в базу даних PostgreSQL.

Розробивши додаток, можна переходити до тестів, для початку перевіримо, чи працює взагалі наш сервіс. Для цього відправимо запит через Postman без ніякого логічного значення(рис. 4.3.3).

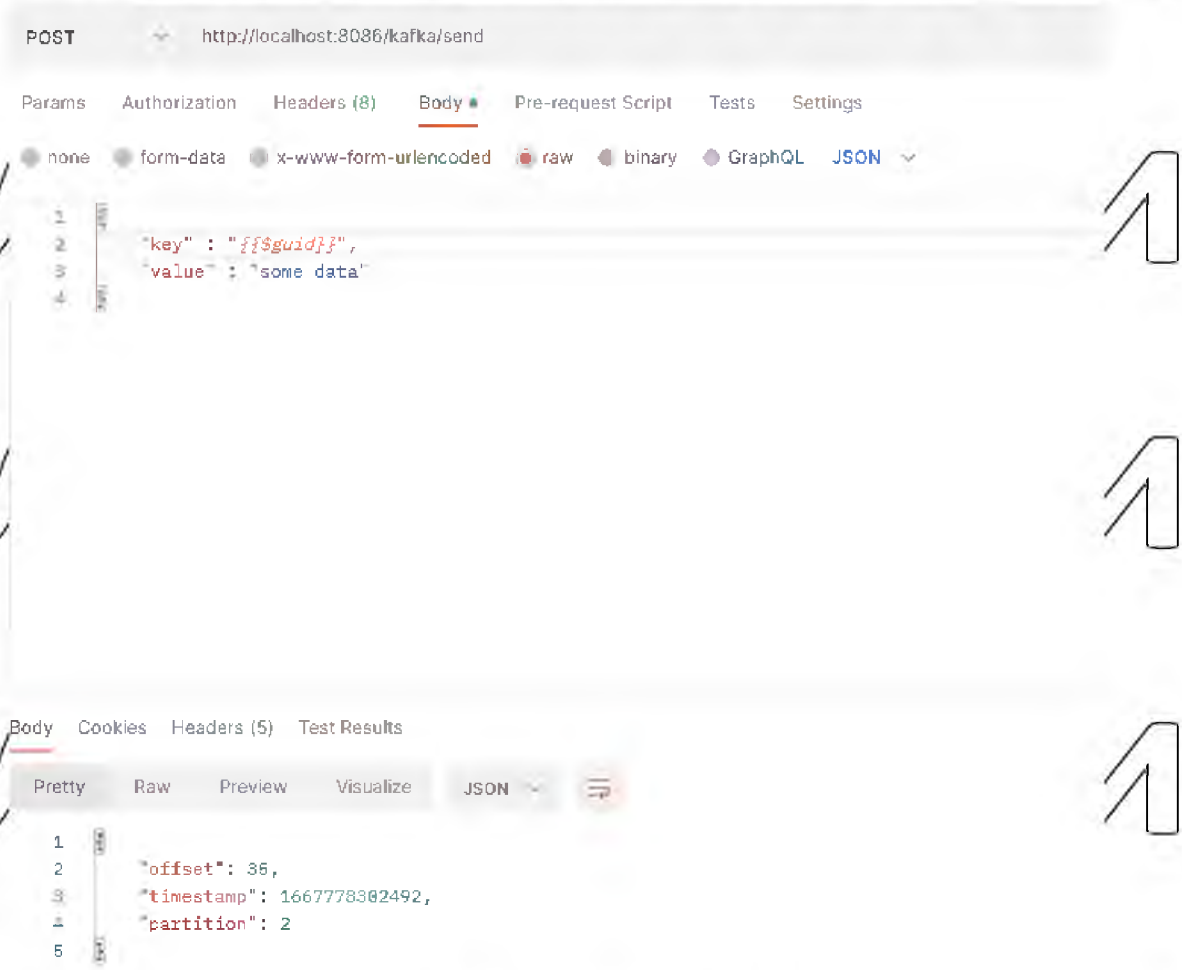


Рис. 4.3.3 Відправка запиту

Відправивши запит, ми отримали технічні дані, які можуть знадобитися для подальшої логіки системи. Це поля offsett – позиція в черзі Kafka, timestamp – час відправки запиту та partition – розділ черги, в який було записано повідомлення.

Перевіривши правильність обробки, можна виконати навантажувальне тестування. Для цього використаємо Apache JMeter. Для тестування запустимо 100 потоків, які будуть відправляти запити з інтервалом в 1 секунду нескінченно.

На рис. 4.3.4 та рис.4.3.5 можна побачити результат тестування відправки мільйона запитів:

Рис. 4.3.4 Результати тестування збереження повідомлень в Kafka

Також варто звернути увагу на навантаження бази даних при відправці повідомлень через брокер Kafka(рис. 4.3.5):

Рис. 4.3.5 Навантаження при тестуванні Kafka

Для порівняння і повної оцінки проведемо тестування без використання брокеру повідомлень, прямий запит і збереження одразу в базу даних (рис 4.3.6):

Label	# Samples	Average	Min	Max	Std. Dev.	Error %	Throughput	Received KB/sec	Sent KB/sec	Avg. Bytes
HTTP Request	1015056	11	0	350	11.78	0.01%	5986.7/sec	1565.78	1501.65	267.8
TOTAL	1015056	11	0	350	11.78	0.01%	5986.7/sec	1565.78	1501.65	267.8

Рис. 4.3.6 Навантаження при тестуванні без Kafka

Відправка даних відбулася трохи довше 3 хвилини, але вже видно деякі відмінності. Також порівняємо навантаження бази даних при такому принципу роботи(рис. 4.3.7):

Рис. 4.3.6 Навантаження при тестуванні без Kafka

Порівняння графіків баз даних дають нам ще більш чітке розуміння, що відбувається. Тож зробимо висновки:

- Середня швидкість обробки одного запита суттєво відрізняється, разом з Kafka – 12мс, без – 16мс, але варто розуміти, що запити з брокером повідомлень надходять в базу даних значно пізніше(знадобилось ще 10 хвилин для вивантаження усіх повідомлень);
- Кількість запитів відповідно була оброблена в секунду більша;
- Навантаження на базу даних має значну різницю – якщо брати версію з Kafka, то кількість транзакцій в середньому 2000 в секунду, коли без досягає 6000. Можна сказати, що брокер повідомлень «розтягує» навантаження на більший проміжок часу, зменшуючи саме навантаження;
- Важливо зрозуміти – так як тест був виконаний штучно, це не дає повної картини при звичайному робочому процесу. Але, можна розглянути ситуації, які можуть виникнути насправді – ресурс баз даних буде не доступний(технічні проблеми або блокування таблиць великими запитами, їх великої кількістю), тоді без брокеру повідомлень запити не зможуть обробитись одразу і просто будуть віддавати помилку по тайм-ауту, що призвете до помилок користувачів, теж саме може виникнути з сервером додатків. Kafka

не обробляє запити, а лише зберігає їх, тому запит виконується швидше;

- Так як запит збереження з брокером повідомлень, може надійти пізніше, то такі запити не підходять для тих, що повинні виконуватись одразу. Але, зазвичай, більшість операцій, можуть почекати оновлення даних. Наприклад, платіжні онлайн транзакції переводу, навіть в інтернет-банкінгу, якщо перевести другу гроші, то вони не одразу будуть отримані. Однак, якщо це наприклад покупка в магазині, то запит повинен виконатись одразу, щоб прийшла відповідь про виконану оплату.

Підсумувавши всі висновки, можна сказати, що брокер повідомлень, який є лідером на ринку, може дуже допомогти в стабільності роботи системи. Він є панацеєю від всіх проблем, але те, в чому він може використатись, там він показує себе дуже добре. Якщо розглядати майбутні розробки аналітичної системи в сфері логістики, то без цього рішення буде дуже важко – відслідковування вантажівок з товарами, особливо якщо їх велика кількість, може сильно навантажити системи. Так само з великим об’ємом даних експертних систем.

4.4 Результати використання Redis

Щоб оцінити користь використання даного рішення, для початку встановимо Redis, використаємо хмарну версію(рис. 4.2.1):

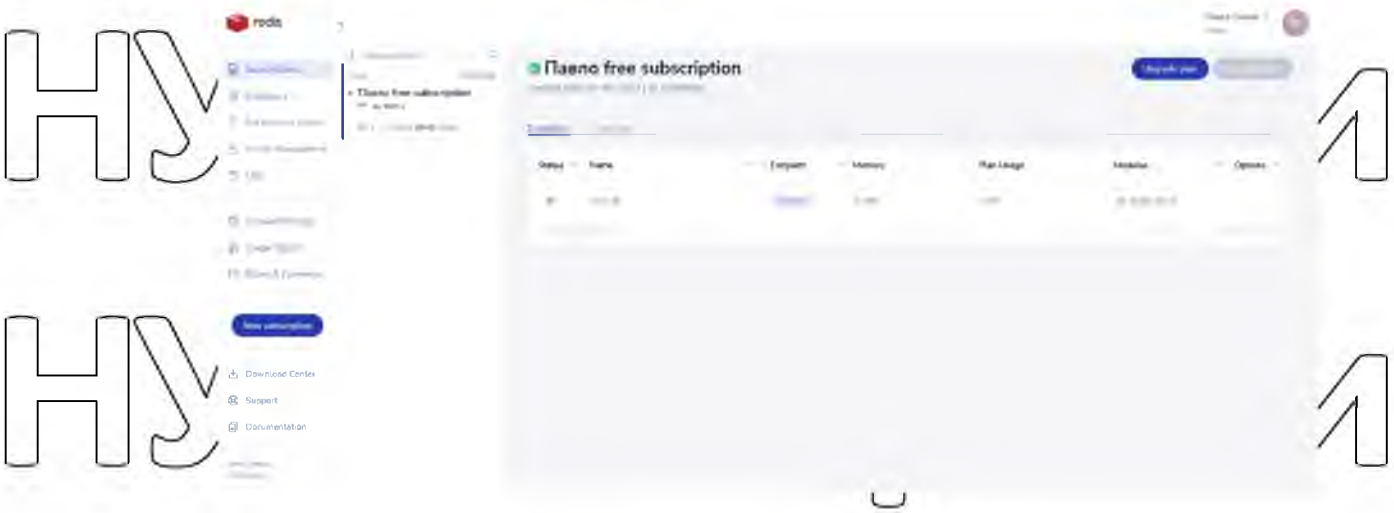


Рис. 4.4.1 Панель адміністратора Redis

Після встановлення, розпочнемо тестування, де будемо відправляти запити для отримання даних з бази даних з використанням Redis і без. Почнемо з використання кешу(рис. 4.4.2), час життя кешу визначений в 10 секунд(TTL):

Рис. 4.4.2 Тестування використовуючи Redis

Також вимірямо навантаження на базу даних(рис. 4.4.3):

Рис. 4.4.3 Навантаження бази даних використовуючи Redis

Тепер виконаємо теж саме, але без використання Redis(рис. 4.4.4):

Label	# Samples	Average	Min	Max	Std. Dev.	Error %	Throughput	Received KB/sec	Sent KB/sec	Avg. Bytes
HTTP Request	1039745	11	0	494	19.69	0.00%	8001.2/sec	2137.49	1272.76	273.6
TOTAL	1039745	11	0	494	19.69	0.00%	8001.2/sec	2137.49	1272.76	273.6

Рис. 4.4.4 Тестування без використання без Redis

Виміряємо навантаження на базу даних без використання кешу(рис. 4.4.5):

Рис. 4.4.5 Навантаження бази даних без Redis

Виконавши тестування можна підвести висновки:

- Середній час обробки є практично однаковим, кількість оброблених запитів теж є приблизно однакова;
- Найцікавіше відбувається з базою даних, пікові навантаження досягають 5 тисяч транзакцій, які є не постійні, а тільки моментами, коли у кешу виходить час дії(10 секунд) виникають запити в базу даних – весь інший час дані беруться з кешу. Без використання кешу ситуація кардинально інша, кількість транзакцій варіюється від 5 тисяч до пікових 25 тисяч, що є дуже багато і навантажує не тільки базу, а й ресурси самого серверу додатку.

З одної сторони, швидкість запитів не змінилось, хоча скоріш за все, це пов’язано з простотою запитів, що викликається(запит даних по id), ситуація зміниться, коли база даних буде більш навантажена або запити будуть більш комплексними і будуть виконуватись декілька або десятки секунд. З іншого боку, навантаження бази відчутно менше при використанні кешу. Але є нюанс, це те, що отримуються не самі актуальні дані. В нашому випадку це лише максимум 10 секунд, в запитах, що не потребують оперативної актуальної інформації, це не значить нічого. Це значення може змінити і потрібно тестувати в залежності від того, який виконується запит. Навіть враховуючи затримку, можна підтвердити, що Redis дуже позитивно вплинув на працездатність системи.

ВИСНОВОК

У ході магістерської кваліфікаційної роботи було розроблено аналітичну систему обліку аграрного підприємства – було проаналізовано предметну область, визначено технічні вимоги, створено модель системи і безпосередньо розроблено. Для розробки було визначено інструментарій розробки, зберігання даних, тестування, визначено алгоритми обробки. Після розробки було виконано тестування на основі якого складено результати досліджень. Дослідження полягали в тому, щоб визначити вплив різних баз даних на функціональність і продуктивність системи.

Програмний код серверної частини додатку був написаних на Java(фреймворк Spring), а клієнтської на мові JavaScript(фреймворк Vue.js), разом з мовою розмітки HTML та мовою каскадних таблиць CSS.

Для зберігання та маніпуляції даними використано 4 види баз даних, серед яких PostgreSQL, MongoDB, Apache Kafka та Redis.

PostgreSQL виступив основною базою даних, а інші бази даних відповідали за оптимізацію роботи PostgreSQL та системи в цілому. MongoDB була використана як база даних для зберігання документів угод, які укладалися під час покупок та продажів. Це дозволило зменшити навантаження з PostgreSQL і не дозволило великому комплексному запиту не блокувати інші запити користувачів. Apache Kafka відповідала, за синхроину обробку транзакцій бухгалтерії, що дозволило зробити відправку важливих фінансових запитів більш безпечною за рахунок зберігання даних у потоках Kafka, навіть, якщо сервер додатку не міг би обробляти запити в момент критичного навантаження та ситуацій або технічних робіт. Redis був використаний як сервер кешування, який надавав користувачам дані, які були застарілі на рівні PostgreSQL, але були все одно необхідні, доречні та актуальні. Це зробило швидкість обробки запитів більш швидким і також зменшило навантаження на систему.

В ході дослідження було підтверджено гіпотезу про доцільність використання декількох засобів зберігання даних. З результатами дослідження можна ознайомитись в розділі 4. Дослідження проводилось порівнянням стандартних алгоритмів з одної базою даних, яка виконувала всі функції самостійно та використанням допоміжних баз даних.

В результаті, аналітична система обліку аграрного підприємства змогла покрити всі задачі, які від неї вимагались, серед яких були задачі обліку та аналітики. Крім того, система була оптимізована за рахунок розділення обов'язків на різні модулі. Отримані результати дозволяють у перспективі продовжувати експериментувати з іншими види баз даних, а також за рахунок гнучкої архітектури, збільшувати функціонал системи більш спеціалізованими пакета, наприклад, таких як аналітика обліку персоналу або інтеграції з експертними системами аграрної сфери.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Косуіна Уляна., Харроп Роб., Чафер Кріс, Хо Кларенс. Pro Spring 5. Fifth Edition, 2017.
2. Tiago Macedo, Fred Oliveira. Redis Cookbook, 2011.
3. Neha Narkhede, Gwen Shapira, Todd Palino. Kafka: The Definitive Guide, 2017.
4. Regina Obe, Leo Hsu. PostgreSQL: Up and Running, Third Edition, 2018
5. Kristina Cbodorow. MongoDB: The Definitive Guide, Second Edition, 2013.
6. Introduction — Vue.js [Електронний ресурс] – Режим доступу до ресурсу: https://vuejs.org/v2/guide/
7. Spring Boot [Електронний ресурс] – Режим доступу до ресурсу: https://spring.io/projects/spring-boot
8. PostgreSQL: The World`s Most Advanced Open Source Relational Database [Електронний ресурс] – Режим доступу до ресурсу: https://www.postgresql.org/
9. Unified Modeling Language(UML) Resource Page [Електронний ресур] – Режим доступу до ресурсу: http://www.uml.org/
10. Axios – npm [Електронний ресурс] – Режим доступу до ресурсу: https://www.npmjs.com/package/axios
11. Chart.js | Open Source HTML5 Charts – Режим доступу до ресурсу: https://www.chartjs.org/
12. Project Lombok [Електронний ресурс] – Режим доступу до ресурсу: https://projectlombok.org/
13. Jackson Tutorial | Bael dung [Електронний ресурс] – Режим доступу до ресурсу: https://projectlombok.org/
14. API Documentation & Design Tools For Teams | Swagger [Електронний ресурс] – Режим доступу до ресурсу: https://swagger.io/
15. Apache Tomcat 10 [Електронний ресурс] – Режим доступу до ресурсу: http://tomcat.apache.org/tomcat-10.0-doc/introduction.html

16.Math – The Commons Math User Guide [Електронний ресурс] – Режим доступу до ресурсу: http://tomcat.apache.org/tomcat-10.0-doc/introduction.html