Міністерство освіти і науки України

Харківський національний університет радіоелектроніки

Факультет Інфокомунікацій
(повна назва)

Кафедра Інформаційно-мережної інженерії
(повна назва)

КВАЛІФІКАЦІЙНА РОБОТА

Пояснювальна записка

рівень вищої освіти перший (магістерський)
(рівень вищої освіти)

«Розробка білінг додатку на мові Go»

(тема)

Виконав:

студент 2 курсу, групи ІМІм-22-1

Дроздов Д.О.
(прізвище, ініціали)

Спеціальність 172 «Телекомунікації та радіотехніка»
(код і назва спеціальності)

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

Освітня програма «Інформаційно-мережна інженерія»
(повна назва освітньої програми)

Керівник доцент Чеботарьова Д.В.
(посада, прізвище, ініціали)

Допускається до захисту

Зав. кафедри _____ Безрук В.М.
(підпис) (прізвище, ініціали)

2024 р.

Не містить відомостей, заборонених до відкритого публікування

Студент _____/____Дроздов Д.О.__/

Керівник _____/__Чеботарьова Д.В._/

Харківський національний університет радіоелектроніки

Факультет Інфокомунікацій
(повна назва)

Кафедра Інформаційно-мережної інженерії
(повна назва)

Рівень вищої освіти другий (магістерський)
(рівень вищої освіти)

Спеціальність 172 «Телекомунікації та радіотехніка»
(код і назва спеціальності)

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

Освітня програма «Інформаційно-мережна інженерія»
(повна назва освітньої програми)

ЗАТВЕРДЖУЮ:
Зав. кафедри _____
(підпис)
«____» _____ 20____ р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ

студентові Дроздову Данилу Олексійовичу
(прізвище, ім'я, по батькові)

1. Тема роботи «Розробка білінг додатку на мові Go»

затверджена наказом університету від 23 10 2023 р. № 1233 Ст

2. Термін подання студентом роботи до екзаменаційної комісії 16 01 2024 р.

3. Вихідні дані до роботи: мова програмування Go, вебфреймворк Gin, база даних Postgres, інструмент для визначення та управління багато-контейнерними додатками docker-compose та програма для візуалізації графіків mathcad

4. Перелік питань, що потрібно опрацювати в роботі:

Базові поняття проектування вебдодатків;

Основні технології створення вебдодатків;

Розробка білінг додатку на мові Go

5. Перелік графічного матеріалу із зазначенням креслень, схем, плакатів, комп'ютерних ілюстрацій (п.5 включається до завдання за рішенням випускної кафедри) *слайди у форматі PowerPoint*

КАЛЕНДАРНИЙ ПЛАН

№	Назва етапів роботи	Терміни виконання етапів роботи	Примітка
1	Ознайомлення із завданням.	23.10.2023	*виконано*
2	Підбір літератури за темою роботи.	24.10 – 29.10.2023	*виконано*
3	Виконання розділу 1	30.10 – 07.11.2023	*виконано*
4	Виконання розділу 2	08.11 – 15.11.2023	*виконано*
5	Виконання розділу 3	16.11 – 31.12.2023	*виконано*
6	Оформлення презентаційного матеріалу, підготовка до захисту у ЕК	01.01 – 14.01.2024	*виконано*

Дата видачі завдання 23 жовтня 2023 р.

Студент _____
(підпис)

Керівник роботи _____ доц. Чеботарьова Д. В.
(підпис) (посада, прізвище, ініціали)

РЕФЕРАТ

Пояснювальна записка: 78 с., 45 рис., 5 табл., 3 додатки, 13 джерел.

ДОДАТОК, БІЛІНГ, GO, CSV, ФАЙЛ, HTTP, GRPC, POSTGRES

Об'єкт дослідження – сирі CSV дані

Мета роботи – є розробка білінг додатку на мові Go для покращення роботи з сирими даними.

Результати – в роботі розглянуто основні типи файлів, які використовуються для передачі даних, технології gRPC й REST, вибір бази даних Postgres, мови програмування Go та використання Docker для локальної розробки. У додатку реалізовано завантаження CSV-файлу з транзакціями, отримання відфільтрованих транзакцій, отримання транзакцій у форматі CSV, оптимізація ендпоінтів для завантаження файлу. В результаті експерименту встановлено, що витрати пам'яті при витягуванні всього файлу суттєво впливають на продуктивність додатку. Хоча оптимізація коду дозволила зменшити обсяг використовуваної пам'яті, виявилося, що це призводить до збільшення часу обробки файлу.

THE ABSTRACT

Explanatory note: 78 p., 45 fig., 5 tab., 3 app., 13 sources.

APPLICATION, BILLING, GO, CSV, FILE, HTTP, GRPC, POSTGRES

Object of research – raw CSV data.

The purpose of qualification work is developing a billing application in the Go language to enhance working with raw data.

Result – the work covers the main types of files used for data transmission, gRPC and REST technologies, the choice of the Postgres database, the Go programming language, and the use of Docker for local development. The application implements the loading of a CSV file with transactions, retrieval of filtered transactions, obtaining transactions in CSV format, and optimizing endpoints for file loading. As a result of the experiment, it was established that memory expenses significantly impact the application's performance when extracting the entire file. Although code optimization reduced the memory usage, it was found to increase the file processing time.

ЗМІСТ

ПЕРЕЛІК СКОРОЧЕНЬ

API (Application Programming Interface) – прикладний програмний інтерфейс;

CSV (Comma-Separated Values) – текстовий формат файлів;

HTTP (HyperText Transfer Protocol) – протокол прикладного рівня передачі даних, спочатку у вигляді гіпертекстових документів у форматі HTML gRPC (Remote Procedure Calls) – це система віддаленого виклику процедур;

REST (Representational State Transfer) – архітектурний стиль взаємодії компонентів розподіленої програми в мережі;

SQL (Structured Query Language) – декларативна мова програмування, яка використовується для створення, модифікації та управління даними в реляційній базі даних;

DBMS (Database Management System) – сукупність програмних та лінгвістичних засобів загального або спеціального призначення, що забезпечують управління створенням та використанням баз даних;

JSON (JavaScript Object Notation) – текстовий формат обміну даними на основі JavaScript;

XML (eXtensible Markup Language) – розширювана мова розмітки.

ВСТУП

У сучасному світі, коли обсяги даних невпинно зростають, розробка та впровадження ефективних інструментів для обробки та аналізу інформації стають критично важливим завданням. Мета роботи полягає в створенні білінгового додатку на мові програмування Go, орієнтованого на поліпшення роботи з сирими даними. Ця мета визначена як ключова, оскільки вимагає з'єднання високотехнологічних рішень із сучасними вимогами щодо обробки і зберігання інформації.

Одним із важливих аспектів цього дослідження є поліпшення використання пам’яті додатком при незначній втраті часу, який використовується на обробку файлу. Важливо зазначити, що обробка сирої інформації потребує дбайливого підходу до вибору інструментів для комунікації між різними компонентами системи. У цьому дослідженні розглядаються два важливі протоколи – REST та gRPC, аналізується їх застосування для забезпечення ефективної взаємодії між компонентами додатку. Ще однією ключовою складовою дослідження є вибір системи управління базами даних для зберігання інформації. На цьому етапі обрана система PostgreSQL, враховуючи її надійність, широкі можливості та високий рівень сумісності з мовою програмування Go. Для забезпечення надійності та масштабованості додатку, була вибрана технологія контейнеризації Docker. Ця технологія надає можливість ефективного розгортання та управління додатками у різних середовищах, забезпечуючи стабільну роботу системи.

На сьогоднішній день проблема обробки сирої інформації є надзвичайно важливою, саме тому кваліфікаційна робота, що присвячена розробці додатку для роботи з сирими даними є актуальною.

1 БАЗОВІ ПОНЯТТЯ ПРОЕКТУВАННЯ ВЕБДОДАТКІВ

1.1 Основні типи файлів для передачі даних

Розширення імені файлу, розширення імені файлу або розширення файлу – це суфікс до імені комп'ютерного файлу (наприклад, .txt, .docx, .md). Розширення вказує на характер вмісту файлу або його призначене використання. Розширення імені файлу зазвичай відокремлюється від решти імені файлу крапкою, але в деяких системах воно відокремлюється пробілами. Інші формати розширень можуть включати дефіси і/або підкреслення на ранніх версіях Linux і деяких версіях IBM AIX [1].

Деякі файлові системи реалізують розширення імен файлів як функцію самої файлової системи і можуть обмежувати довжину та формат розширення, тоді як інші розглядають розширення імен файлів як частину імені файлу без особливого розрізнення.

Розширення імен файлів (табл. 1.1) можна розглядати як один вид метаданих. Зазвичай вони використовуються для передачі інформації про те, як дані можуть бути збережені в файлі. Точне визначення, що визначає критерії вибору тієї частини імені файлу, яка є його розширенням, належить правилам конкретної файлової системи, яка використовується; зазвичай розширення - це підрядок, який слідує за останнім входженням крапки, якщо воно є (приклад: txt - це розширення імені файла readme.txt, а html - розширення імені mysite.index.html). На файлових системах деяких головних фреймворків, таких як CMS в VM, VMS, і системах ПК, таких як CP/M та похідні системи, такі як MS-DOS, розширення є окремим простором імен від імені файла. Під DOS і Windows від Microsoft розширення, такі як EXE, COM або BAT, вказують на те, що файл є виконуваним програмним файлом. В OS/360 і його наступниках частина імені набору даних, яка слідує за останньою крапкою і називається низьким рівнем кваліфікації, розглядається як розширення деякими

програмами, наприклад, TSO EDIT, але воно не має особливого значення для операційної системи самої по собі; те ж саме стосується файлів Unix в MVS [1].

Спочатку розширення імен файлів використовувалось для визначення загального типу файлу.[потрібне джерело. Потреба у стисненні типу файлу до трьох символів часто призводила до абревіатур розширень. Приклади включають використання .GFX для графічних файлів, .TXT для простого тексту і .MUS для музики. Проте через те, що було створено багато різних програм, які всі обробляють ці типи даних (і інші) різними способами, розширення імен файлів стали тісно пов'язаними з певними продуктами - навіть конкретними версіями продуктів. Наприклад, ранні файли WordStar використовували .WS або .WSn, де n був номером версії програми. Також розвивалися конфліктні використання деяких розширень імен файлів. Один приклад – .rpm, яке використовується як для пакетів RPM Package Manager, так і для файлів медіапрогравача RealPlayer; Інші приклади - .qif, яке використовується для шрифтів DESQview, фінансових журналів Quicken і зображень QuickTime; .gba, яке використовується для сценаріїв GrabIt і образів ROM для ігрового пристрою Game Boy Advance; .sb, використовується для SmallBasic і Scratch; і .dts, яке використовується для Dynamix Three Space і DTS [1].

У багатьох Інтернет-протоколах, таких як HTTP та MIME-пошта, тип потоку бітів вказується як медіа-тип або тип MIME потоку, а не розширення імені файлу. Це подається в рядку тексту перед потоком, наприклад Content-type: text/plain.

Не існує стандартного відображення між розширеннями імен файлів та медіа-типами, що може призводити до можливих невідповідностей у тлумаченні між авторами, вебсерверами та програмним забезпеченням клієнта під час передачі файлів через Інтернет. Наприклад, автор контенту може вказати розширення svgz для стиснутого файлу масштабованої векторної графіки, але вебсервер, який не впізнає це розширення, може не відправити належний тип контенту application/svg+xml та відповідний заголовок

стиснення, що залишить веббраузери неспроможними правильно інтерпретувати та відображати зображення.

BeOS, яка має файлову систему BFS з підтримкою розширених атрибутів, позначала файл медіа-типом як розширений атрибут. Середовища робочого столу KDE та GNOME асоціюють медіа-тип з файлом, досліджуючи як суфікс імені файлу, так і вміст файлу, у стилі команди file, як евристику. Вони вибирають програму для запуску, коли відкривається файл, на основі цього медіа-типу, зменшуючи залежність від розширень імен файлів. macOS використовує як розширення імен файлів, так і медіа-типи, а також коди типів файлів, для вибору ідентифікатора рівномірного типу, за яким відомо внутрішній тип файлу [1].

Таблиця 1.1 – Основні розширень файлів

Назва розширення файлу	MIME-тип	Опис розширення
.txt	text/plain	Розширення .txt вказує на текстовий файл, який містить звичайний текст без форматування. Такі файли можна відкрити та редагувати у текстових редакторах.
.doc	application/msword	Файли з розширенням .doc є документами у форматі Microsoft Word. Вони можуть містити текст, зображення та форматування та зазвичай створюються та редагуються у програмі Microsoft Word.

Продовження таблиці 1.1

Назва розширення файлу	MIME-тип	Опис розширення
.docx	application/vnd. openxmlformats-officedocument. wordprocessingml.document	Це розширений формат документа Microsoft Word, введений у більш пізніх версіях програми. В файлі .docx може бути багато вмісту, включаючи текст, зображення, таблиці та інші об'єкти.
.pdf	application/pdf	Розширення .pdf вказує на документ у форматі Portable Document Format (PDF). PDF-файли зберігають форматування, зображення та текст так, що їх можна переглядати та друкувати на різних пристроях зі збереженням вигляду.
.jpg, .jpeg	image/jpeg	Файли .jpg та .jpeg містять зображення у форматі JPEG, який використовується для зберігання фотографій та інших зображень з високою якістю при зменшенні розміру файлу.
.png	image/png	Розширення .png також вказує на файл зображення, але у форматі Portable Network Graphics (PNG). PNG-файли зберігають високу якість та підтримують прозорість.

Продовження таблиці 1.1

Назва розширення файлу	MIME-тип	Опис розширення
.gif	image/gif	Файли .gif містять анімовані або статичні зображення, зазвичай використовуються для коротких анімацій та вебграфіки.
.mp3	audio/mpeg	Розширення .mp3 вказує на аудіофайл у форматі MP3, який використовується для зберігання музики та інших аудіозаписів з високою стискальністю.
.mp4	video/mp4	Файли .mp4 містять відео та аудіо та часто використовуються для зберігання відеороликів та фільмів.
.xlsx	application/vnd. openxmlformats-officedocument. spreadsheetml.sheet	Розширення .xlsx вказує на файл таблиці Microsoft Excel у форматі XLSX, який містить дані, таблиці та формули.
.csv	text/csv	Файли .csv містять дані у текстовому форматі, де значення розділені комами (або іншими роздільниками). Зазвичай використовується для імпорту та експорту даних.
.html, .htm	text/html	Розширення .html та .htm вказують на файли вебсторінок у форматі HyperText Markup Language

Продовження таблиці 1.1

Назва розширення файлу	MIME-тип	Опис розширення
.xml	application/xml	(HTML), які використовуються для створення вебсайтів. Файли .xml містять дані у форматі Extensible Markup Language (XML), який використовується для зберігання та обміну структурованими даними.
.zip	application/zip	Розширення .zip вказує на архівований файл, який містить один або кілька інших файлів, стиснені в один архів для зручності зберігання та передачі.

1.2 Вебдодатки на основі мікросервісів

Додатки на основі мікросервісів подолують найбільші обмеження традиційних, монолітних додатків. Монолітний додаток містить програмний код для всіх своїх служб і функцій у єдиній нероздільній кодовій базі, яка керує всіма службами і функціями додатка. Багато компаній починають з монолітного додатку, оскільки це часто є найпростішим варіантом [2].

При додаванні нових служб та функцій до існуючого фреймворку стає все складніше змінювати, оновлювати та масштабувати додаток. Зміна однієї частини додатка може негативно вплинути на інші області. Після декількох масштабувань, оновлень і змін моноліту кодова база стає настільки взаємозалежною і важко зрозумілою, що необхідно повністю перепроектувати весь систему з нуля. Тому все більше компаній працюють поза традиційною

монолітною системою. Вони віддають перевагу використанню мікросервісів [2].

Архітектура додатка на основі мікросервісів вирішує проблему взаємозв'язку всього. Стиль архітектури розбиває моноліт на складові служби та запускає кожну компоненту як автономний додаток. Ці компоненти називаються мікросервісами. Ці окремі мікросервіси використовують API для взаємодії один з одним. Разом ця група мікросервісів, підключених через API, формує більшу архітектуру додатка [2].

Через те, як API дозволяють автономним мікросервісам підключатися один до одного, вони дозволяють створити систему, яка базується на компонентах та є підключуваною. Вони можуть працювати разом та взаємодіяти навіть якщо мікросервіси розроблені і програмовані на різних мовах програмування або на різних платформах. Це робить їх надзвичайно корисними і також забезпечує можливість використовувати мікросервіси, які відповідають потребам користувачів[2].

Оновлення окремих мікросервісів набагато швидше та простіше, оскільки зміни, які вносяться в автономний сервіс, мають менший вплив на весь систему. Масштабування також стає простішим і ефективнішим. Ресурси можуть бути спрямовані на мікросервіси, які їх потребують, в залежності від вимог до використання. Крім того, якщо один мікросервіс відмовляє або сповільнюється, ймовірність падіння всієї інфраструктури менше. Все це перекладається на більш ефективні, стійкі, масштабовані та гнучкі системи, а API роблять це можливим [2].

Мікросервіси використовуються по всьому світу та великими компаніями, а також меншими підприємствами. Не дивно, що Netflix, WordPress, Uber і Amazon - це деякі з компаній, які використовують їх. Здатність розширюватися практично необмежено та надавати якісні послуги та функціональність, мікросервіси мають сенс.

Існує ситуації, коли пропонуються отримати API для використання на вебсайті або блозі. Їх можна використовувати для підключення малих бізнесів та функцій, а також великих. Їх можна масштабувати за потребою.

Починаючи роботу, компанії повинні думати про майбутнє. Вони повинні розглядати, які додатки будуть потрібні через рік або п'ять років, які можливості можуть знадобитися при зростанні бізнесу. Початок - це буквально лише початок, і короткозоро краще починати з системи, яка буде рости разом з компанією. Замість цього найкраще розглядати мікросервіси і налаштовувати все з цим на увазі. Процес може зайняти трохи більше часу, але з часом система буде працювати краще. Це також веде до менших витрат часу та грошей протягом років [2].

Найбільш поширеним архітектурним стилем для API є REST API. Однак також існують RPC API та gRPC API.

1.3 Архітектурний стиль побудови вебдодатків REST API

REST (Representational State Transfer) описує організацію клієнт-сервер, в якій дані з сервера стають доступними для клієнтів через формат обміну повідомленнями JSON (рис. 1.1) або XML. Згідно з визначенням Роя Філдінга, API вважається "RESTful", коли воно відповідає обмеженням наведеним нижче.

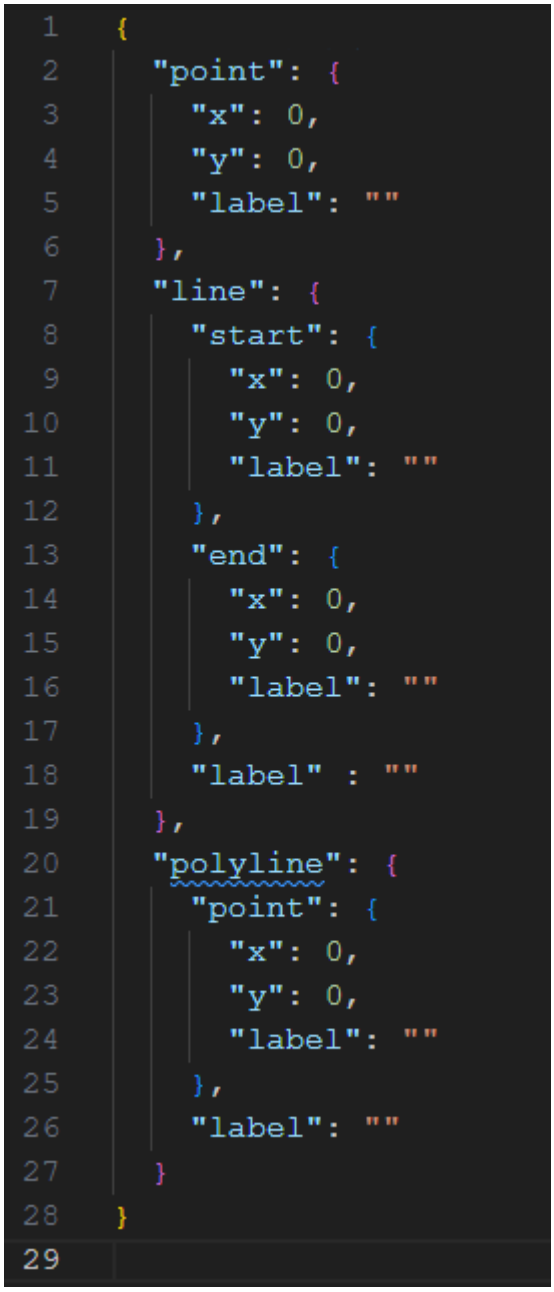
Однорідний інтерфейс: API повинен відкривати конкретні ресурси додатка для споживачів API.

– Незалежність клієнта і сервера: клієнт та сервер працюють незалежно один від одного. Клієнт буде знати тільки URIs, які вказують на ресурси додатка. Зазвичай ці URIs публікуються в документації API.

– Безстанційний: сервер не зберігає дані, що стосуються запиту клієнта. Клієнт зберігає ці "дані стану" на своєму боці (за допомогою кешу).

– Кешування: ресурси додатка, доступні через API, повинні бути піддаються кешуванню.

– Шарована архітектура: архітектура має бути шарованою, що дозволяє різним компонентам зберігатися на різних серверах.



```
{
  "point": {
    "x": 0,
    "y": 0,
    "label": ""
  },
  "line": {
    "start": {
      "x": 0,
      "y": 0,
      "label": ""
    },
    "end": {
      "x": 0,
      "y": 0,
      "label": ""
    },
    "label" : ""
  },
  "polyline": {
    "point": {
      "x": 0,
      "y": 0,
      "label": ""
    },
    "label": ""
  }
}
```

Рисунок 1.1 – Приклад описання JSON структур

– Code-on-Demand (COD): це єдине необов'язкове обмеження REST. Це дозволяє клієнту отримувати виконуваний код сервера як відповідь. Іншими словами, сервер визначає, які конкретні дії виконуються.

Також важливо відзначити, що практично у всіх випадках REST API використовує протокол HTTP. Це найпоширеніший формат, який використовується у вебдодатках або для взаємодії між мікросервісами. Після того, як REST API стає загальнодоступним у вебслужбі, клієнти можуть використовувати кожен компонент як ресурс. Зазвичай ресурси доступні через загальний інтерфейс, який приймає різні HTTP команди, такі як GET, POST, DELETE і PUT [2].

Можно побачити REST API в дії на таких сайтах, як Facebook та Mailchimp. Ці додатки використовують архітектуру API по-різному, але вони все ще схожі. Інші сайти, які використовують REST, включають Spotify, Netflix та Uber [2].

Створення REST API – це відмінний спосіб забезпечити себе тим, що отримується саме те, що потрібно.

1.4 Технології RPC API та gRPC API

RPC або віддалений виклик процедури був створений спочатку, у 1970-х роках і вважається попередником REST. Незважаючи на те, що ви, безумовно, можете використовувати REST API, не знаючи історії, ви, можливо, бажатимете дізнатися трохи більше про те, як працюють ці старі методи. RPC викликає функцію на віддаленому сервері, але, на відміну від новіших API, він використовує конкретний формат і повинен отримувати такий самий формат у відповідь [3].

Основна концепція RPC API подібна до концепції REST API. RPC API визначає правила взаємодії та методи, які клієнт може використовувати для взаємодії з ним. Клієнти надсилають виклики, які використовують "аргументи", щоб викликати ці методи. Проте техніка знаходиться в URL у випадку RPC API. Аргументи, які викликають методи, знаходяться у рядку запиту. Щоб проілюструвати це, ось як запит RPC API порівнюється з запитом REST API:

RPC: Запит RPC API може виглядати, наприклад, як POST /deleteResource та містити рядок запиту { "id": 3 }.

REST: У REST API запит буде виглядати так DELETE /resource/2.

RPC вважається дещо застарілим, і його рідко можно знайти використовуваним в сучасних системах. У більшості випадків його замінили іншими варіантами. Інші два варіанти швидко витіснили все, що працювало з використанням RPC.

Як варіант архітектури RPC, gRPC був створений компанією Google у 2015 році для прискорення передачі даних між мікрослужбами та іншими системами, які повинні взаємодіяти. Ця архітектура API відрізняється від попередніх API з кількох причин [3].

По-перше, gRPC використовує інший формат. Замість JSON, це API використовує Protobuf. Protobuf також був розроблений Google і є мово- та платформонезалежним. Він схожий на XML, але вважається більш ефективним.

Ця структура API також використовує HTTP2, замість оригінального HTTP1, і значно швидше за оригінальний RPC. Це означає, що його легше впроваджувати, оскільки він може передавати повідомлення вдекілька разів швидше, ніж попередні версії. Для більших додатків це робить можливим управління стороною комунікації, хоча він менш швидкий, ніж REST під час впровадження API [3].

Протоколи Buffers (Protobufs) не мають єдиного стандарту. Цей формат найбільше підходить для невеликих фрагментів даних, які не перевищують кількох мегабайтів і можуть бути завантажені/відправлені в пам'ять безпосередньо, тому він не є потоковим форматом. Бібліотека не надає компресію "з коробки". Формат також погано підтримується в мовах, які не є об'єктно-орієнтованими (наприклад, Fortran) [3].

Схема для конкретного використання протоколу Protocol Buffers асоціює типи даних з іменами полів, використовуючи цілі числа для ідентифікації кожного поля. Дані протоколу Protocol Buffers містять лише числа, а не імена

полів, що забезпечує економію пропускної здатності та зберігання порівняно із системами, які включають імена полів у дані (рис.1.2) [3].

```
// polyline.proto
syntax = "proto2";

message Point {
  required int32 x = 1;
  required int32 y = 2;
  optional string label = 3;
}

message Line {
  required Point start = 1;
  required Point end = 2;
  optional string label = 3;
}

message Polyline {
  repeated Point point = 1;
  optional string label = 2;
}
```

Рисунок 1.2 – Приклад описання Protobuf структур

Структура "Point" визначає два обов'язкових елементи даних, x та y. Мітка елемента даних є необов'язковою. Кожен елемент даних має тег. Тег визначається після знаку рівності. Наприклад, у x тег має номер 1.

Структури "Line" і "Polyline", які обидві використовують тип "Point", демонструють, як працює композиція в протоколі Protocol Buffers. У Polyline є повторний (repeated) поле, яке веде себе подібно до вектора.

1.5 Порівняння архітектурних стилів gRPC та REST

Незважаючи на схожі функції між REST і gRPC, їх базові моделі значно відрізняються за своєю архітектурою.

Порівняння архітектурних стилів gRPC та REST наведено в табл. 1.2.

Таблиця 1.2 – Порівняння архітектурних стилів gRPC та REST

Відмінність	REST	gRPC
Комунікаційна модель	При використанні REST API клієнт відправляє на сервер один запит REST API, і сервер надсилає одну відповідь на цей запит. Клієнт повинен зачекати відповіді від сервера, перш ніж продовжити роботу. Цей механізм представляє собою модель "запит-відповідь" і є одночасною передачею даних (один-до-одного) [4].	У разі використання gRPC клієнт може відправити один або кілька API-запитів на сервер, які можуть призвести до однієї або декількох відповідей від сервера. Передача даних може бути односторонньою (один-до-одного), потоковою (один-до-багатьох), потоковою від клієнта (багато-до-одного) або двонапрямленою (багато-до-багатьох).
Виклик операцій на сервері	У gRPC API виклик операцій на сервері здійснюється за допомогою сервісів, які також відомі як функції або процедури. Клієнт gRPC викликає ці функції так само, як і сервіс всередині	Цей механізм представляє собою комунікаційну модель "відповідь для передачі на сторону клієнта" і можливий завдяки тому, що gRPC працює на основі HTTP 2 архітектурою.

Продовження таблиці 1.2

Відмінність	REST	gRPC
	додатку. Це називається сервіс-орієнтованою архітектурою. createNewOrder(customer_id, item_id, item_quantity) -> order_id [4].	Суб'єкт- орієнтована архітектура добре поєднується з методами об'єкт-орієнтованого програмування. POST /orders <headers> (customer_id, item_id, item_quantity) -> order_id [4].
Формат обміну даними	При використанні REST API структури даних, які передаються між програмними компонентами, зазвичай виражаються у форматі обміну даними JSON. Можна передавати інші формати даних, такі як XML і HTML [4]. JSON легко читається і є гнучким, хоча його необхідно серіалізувати та переводити на мову програмування.	gRPC за замовчуванням використовує формат Protocol Buffers (Protobuf), хоча в ньому також є вбудована підтримка JSON [4]. Сервер визначає структуру даних, використовуючи мову опису інтерфейсу Protocol Buffer (IDL) у файлі протоспецифікації. Потім gRPC серіалізує структуру в двійковий формат та десеріалізує її на будь-яку задану мову програмування. Цей

Продовження таблиці 1.2

Відмінність	REST	gRPC
Взаємозалежність «клієнт-сервер»	REST має слабку взаємозалежність, тобто клієнт і сервер не повинні знати нічого про реалізацію один одного. Така взаємозалежність спрощує модифікацію API з часом, оскільки зміна визначень сервера не обов'язково вимагає зміни коду на стороні клієнта [4].	механізм працює швидше, ніж використання JSON, який не стискається під час передачі. Формат Protocol Buffers не є читабельним для людини, на відміну від JSON, що використовується у REST API [4]. gRPC має сильну взаємозалежність, тобто клієнт і сервер повинні мати доступ до одного і того ж файлу PROTO. Будь-які оновлення файлу потребують оновлень як на стороні сервера, так і на стороні клієнта [4].
Генерація коду	gRPC має вбудований набір функцій для генерації внутрішнього коду як на стороні клієнта, так і на стороні сервера. Ці функції	У той час як REST не має вбудованих механізмів генерації коду. Тому розробники повинні використовувати зовнішні інструменти від

Продовження таблиці 1.2

Відмінність	REST	gRPC
	доступні на кількох мовах завдяки компілятору протоколів Protocol Buffers [4]. Після визначення структури в файлі PROTO система gRPC автоматично генерує код для обох сторін, що робить процес розробки API більш ефективним і менше затратним у часі [4].	сторонніх виробників, якщо вони потребують функціональність автоматичної генерації коду [4].
Двосторонній обмін даними	У REST немає такої функції.	gRPC надає можливість двостороннього потокового обміну даними. Це означає, що як клієнт, так і сервер можуть одночасно надсилати та отримувати кілька запитів і відповідей в межах одного з'єднання [4].

У результаті порівняння, я вирішив використовувати REST у своєму додатку, враховуючи його поєднання простоти та функціональності для забезпечення ефективної взаємодії з вебсервісами.

2 ОСНОВНІ ТЕХНОЛОГІЇ СТВОРЕННЯ ВЕБДОДАТКІВ

2.1 Вибір сучасної бази в цілях збереження даних у вебдодатку

У світі розробки програмного забезпечення вибір правильної бази даних є важливим рішенням, яке значно впливає на продуктивність, масштабованість та зручність використання вебдодатка. З великою кількістю варіантів на вибір, визначити найкращу систему управління базами даних (DBMS), яка ідеально підходить для заданих потреб, може бути складною задачею [5].

Система управління базами даних (DBMS) – це спеціалізоване програмне забезпечення, призначене для зберігання, витягування та обробки даних. Вона виступає як посередник між базою даних, додатками та інтерфейсами користувача для ефективного управління та організації даних. Система надає комплексний набір інструментів для управління базами даних, забезпечуючи безпеку, послідовність та цілісність даних [5].

DBMS підтримує різні застосування, від простих завдань зберігання та витягування даних до складних систем, заснованих на даних, шляхом впровадження ефективних методів доступу та управління даними. Крім того, система може обслуговувати одночасних користувачів, забезпечувати транзакційну послідовність та надійні опції резервного копіювання і відновлення, роблячи її невід'ємною складовою будь-якого середовища, орієнтованого на дані [5].

Існують два типи систем управління базами даних (DBMS): реляційні і нереляційні, які також називають SQL та NoSQL відповідно. Перед обговоренням найпопулярніших варіантів баз даних требо розглянути, в чому відмінності між реляційними та нереляційними системами управління базами даних (табл. 2.1), враховуючи загальноприйняті структури даних, продуктивність, масштабованість та безпеку [5].

Таблиця 2.1 – Порівняння функцій реляційних (SQL) баз даних з нереляційними (NoSQL) базами даних

Функція	Реляційні (SQL) бази даних	Нереляційні (NoSQL) бази даних
Структура даних	Використовують таблиці з рядками та стовпцями для організації даних. Дані мають строго визначену схему, і між таблицями встановлюються зв'язки за допомогою первинних та зовнішніх ключів [6].	Можуть використовувати різні структури даних, такі як документи, ключ-значення, стовпці, графи та інші. Вони зазвичай не мають фіксованої схеми та дозволяють гнучку модель даних [6].
Продуктивність	Відмінно працюють зі складними операціями зчитування та запису на невеликих та середніх обсягах даних. Вони можуть покращити швидкість запитів за допомогою індексів [6].	Зазвичай підходять для завдань з великим обсягом даних та високою паралельністю, таких як аналіз великих даних, IoT та вебзастосунки. Вони зазвичай відзначаються високою продуктивністю та масштабованістю [6].

Продовження таблиці 2.1.

Функція	Реляційні (SQL) бази даних	Нереляційні (NoSQL) бази даних
Масштабованість	Зазвичай масштабуються вертикально, додаючи більше ресурсів (пам'яті, CPU) до існуючого сервера. Масштабування горизонтально (на декілька серверів) може бути складним завданням, оскільки вимагає зміни схеми даних та додаткових зусиль [6].	Зазвичай легше масштабуються горизонтально, додаючи нові сервери або вузли до кластера. Це робить їх більш підходящими для розподіленого зберігання даних [6].
Безпека	Мають вбудовану систему безпеки та доступу, яка може бути керована на рівні таблиць, стовпців та користувачів. Вони підтримують ACID-відповідність, що важливо для додатків, де цілісність даних має велике значення [6].	Мають заходи безпеки, але ці заходи можуть варіюватися в залежності від конкретної системи та конфігурації [6].

PostgreSQL, також відома як Postgres, є безкоштовною та відкритою реляційною системою управління базами даних (RDBMS), яка наголошує на

розширюваності та дотриманні мови SQL. Спочатку вона мала назву POSTGRES, що походило від її походження як наступник до бази даних Ingres, розробленої в Університеті Каліфорнії, Берклі. У 1996 році проект було перейменовано на PostgreSQL, щоб відзначити підтримку мови SQL. Після перегляду в 2007 році команда розробників вирішила залишити назву PostgreSQL та псевдонім Postgres [6].

PostgreSQL має можливості транзакцій з атомарністю, консистентністю, ізоляцією, стійкістю (ACID), автоматично оновлювані види, матеріалізовані види, тригери, зовнішні ключі та збережені процедури. Вона призначена для обробки різноманітних завдань, від одного комп'ютера до складів даних чи вебсервісів з багатьма одночасними користувачами [6].

Переваги PostgreSQL наведено нижче.

– Велика масштабованість: вертикальна масштабованість є відзнакою PostgreSQL. Враховуючи те, що практично будь-яке індивідуальне програмне рішення тенденційно росте і призводить до розширення бази даних, ця опція підтримує розвиток бізнесу [6].

– Підтримка користувацьких типів даних: PostgreSQL нативно підтримує багато типів даних за замовчуванням, таких як JSON, XML, H-Store та інші. PostgreSQL використовує це, бувши однією з небагатьох реляційних баз даних з міцною підтримкою функцій NoSQL. Крім того, вона дозволяє користувачам визначати свої власні типи даних. Оскільки бізнес-модель програмного забезпечення може потребувати різних типів баз даних протягом всього існування для покращення продуктивності чи всебічності застосунку, ця опція принесе покращену гнучкість [6].

– Легко інтегровані сторонні інструменти: система управління базами даних PostgreSQL має сильну підтримку додаткових інструментів, як безкоштовних, так і комерційних. Діапазон таких інструментів включає розширення для покращення багатьох аспектів. Наприклад, ClusterControl надає

вражаючу допомогу в управлінні, моніторингу та масштабуванні відкритих SQL та NoSQL баз даних [6].

– Відкритий код та підтримка спільноти: Postgres є повністю відкритим та підтримується своєю спільнотою, що посилює його як повну екосистему. Крім того, розробники завжди можуть очікувати безкоштовної та оперативної підтримки спільноти [6].

Недолікі PostgreSQL наведено нижче.

– Непослідовна документація: незважаючи на велику спільноту PostgreSQL та сильну підтримку її учасників, документація все ще не завжди має послідовність та повноту. Оскільки спільнота PostgreSQL досить розподілена, документація не завжди відповідає єдиності стандартів для всіх функцій PostgreSQL [6].

– Відсутність інструментів для звітування та ревізії: суттєвим недоліком PostgreSQL є відсутність інструментів для візуалізації стану бази даних. Вам доведеться постійно перевіряти, чи щось пішло не так. Існує завжди ризик того, що інженери баз даних помітять неполадки занадто пізно [6].

Завдяки складним запитам та широкому вибору користувацьких інтерфейсів, які виконуються за допомогою попередньо визначених функцій, PostgreSQL ідеально підходить для аналізу та зберігання даних. Якщо розробляється інструмент автоматизації баз даних, то PostgreSQL є найкращим вибором завдяки своїм потужним аналітичним можливостям, відповідності ACID та потужному SQL-двигуну. Завдяки цим плюсам СУБД набула свою популярність серед фінансових установ та телекомунікаційних систем [6].

2.2 Вибір сучасної мови програмування в цілях веброзробки

Зараз існують кілька мов програмування для створення вебсайтів. Python і Golang – це дві з передових мов для веброзробки. З серед багатьох

загальновживаних мов програмування розробники часто стикаються з інтенсивним протистоянням Golang проти Python.

Ці дві мови помітно різняться. Python був розроблений у 1991 році, що є старожилом у світі програмування, в той час як Go був опублікований вперше в 2012 році Google, відносно новачок серед своїх колег. Програмісти в Google запустили Golang, щоб прискорити продуктивність і виправити помилки, які виникали при розробці іншими мовами програмування. Golang є мовою, розробленою більш молодшою, ніж Python. Golang відомий своєю швидкістю, тоді як Python вважається повільною мовою. Python є універсальним, в той час як Go має більш жорсткий синтаксис і форматування. Проте обидві мови вважаються відповідями на багато традиційних запитань щодо програмування [7].

Python – це загального призначення, високорівнева і дуже популярна мова програмування. Мову програмування Python використовують у застосуваннях машинного навчання, науці про дані, розробці вебсайтів (рис. 2.1), а також у всій сучасній технології в галузі програмного забезпечення. Python має простий для вивчення синтаксис, який поліпшує зрозумілість коду і зменшує витрати на обслуговування програм. Python підтримує пакети і модулі, що сприяє модульності програми і використанню коду повторно. Інтерпретатор Python і велика стандартна бібліотека доступні у вигляді вихідного коду або бінарного коду безкоштовно для всіх значущих платформ і можуть бути вільно поширювані [7].

Приклад, створення веб додатку на мові Python наведено на рис.2.1.

Переваг мови Python наведені нижче.

– Легкість програмування: Python – це високорівнева мова програмування. Python можна вивчати легше, ніж інші мови програмування. Програмування на Python також дуже просте, і будь-яка людина може вивчити основи Python за кілька годин або днів. Це також дуже дружелюбна для програмістів мова [7].

```python
from flask import json, Blueprint, Response, request
from flask_login import login_required
from pydantic import ValidationError

from src.domain import films_dom
from src.exception import films_exc


film_blueprint = Blueprint('film_blueprint', __name__, url_prefix='/api/v1')


@film_blueprint.route('/ping')
def ping():
    return Response(
        response=json.dumps({'msg': 'pong'}),
        status=200,
        mimetype='application/json'
    )
```

Рисунок 2.1 – Приклад HealthChek ендпоінту на мові Python

– Безкоштовний та відкритий код: мову програмування Python можна завантажити з офіційного вебсайту, клікнувши на ключове слово "Завантажити Python". Оскільки вона є відкритою для громадськості, це означає, що вихідний код також доступний публіці [7].

– Об'єктно-орієнтована мова: об'єктно-орієнтоване програмування є однією з основних особливостей Python. Python підтримує концепції класів, об'єктів, інкапсуляції та інші принципи об'єктно-орієнтованої мови.

– Підтримка розробки графічного інтерфейсу користувача (GUI): за допомогою модулів, таких як PyQt5, PyQt4, wxPython або Tk у Python, можна створювати графічний інтерфейс користувача (GUI). PyQt5 є найпоширенішим варіантом для створення графічних програм на Python [7].

– Високорівнева та інтерпретована мова: Python – це високорівнева мова. При формулюванні програм на Python не потрібно пам'ятати архітектуру системи, і не потрібно керувати пам'яттю. Python вважається інтерпретованою мовою, оскільки код Python виконується рядок за рядком. Подібно до інших

мов програмування, не потрібно компілювати код Python, що робить його легким для налагодження. Вихідний код Python перетворюється в проміжну форму, яку називають байт-кодом [7].

– Можливість розширення та переносима мова: Python – це розширювана мова. Існує можливість написати деякий код на Python, а потім перетворити його у мову C або C++, і навіть скомпілювати цей код у мові C/C++. Python також є дуже переносною мовою. Якщо є код Python для Windows і треба запустити його на платформах, таких як Unix, Linux і Mac, то не потрібно його змінювати [7].

На відміну від Python, мова Golang - це процедурна, компільована та статично типізована мова програмування, розроблена Google. Вона була розроблена в 2007 році Кеном Томпсоном, Робертом Гріземером і Робом Пайком в Google, але була запущена у 2009 році як відкрита мова програмування. Програми збираються за допомогою пакетів для ефективного управління залежностями. Golang також дозволяє використовувати патерни, схожі на динамічні мови, наприклад, виведення типу [7].

Початково Go була розроблена для програм, пов'язаних з мережами та інфраструктурою. Вона була створена для заміни традиційних мов програмування на стороні сервера з високою продуктивністю, таких як C++ та Java. Зараз Go використовується для різних видів застосувань:

– Go є найбільш популярною для хмарних або серверних додатків;

– DevOps і автоматизація надійності сайтів також рекомендовані способи роботи з Golang;

– багато командних інструментів написані на Go;

– Go використовується в світі науки про дані та штучного інтелекту;

– багато розробників використовують Golang для програмування мікроконтролерів, створення ігор та робототехніки.

Проте Go дійсно популярний, коли мова йде про інфраструктуру. Кілька найпоширеніших інструментів для інфраструктури написані на Go, включаючи Kubernetes, Prometheus і Docker.

Переваги мови Go наведені нижче.

– Простота: Golang встановлює надійність, зрозумілість та підтримку як свої основні пріоритети і не конкурує, щоб довести свою багаторічність. Розробники Golang включають в мову лише ті характеристики, які є важливими, і не роблять мову складною, додавши безліч речей. Крім того, якщо перевіряється код Golang іншої людиною, незалежно від розміру коду, кожен рядок буде дуже простим і зрозумілим [7].

– Потужна стандартна бібліотека: Golang має потужний набір бібліотек, що спрощує написання власного коду. Хоча його бібліотека не така різноманітна, як у Python або Java, вона містить всі основні компоненти [7].

– Конкурентність в Golang: Іншою неймовірною функцією, яка зробила Golang популярним, є його можливість конкурентності. Golang пропонує Goroutines і канали для роботи з конкурентністю. Конкурентність ефективно використовує архітектуру багатоядерних процесорів. Конкурентність також допомагає більш надійному масштабуванню великих додатків. Деякі відомі проекти, написані на Go, включають Docker, Hugo, Kubernetes і Dropbox [7].

– Створення вебдодатків (рис.2.2): Golang привертає увагу як мова для створення вебдодатків завдяки своїм простим конструкціям і швидкості виконання.

– Підтримка тестування: Golang пропонує спосіб тестувати пакет. За допомогою команди "go test" можно тестувати код, написаний у файлах "*_test.go". Для забезпечення стабільності програми тестування є необхідністю додати тестову функцію разом із фактичною функцією кожного разу, коли створюєте який-небудь код [7].

– Швидкість компіляції: Golang і завершення є набагато стійкішою, ніж у багатьох відомих мов програмування. Golang можна легко розбирати без таблиці символів.

```go
package server

import (
        "net/http"

        "github.com/Drozd0f/csv-app/schemes"
        "github.com/gin-gonic/gin"
)

// ping godoc
// @Summary show pong
// @Tags    Healthcheck
// @Produce json
// @Success 200 {object} schemes.Response "Server is alive"
// @Router  /ping [get]
func (s *Server) ping(c *gin.Context) {
        c.JSON(http.StatusOK, schemes.Response{
                Message: "pong",
        })
}
```

Рисунок 2.2 – Приклад HealthChek ендпоінту на мові Golang

Можна говорити безкінечно про переваги та недоліки Golang або Python (табл. 2.2), але коли мова йде про реальну продуктивність стає питання наскільки повільно вони виконуються. Таким чином, був проведені деякі тести на тестовій системі [8].

Таблиця 2.2 – Порівняння швидкостей мов Golang та Python

Назва тесту	Швидкість мови Golang	Швидкість мови Python
Бінарний пошук	20.8 ns/op (наносекунд на операцію)	2442.13377 ns/op (наносекунд на операцію)
Bubble сортування	90805247 ns/op	6708160950.6 ns/op
Читання з файлу	5305 ns/op	58359 ns/op
Обробка HTTP-запиту	0.070 ms/RQ (мілісекунд на запит)	1.261 ms/RQ (мілісекунд на запит)

Хоча Python залишився улюбленою мовою спільноти, зберігши друге місце в першому кварталі 2019 року серед найшвидших мов програмування на GitHub за кількістю запитів на злиття (+17%), Golang не відстає далеко і тримає четверту позицію (+8%). Вибір між Golang та Python стає ще більш розмитим. Незалежно від цього, є кілька речей, які варто врахувати під час вибору, яка мова може бути правильним вибором для веброзробки [8].

– Масштабованість: Golang був створений з урахуванням масштабованості. Він має вбудовану конкурентність для обробки кількох завдань одночасно. Python використовує конкурентність, але це не вбудована функція; він реалізує паралелізм через потоки. Це означає, що якщо планується працювати з великими наборами даних, то Golang є більш підходящим вибором [8].

– Продуктивність: Python не відомий як мова, що ефективно використовує пам'ять або процесор, але завдяки великій кількості бібліотек, Python ефективно працює для базових завдань розробки. Golang має вбудовані функції і більше підходить для архітектур програмного забезпечення мікросервісів [8].

– Застосування: Python видається блискучим, коли використовується для написання коду для штучного інтелекту, аналізу даних, глибокого навчання та

веброзробки. У той час як Golang використовується для системного програмування і є улюбленою мовою розробників, які використовують її для обчислювання в хмарних обчисленнях та обчисленнях у кластері [8].

– Спільнота та бібліотеки: як вже зазначалося, старість Python надає йому певні переваги. Однією з них є кількість бібліотек і велика спільнота, яка його підтримує. Golang, з іншого боку, є все ще ростучою мовою і не має кількості бібліотек та підтримки спільноти, якою користується Python. Проте не варто ще відраховувати Go. Його темпи росту та прийняття неймовірні, і він розширюється щодня [8].

– Виконання: якщо швидкість - це головне, то Golang виграє з великим відривом [8].

Після врахування всіх цих факторів сфера застосування буде визначальним фактором у виборі мови для використання. У випадку, коли э потреба у створенні мікросервісів, Golang був би більш розумним вибором, оскільки він швидкий, легкий для програмування і добре масштабується. Python, з іншого боку, більше спрямований на штучний інтелект, машинне навчання та аналіз даних. Отже, порівнюючи один з одним, в більшості випадків Go виходить вперед і вважається валідною альтернативою Python [8].

2.3 Програмне забезпечення Docker в цілях веброзробки

Docker – це відкрите програмне забезпечення, яке дозволяє розробникам створювати, розгортати та керувати додатками в різних обчислювальних середовищах. Він надає систему віртуалізації на основі контейнерів (табл. 2.3), яка дозволяє розробникам упаковувати свої додатки в ізольовані образи, які потім можна розгортати на будь-якій операційній системі чи хмарній платформі. За допомогою Docker розробники можуть швидко і легко створювати, тестувати та розгортати додатки, не переймаючись питаннями сумісності чи апаратними вимогами.

Таблиця 2.3 – Різниця між контейнерами Docker і віртуальними машинами

Контейнери Docker	Віртуальні машини
Контейнери Docker містять бінарні файли, бібліотеки та файли конфігурації разом із самим додатком [9].	Віртуальні машини (ВМ) працюють на гіпервізорах, які дозволяють запускати кілька віртуальних машин на одному пристрої разом із власною операційною системою [9].
У них немає гостьової операційної системи для кожного контейнера і вони покладаються на ядро операційної системи, на якій вони працюють, що робить контейнери легкими [9].	Кожна ВМ має свою власну копію операційної системи разом із додатком і необхідними бінарними файлами, що робить її значно більшою та вимагає більше ресурсів [9].
Контейнери ділять ресурси з іншими контейнерами на тій же операційній системі хоста та забезпечують ізоляцію процесів на рівні операційної системи [9].	Вони забезпечують ізоляцію процесів на рівні апаратного забезпечення та повільно запускаються [9].

Важливі терміни в Docker наведені нижче.

– Docker Image (образ Docker) – це легкий, автономний та виконуваний пакет, який містить все необхідне для запуску додатка, включаючи код, середовище виконання, бібліотеки та системні інструменти. Він забезпечує послідовну та відтворювану платформу для розгортання додатків незалежно від базової інфраструктури чи операційної системи хоста. Образи Docker створюються на основі базового образу за допомогою набору інструкцій, які називають Dockerfile і вказують, як створювати образ крок за кроком [10].

– Docker Container (контейнер Docker) – це ізольоване середовище для коду. Це означає, що контейнер не має знання про операційну систему чи файли. Він працює в середовищі, яке надається Docker Desktop. Тому контейнер зазвичай має все, що потрібно коду для запуску, включаючи базову операційну систему. Можно використовувати Docker Desktop для управління та дослідження контейнерів [10].

– Dockerfile (рис.2.3) – Docker може автоматично створювати образи, читаючи інструкції з Dockerfile. Dockerfile – це текстовий документ, який містить всі команди, які користувач може викликати в командному рядку для збірки образу [10].

До найбільш часто використовуваних тегів в Dockerfile відносяться:

– FROM: вказує базовий образ для побудови образу;

– RUN: виконує команду під час процесу побудови;

– CMD: вказує команду за замовчуванням для запуску контейнера;

– ENV: встановлює змінну середовища в контейнері;

– COPY: копіює файли або каталоги з локальної машини тільки в контейнер;

– ADD: додає функціональність копіювання і дозволяє використовувати URL замість локального файлу/каталогу та розпаковувати tar з джерела до призначення;

– EXPOSE: відкриває конкретний порт чи порти для використання контейнером;

– LABEL: додає метадані до образу у формі пар "ключ-значення";

– USER: вказує користувача, який використовуватиметься при запуску контейнера;

– WORKDIR: як і сама назва вказує, встановлює робочий каталог для контейнера.

```
1   FROM golang:1.19.2 as base
2   WORKDIR /app
3   COPY . .
4   RUN CGO_ENABLED=0 GOOS=linux go build -o /ttto ./cmd/monolith
5
6
7   FROM alpine:3.16.2 as prod
8
9   COPY --from=base /ttto /usr/bin/ttto
10  CMD ["sh", "-c", "ttto migrate && ttto run"]
```

Рисунок 2.3 – Приклад Dockerfile

Docker Compose (рис. 2.4 – 2.5) – це інструмент для визначення та запуску багатоконтейнерних додатків Docker. Він використовує файли YAML для налаштування послуг додатка та виконує процес створення та запуску всіх контейнерів однією командою. Утиліта docker-compose CLI дозволяє користувачам виконувати команди на декількох контейнерах одночасно, наприклад, будувати образи, масштабувати контейнери, запускати зупинені контейнери та інше. Команди, пов'язані з операціями над образами або інтерактивними опціями користувача, не є важливими в Docker Compose, оскільки вони стосуються окремого контейнера. Файл docker-compose.yml використовується для визначення послуг додатка і включає різноманітні параметри конфігурації. Наприклад, опція build визначає параметри конфігурації, такі як шлях до Dockerfile, опція command дозволяє перевизначити типові команди Docker та інше [11].

```yaml
version: '2.1'

services:
  app:
    restart: always
    build:
      context: .
      dockerfile: Dockerfile
    depends_on:
      db:
        condition: service_healthy
  db:
    image: postgres:latest
    restart: always
    healthcheck:
      interval: 5s
      timeout: 5s
      retries: 10
```

Рисунок 2.4 – Приклад базового Docker Compose файлу

```yaml
services:
  nginx:
    image: nginx:1.22
    ports:
      - '80:80'
    depends_on:
      - app
    volumes:
      - ./nginx:/etc/nginx/conf.d

  app:
    command: bash -c "python3 /film_library/run.py"
    build:
      target: base
    volumes:
      - ./film_library:/film_library
    env_file:
      - .env
    environment:
      PYTHONUNBUFFERED: 1
      DEV_DB_URI: postgresql://${POSTGRES_USER}:${POSTGRES_PASSWORD}@db:5432/${POSTGRES_DB}

  db:
    env_file:
      - .env
    volumes:
      - db:/var/lib/postgresql/data
    healthcheck:
      test: [ "CMD-SHELL", "pg_isready -U $POSTGRES_USER" ]

  swagger:
    image: swaggerapi/swagger-ui:v4.12.0
    volumes:
      - ./docs/swagger_doc.json:/app/swagger.json
    environment:
      -  BASE_URL=/api-docs

volumes:
  db:
```

Рисунок 2.5 – Приклад Docker Compose файлу для локальної розробки

Docker Daemon (фоновий сервіс Docker) – фоновий сервіс, який працює на хості і відповідає за збірку, запуск та розподіл контейнерів Docker. Демон - це процес, який працює в операційній системі і з яким взаємодіють клієнти [11].

Docker Client (клієнт Docker) – командний рядок, який дозволяє користувачеві взаємодіяти з демоном [11].

Docker Hub (реєстр Docker) – каталог усіх доступних Docker-образів. Реєстри Docker можуть бути використані для завантаження/вивантаження образів [11].

Docker є однією з компонентів технології – Kubernetes. Kubernetes – це система керування контейнерами, розрозроблена для автоматизації процесів розгортання та управління контейнеризованими додатками на багатьох серверах одночасно. Вона надає інструменти для масштабування, забезпечує високу доступність додатків та спрощує керування великими кластерами контейнерів. Kubernetes вирішує складні завдання оркестрації та автоматизації, допомагаючи забезпечити стабільну та надійну роботу додатків [11].

Основні компоненти Kubernetes наведені нижче.

– Майстер-вузол (Master Node): центральна частина системи, яка керує всіма операціями в кластері Kubernetes. Майстер-вузол включає в себе сервер керування API (API Server), контролери (Controllers) та реєстр конфігурації (etcd), який зберігає інформацію про конфігурацію кластера [12].

– Вузли (Nodes): фізичні або віртуальні сервери, на яких запускаються контейнери. Вони становлять робочу частину кластера і виконують завдання, надані майстер-вузлом [12].

– Поди (Pods): найменша одиниця роботи в Kubernetes, що містить один або кілька контейнерів. Піди групують контейнери, які повинні працювати разом, і надають їм спільні ресурси та мережевий стек [12].

– Служби (Services): служби дозволяють надавати доступ до групи підів, як одному додатку. Вони автоматично керують балансуванням навантаження, щоб забезпечити доступність додатків [12].

– Розпорядники (Controllers): ці компоненти відповідають за стан додатків у кластері. Наприклад, контролери реплікації дозволяють забезпечити певну кількість однакових копій додатків для надійності [12].

3 РОЗРОБКА БІЛІНГ ДОДАТКУ НА MOBI GO

3.1 Опис функціоналу додатку та його області застосування

На сьогоднішній день дуже гостро стоїть проблема обробки сирої інформації, особливо у контексті фінансових транзакцій та оплати покупок.

Сучасний білінг нерідко стикається з великим обсягом неструктурованих даних, що виникають у процесі фінансових операцій та транзакцій оплати. Обробка цих сирих даних стає важливим завданням, оскільки вони можуть бути різнорідні, погано структуровані та потребують систематизації для подальшого аналізу та використання.

З урахуванням цієї необхідності, розроблений білінговий додаток вирішує важливі завдання зі збору, обробки та аналізу даних.

Додаток не лише визначає цю проблему, але й пропонує рішення для ефективного управління сирою інформацією. Він створений з урахуванням потреб підприємств у вивченні та використанні фінансових даних для прийняття стратегічних рішень. Додаток демонструє необхідність та актуальність інструментів, які спрощують обробку великих обсягів даних та забезпечують їхню структурування для подальшого використання в аналізі та стратегічному плануванні.

Додаток має наступні ключові функціональні можливості, які визначають його як ефективний інструмент у сфері оптимізації роботи з фінансовою інформацією.

Додаток володіє вражаючою здатністю ефективно взаємодіяти з CSV-файлами, де знаходяться транзакції оплати покупок. Механізм парсингу автоматично видобуває інформацію з цих файлів та зберігає її в базі даних. Це вирішує проблему неструктурованих даних та підготовки їх для подальшого використання.

У додатку реалізовано точка, яка надає можливість отримати деталі про транзакції з бази даних за допомогою фільтрів наведених нижче.

– transaction_id – ідентифікатор певної транзакції для точного пошуку.

– terminal_id – список ідентифікаторів терміналів для пошуку транзакцій, які проводились на певних терміналах.

– status – статус транзакції (accepted/declined).

– payment_type – тип оплати транзакції (card/cash).

– from – дата початку транзакції.

– to – дата завершення транзакції.

– payment_narrative – розрахунок або перерахування коштів згідно з умовами договору про надання послуг.

Додаток забезпечує можливість експорту даних з бази у форматі CSV. Це є важливим для забезпечення зручного обміну інформацією між системами та використання даних у стандартному форматі.

Розглядаючи функції додатку, стає зрозумілим, що він не просто вирішує технічні проблеми обробки даних, але також відповідає на виклики, які стоять перед компаніями в галузі фінансів. Від парсингу та збереження даних до надання зручного інтерфейсу для аналізу, додаток стає невід'ємною частиною стратегічного управління інформацією в організаціях.

3.2 Реалізація точки завантаження CSV файлу у додаток

Для реалізації точки завантаження CSV файлу, у додатку, було прийнято рішення всю логіку по опрацюванню сирих даних зосередити в сервісному методі (рис.3.1). Код представляє метод UploadCsvFile, вбудований у структуру Service. Цей метод призначений для обробки CSV-файлів та наступного завантаження даних у сховище. Процес завантаження даних включає кілька етапів, починаючи з відкриття файлу та завершуючи збереженням структурованих даних у сховище за допомогою вказаних схем і відображень.

```go
func (s *Service) UploadCsvFile(ctx context.Context, f IFileHeader) error {  5 usages
    file, err := f.Open()
    if err != nil : ErrOpenFile ⤴
    defer file.Close()

    r := csv.NewReader(file)
    rowRecords, err := r.ReadAll()
    if err != nil : fmt.Errorf("csv read all: %w", err) ⤴

    headers := rowRecords[0]

    records := make([]schemes.Transaction, len(rowRecords)-1)
    for idx, rowRecord := range rowRecords {
        if idx != 0 {
            var t schemes.Transaction
            if err = schemes.BindFromCsv(&t, rowRecord, headers); err != nil {
                return fmt.Errorf("schemes bind from csv: #{err}")
            }

            records[idx-1] = t
        }
    }

    if err = s.r.CreateTransactions(ctx, records); err != nil : fmt.Errorf("repository create transactions: %w", err) ⤴

    return nil
}
```

Рисунок 3.1 – Реалізація сервісного методу для завантаження CSV файлу

Діаграма реалізації сервісного методу для завантаження CSV файлу представлена в дод. Б, рис. Б.1.

Метод UploadCsvFile розділяється на ключові етапи які наведені нижче.

– Відкриття файлу – код починається з відкриття CSV-файлу, який передається через інтерфейс IFileHeader. Якщо операція відкриття завершується успішно, файл буде автоматично закритий після виконання функції за допомогою конструкції defer.

– Читання CSV-даних – Після відкриття файлу створюється читач CSV, і всі рядки файлу зчитуються за допомогою ReadAll. У випадку виникнення помилки під час читання даних, повертається відповідна помилка.

– Витягування заголовків – перший рядок CSV-файлу зчитується як заголовок, припускаючи, що він містить назви полів для подальшого прив'язування даних.

– Перетворення CSV-рядків у структури – всі інші рядки CSV-файлу перетворюються у структури schemes. Transaction за допомогою функції BindFromCsv. Ці структури зберігаються у слайс для подальшого використання.

– Збереження даних у сховище – отримані структури передаються методу CreateTransactions інтерфейса Repository, відповідального за збереження даних у сховище. У випадку помилки на цьому етапі, метод повертає помилку з докладним описом причини невдачі.

– Повернення результату – на завершення метод повертає nil, якщо процес завантаження даних успішно завершено. У випадку помилки повертається відповідна помилка з детальним описом причини збою й наступного логування.

3.3 Реалізація точки отримання транзакцій з урахуванням фільтрів

Для реалізації точки отримання транзакцій з урахуванням фільтрації, у додатку було прийнято рішення створити зручний механізм фільтрації, що базується на внутрішніх структурах та функціях для опрацювання вхідних даних. Всі ці елементи об'єднуються в методі GetFilteredTransactions (рис. 3.2) сервісу. Діаграма реалізації сервісного методу для отримання відфільтрованих транзакцій представлена в дод. Б, рис. Б.2.

Цей метод надає прозорий інтерфейс для взаємодії з фільтрованими транзакціями. Його реалізація дозволяє зручно використовувати структури та функції для перетворення та фільтрації даних перед передачею їх до репозиторію. Додатково, він забезпечує обробку помилок та конвертацію отриманих результатів для зручної взаємодії з рівнем вищого рівня додатку.

Цей підхід дозволяє зробити код більш читабельним та підтримуваним. Операції конвертації та фільтрації винесені у відокремлені функції для полегшення їх розширення або зміни в майбутньому. Також, зручний механізм

обробки помилок допомагає підтримувати надійність програми під час взаємодії з репозиторієм.

Такий підхід до реалізації фільтрації транзакцій сприяє створенню ефективного та розширюваного механізму, який може бути легко адаптований до змін у вимогах програми чи розширення функціональності.

```go
func (s *Service) GetFilteredTransactions(ctx context.Context, rrt schemes.RawTransactionFilter) (schemes.SliceTransactions, error) {
    storedSliceT, err := s.r.GetSliceTransactions(ctx, schemes.NewTransactionFilterFromRaw(rrt))
    if err != nil {
        return schemes.SliceTransactions{}, fmt.Errorf( format: "repository get slice transactions: %w", err)
    }

    return schemes.NewSliceTransactionsFromDB(storedSliceT), nil
}
```

Рисунок 3.2 – Реалізація сервісного методу для отримання відфільтрованих транзакцій

Для зручності фільтрації було прийнято рішення створити структуру фільтра та надати можливість зручного перетворення вхідних даних з необхідного формату. Розглянемо функцію NewTransactionFilterFromRaw (рис. 3.3), яка створює екземпляр структури TransactionFilter з вхідних даних типу RawTransactionFilter.

```go
73  func NewTransactionFilterFromRaw(r RawTransactionFilter) TransactionFilter {
74      return TransactionFilter{
75          ID:              stringToInt32(r.ID),
76          Status:          stringToExpected(r.Status, Statuses),
77          TerminalIDs:     sliceStringToInt32(r.TerminalIDs),
78          PaymentType:     stringToExpected(r.PaymentType, PaymentTypes),
79          DatePost:        newDatePostFromRaw(r.DatePost),
80          PaymentNarrative: r.PaymentNarrative,
81      }
82  }
```

Рисунок 3.3 – Реалізація функції для створення фільтру транзакцій

Для зручності конвертації були написані наступні функції: stringToInt32 (рис. 3.4) для перетворення рядка у 32-бітне ціле число, sliceStringToInt32 (рис. 3.5) для конвертації зрізу рядків у зріз 32-бітних цілих чисел, stringToExpected (рис. 3.6) для визначення, чи рядок належить до масиву очікуваних значень, та newDatePostFromRaw (рис. 3.7) для створення об'єкта дати з вхідних даних у форматі RawDatePost. Ці функції є ключовими для підтримки ефективного та структурованого підходу до обробки даних при реалізації точки отримання фільтрованих транзакцій у додатку.

```
func stringToInt32(s string) int32 {  2 usages
    // Ignore error and return 0 as default value
    intS, _ := strconv.ParseInt(s,  base: 10,  bitSize: 32)
    return int32(intS)
}
```

Рисунок 3.4 – Реалізація функції для конвертації строки у цифрове значення

Функція stringToInt32 використовується для конвертації рядка у 32-бітне ціле число. Вона ігнорує помилки під час конвертації та повертає значення за замовчуванням (у цьому випадку, 0), що може бути корисним, коли точне значення не критичне.

```
func sliceStringToInt32(sliceS []string) []int32 {  1 usage
    sliceI := make([]int32, 0, len(sliceS))

    for _, s := range sliceS {
        sliceI = append(sliceI, stringToInt32(s))
    }

    return sliceI
}
```

Рисунок 3.5 – Реалізація функції для конвертації списку строк у список цифрових значень

Функція sliceStringToInt32 призначена для конвертації кожного елемента у зрізі рядків у відповідне 32-бітне ціле число. Вона використовує stringToInt32 для конвертації кожного рядка та повертає зріз 32-бітних цілих чисел.

```
func stringToExpected(s string, expectedStrings [2]string) string {  2 usages
    for _, str := range expectedStrings {
        if s == str : s
    }

    return DefaultString
}
```

Рисунок 3.6 – Реалізація функції для валідації строки

Функція stringToExpected перевіряє, чи вхідний рядок s належить до масиву очікуваних рядків expectedStrings. Якщо так, функція повертає сам рядок s. В іншому випадку, вона повертає значення за замовчуванням (DefaultString). Це корисно для забезпечення коректності та зручності в роботі з вхідними даними.

```
func newDatePostFromRaw(dpr RawDatePost) DatePost {  1 usage
    layout := "2006-01-02"
    timeFrom, err := time.Parse(layout, dpr.From)
    timeTo, err := time.Parse(layout, dpr.To)
    if err != nil {
        timeTo = time.Now()
    }
    return DatePost{
        From: timeFrom,
        To:   timeTo,
    }
}
```

Рисунок 3.7 – Реалізація функції для обробки часового діапазону

Функція newDatePostFromRaw використовується для створення об'єкта DatePost із вхідної структури RawDatePost. Вона конвертує рядкові представлення дат From і To в об'єкти time.Time. У разі помилки під час конвертації To, використовується поточна дата та час. Отримані значення використовуються для ініціалізації об'єкта DatePost.

Метод GetFilteredTransactions розділяється на ключові етапи, які наведені нижче.

– Отримання відфільтрованих транзакцій – використовується репозиторій s.r, в якому викликається метод GetSliceTransactions. Для фільтрації транзакцій передається об'єкт структури TransactionFilter, який створюється з вхідних даних RawTransactionFilter за допомогою schemes.NewTransactionFilterFromRaw(rrt).

– Обробка помилок при отриманні транзакцій – перевіряється наявність помилки під час виклику методу репозиторію. У разі наявності помилки, метод повертає порожній об'єкт типу SliceTransactions та створює помилку з додатковим повідомленням про виникнену помилку при отриманні транзакцій.

– Конвертація та повернення результату – отриманий результат, який є зрізом транзакцій із репозиторію, конвертується за допомогою schemes.NewSliceTransactionsFromDB у формат SliceTransactions та повертається як результат методу. У разі успішного виклику методу репозиторію, метод повертає відфільтрований зріз транзакцій та nil як помилку.

3.4 Реалізація точки вивантаження транзакцій у вигляді CSV файлу

Для реалізації альтернативної точки отримання транзакцій, у додатку було прийнято рішення також створити можливість завантаження цих транзакцій у форматі CSV. Цей механізм завантаження в CSV є корисним для зручності обміну даними та подальшого аналізу за допомогою зовнішніх інструментів.

Метод DownloadCsvFile (рис. 3.8) виконує ключову функцію у цьому контексті, дозволяючи отримувати та експортувати транзакції у форматі, зрозумілому для інших систем або для подальшого аналізу даних. Цей підхід забезпечує гнучкість та доступність інформації про транзакції, що дозволяє користувачам ефективно використовувати дані та допомагає покращити взаємодію з додатком в цілому.

```
func (s *Service) DownloadCsvFile(ctx context.Context, w io.Writer) error {
    p := schemes.Paginator{
        Page:  1,
        Limit: s.c.ChunkSize,
    }
    writer := csv.NewWriter(w)
    isHeaderWrote := false

    for {
        storTrans, err := s.r.GetTransactions(ctx, p)
        if err != nil { return err }

        st := schemes.NewSliceTransactionsFromDB(storTrans)
        if !isHeaderWrote {
            if err = writer.Write(st.GetCsvNames()); err != nil { return err }
            isHeaderWrote = true
        }

        if err = writer.WriteAll(st.ToString()); err != nil { return err }

        if int32(len(storTrans)) < p.Limit { return nil }

        p.Page++
    }
}
```

Рисунок 3.8 – Реалізація сервісного методу для вивантаження транзакцій з додатку

Діаграма реалізації сервісного методу для вивантаження транзакцій з додатку представлена в дод. Б, рис. Б.3.

Метод DownloadCsvFile розділяється на ключові етапи, які наведені нижче.

– Ініціалізація пагінатора та об'єкта для запису CSV – створюється пагінатор для отримання транзакцій порціями, а також ініціалізується об'єкт для запису CSV з використанням переданого io.Writer.

– Отримання порції транзакцій та запис даних у вихідний потік – викликається метод репозиторію для отримання порції транзакцій з використанням пагінатора. Отримані дані конвертуються у внутрішній формат SliceTransactions для подальшого запису у CSV.

– Запис заголовків та даних у вихідний потік – записує заголовки колонок у вихідний потік, якщо це не було зроблено раніше. Після цього записує дані транзакцій у вихідний потік у форматі CSV.

– Перевірка умови завершення та збільшення номеру сторінки – перевіряється, чи отримано менше транзакцій, ніж обмежено пагінатором. Якщо умова виконується, метод завершується. В іншому випадку збільшується номер сторінки для отримання наступної порції транзакцій.

3.5 Оптимізація точки завантаження CSV файлу у додаток

Пам'ять, яку використовує додаток при обробці та збереженні обширних файлів, є ключовим аспектом ефективності. В ідеалі, бажано уникати зайвого збільшення використання пам'яті, особливо при роботі з великими обсягами даних. З плином часу та збільшенням розміру файлів пам'ять, яку програма споживає, може стати проблемою.

Один з основних негативних ефектів полягає в тому, що при збільшенні розміру оброблюваних файлів збільшується і використана програмою пам'ять. Це може впливати на продуктивність та швидкодію додатку, особливо на пристроях з обмеженими ресурсами.

Оптимізація пам'яті стає важливим аспектом для покращення продуктивності та забезпечення стабільної роботи додатку. Для зменшення використання пам'яті були розглянуті та впроваджені стратегії та техніки під час обробки та збереження даних. Це включає у себе ефективне управління ресурсами та оптимізацію алгоритмів для забезпечення ефективності роботи додатку, навіть при опрацюванні великих обсягів інформації.

При дослідженні ефективності додатку важливим аспектом було – оптимізація використання пам'яті. Для вивчення цієї проблеми були зібрані метрики використання пам'яті під час завантаження файлів різного розміру. З отриманих даних випливає, що розмір використовуваної пам'яті зростає геометрично (рис.3.9) в залежності від розміру файлу. Це свідчить про необхідність впровадження оптимізаційних заходів для покращення продуктивності додатка.

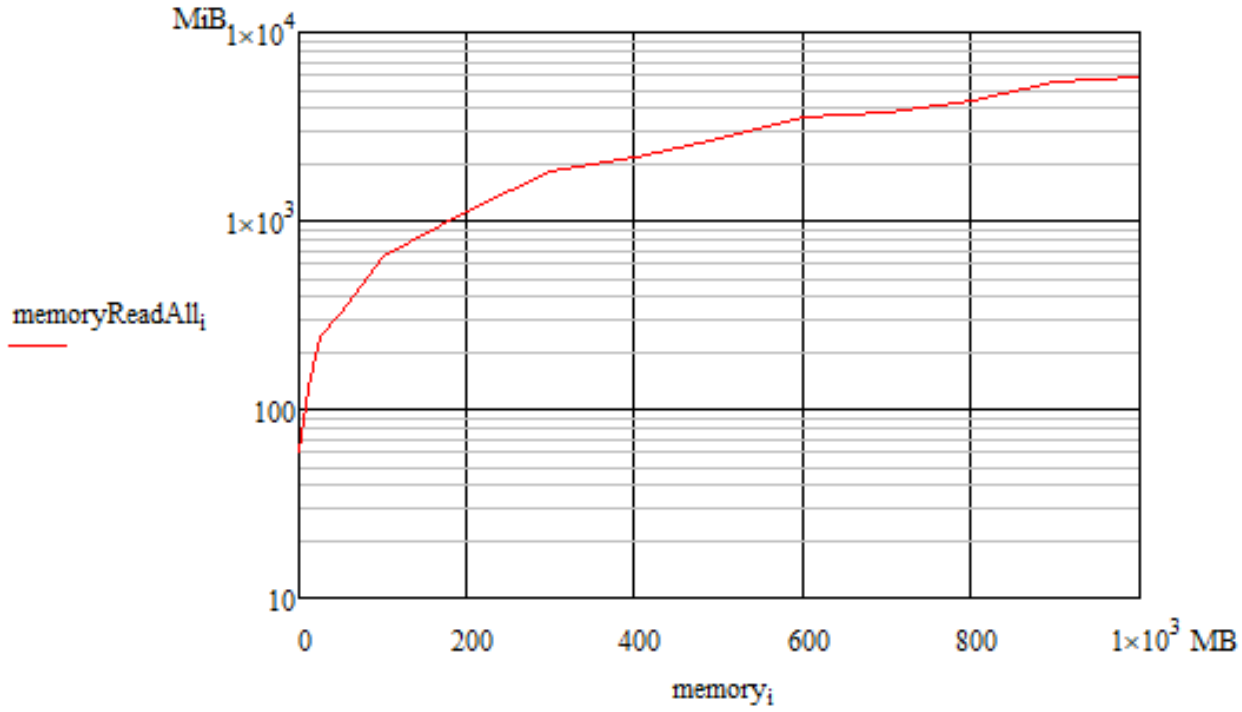


Рисунок 3.9 – Графік залежності пам’яті яку займає додаток від розміру файлу

Діаграма реалізації оптимізованого сервісного методу для завантаження CSV файлу представлена в дод. Б, рис. Б.4.

Для оптимізації використаної пам’яті метод UploadCsvFile був переписаний з урахуванням вже готових структур транзакцій. Новий метод (рис. 3.10) почав складатися з етапів які наведені нижче.

```
func (s *Service) UploadCsvFile(ctx context.Context, f IFileHeader) error {  5 usages
    file, err := f.Open()
    if err != nil : ErrOpenFile 
    defer file.Close()

    reader := bufio.NewReader(file)

    // reading file headers
    l, _, err := reader.ReadLine()
    if err != nil {
        if errors.Is(err, io.EOF) : nil 

        return err
    }
    headers := strings.Split(string(l), sep: ",")
    chTrans := make(chan []schemes.Transaction)
    chError := make(chan error)

    // create goroutine for insert all transactions by one database transaction
    go s.r.InsertToTransactions(ctx, s.c.ChunkSize, chTrans, chError)
    // check channel error on error at goroutine initialization
    if err = <-chError; err != nil : fmt.Errorf("InsertToTransactions initial goroutine: %w", err) 

    var errLoop error
```

Рисунок 3.10 – Реалізація оптимізованого сервісного методу для завантаження CSV файлу

Відкриття файлу: функція спочатку намагається відкрити файл CSV, використовуючи метод Open() на переданому об'єкті IFileHeader. У випадку помилки при відкритті файлу (наприклад, файл не знайдено), функція повертає помилку ErrOpenFile. Якщо файл успішно відкритий, він автоматично закривається після завершення роботи функції завдяки інструкції defer file.Close().

Читання заголовків файлу: використовуючи буферізований читач bufio.Reader, функція спробує прочитати перший рядок файлу, який містить заголовки стовпців. Ці заголовки розділяються комами і зберігаються в змінній headers.

Створення каналів для транзакцій і помилок: функція ініціює два канали - chTrans для передачі об'єктів транзакцій і chError для обробки помилок.

Запуск горутини для вставки транзакцій: через горутину запускається функція InsertToTransactions, що вставляє транзакції в базу даних. Ця функція використовує канали chTrans і chError для комунікації.

Перевірка помилок при ініціалізації горутини: перед входженням в основний цикл, функція перевіряє канал chError на наявність помилок, які могли виникнути під час ініціалізації горутини.

Основний цикл обробки файлу (рис.3.11):

Читання рядків: у цьому циклі функція читає рядки з файлу до тих пір, поки не буде досягнуто кінця файлу або не виникне помилка. Кожен рядок розглядається як потенційна транзакція.

Перетворення рядків на транзакції: рядки перетворюються на об'єкти транзакцій за допомогою функції BindFromCsv.

Обробка помилок під час читання: якщо виникає помилка при читанні файлу або під час перетворення рядка, цикл переривається.

Відправка транзакцій через канал: після обробки кожного рядка або групи рядків, об'єкти транзакцій відправляються через канал chTrans у горутину для вставки в базу даних.

```
mainLoop:
    for {
        records := make([]schemes.Transaction, 0, s.c.ChunkSize)
        select {
        case <-ctx.Done():
            // if user request is turn down connection
            break mainLoop
        default:
            // while ChunkSize to insert not equal to config value
            // fil chunk slices
            for int32(len(records)) != s.c.ChunkSize {
                row, _, err := reader.ReadLine()
                if err != nil {
                    if errors.Is(err, io.EOF) {
                        chTrans <- records
                        errLoop = <-chError
                        break mainLoop
                    }

                    errLoop = err
                    break mainLoop
                }

                var t schemes.Transaction
                if err = schemes.BindFromCsv(&t, strings.Split(string(row), sep: ","), headers); err != nil {
                    errLoop = err
                    break mainLoop
                }

                records = append(records, t)
            }

            chTrans <- records
            if errLoop = <-chError; errLoop != nil {
                break mainLoop
            }
        }
    }
```

Рисунок 3.11 – Основний цикл обробки файлу в сервісному методі для завантаження CSV файлу

Обробка помилок після основного циклу представлена на рис. 3.12.

Специфічні помилки: функція обробляє конкретні типи помилок, які могли виникнути в процесі обробки файлу, такі як помилки парсингу або порушення унікальних обмежень в базі даних.

Звичайні помилки: якщо виникають інші помилки, вони обробляються і повертаються з функції.

Завершення функції: Функція повертає nil, якщо всі операції були успішно завершені без помилок.

```
    if errLoop != nil {
        switch {
        case errors.Is(errLoop, schemes.ErrParsing):
            var erw *errs.ErrorWithMessage
            if errors.As(errLoop, &erw) {
                return &errs.ErrorWithMessage{
                    Err: ErrParsing,
                    Msg: erw.Msg,
                }
            }
        case errors.Is(errLoop, repository.ErrUniqueConstraint):
            var erw *errs.ErrorWithMessage
            if errors.As(errLoop, &erw) {
                return &errs.ErrorWithMessage{
                    Err: ErrTransactionExist,
                    Msg: erw.Msg,
                }
            }
        }

        return errLoop
    }

    return nil
}
```

Рисунок 3.12 – Обробка помилок після основного циклу в сервісному методі для завантаження CSV файлу

В результаті оптимізації були сформовані графіки залежності пам’яті, яку займає додаток (рис. 3.13) та залежності часу обробки файлу додатком (рис. 3.14) до та після оптимізації від розміру файлу.

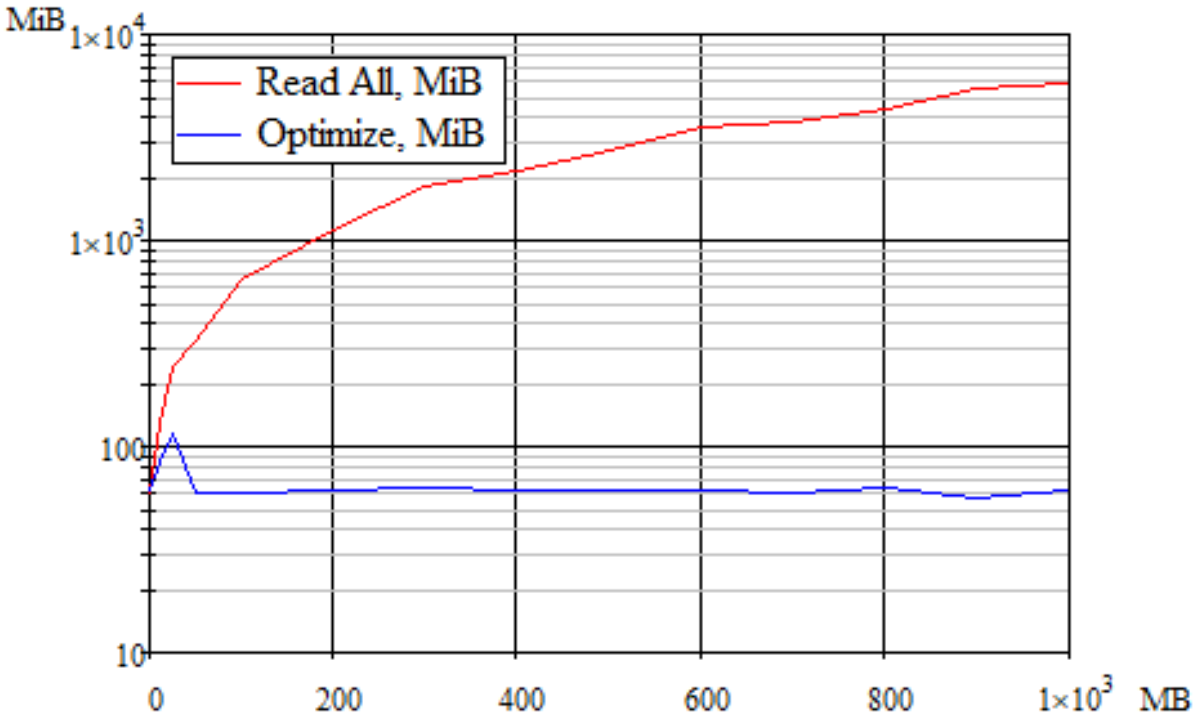


Рисунок 3.13 – Графік залежності пам’яті, яку займає додаток до та після оптимізації від розміру файлу

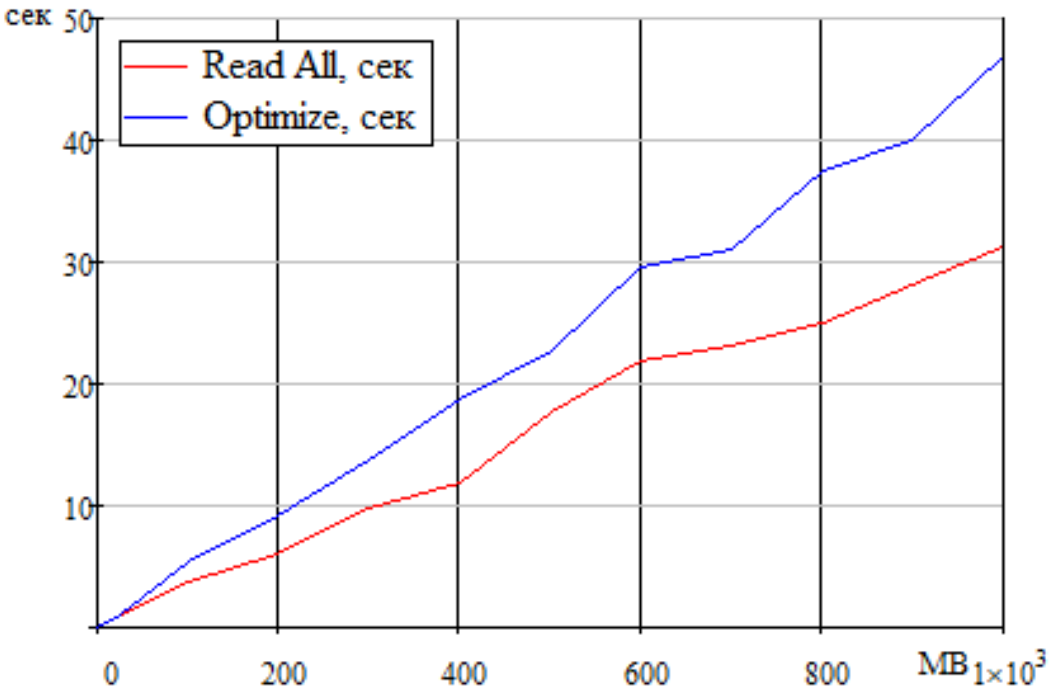


Рисунок 3.14 – Графік залежності часу обробки файлу додатком до та після оптимізації від розміру файлу

ВИСНОВКИ

В результаті дослідження, спрямованого на розробку білінгового додатку з метою поліпшення обробки сирої інформації, були досягнуті важливі висновки та наукові підстави для обрання конкретних технологій і рішень.

У процесі дослідження було проведено порівняння різних мов програмування, активно використовуваних у веброзробці. На основі аналізу було виявлено, що мова Go відзначається високою продуктивністю та швидкодією. Використання каналів та горутин в Go дозволяє ефективно керувати багатозадачними додатками, що особливо важливо для обробки великих обсягів даних.

Під час розгляду різних підходів до комунікації сервера з клієнтом, було обрано REST (Representational State Transfer) як принцип архітектури для обміну даними. REST є досить ефективним та простим у реалізації підходом для роботи з файлами та ресурсами.

У процесі дослідження різних систем управління базами даних було обрано PostgreSQL як оптимальний варіант для зберігання даних. Цей вибір базується на важливості надійності, широких можливостях та високому рівні сумісності PostgreSQL з мовою програмування Go.

Docker був визнаний надзвичайно важливим інструментом як для локальної розробки, так і для розгортання додатків. Його здатність пакувати всі необхідні залежності та середовище в контейнери дозволяє спростити процес розгортання та забезпечити передбачуваність у виробничому середовищі. Такий вибір базується на оптимізації розробки програмного забезпечення та ефективного використання ресурсів.

Загалом, в результаті експерименту встановлено, що витрати пам'яті при витягуванні всього файлу суттєво впливають на продуктивність додатку. Хоча оптимізація коду дозволила зменшити обсяг використовуваної пам'яті, виявилося, що це призводить до збільшення часу обробки файлу. Ця

суперечливість вимагає більш глибокого аналізу та розробки стратегій, які збалансують витрати пам'яті та час обробки для досягнення оптимальної продуктивності. Важливо враховувати, що оптимізація не завжди є однозначним питанням, і необхідно узгоджувати рішення з конкретними потребами та обмеженнями проекту. Ці рішення відкривають перспективи для подальших наукових та практичних досліджень у сфері оптимізації систем обробки даних та інтеграції з іншими інформаційними системами, враховуючи найсучасніші підходи та технології.

Результати роботи було апробовано на одинадцятій міжнародній науково-технічній конференції «Проблеми інформатизації» та опубліковано тези доповіді [13] за тематикою кваліфікаційної роботи.

ПЕРЕЛІК ПОСИЛАНЬ

1. Filename extension [Електронний ресурс]. – 2023. – Режим доступу до ресурсу: https://en.wikipedia.org/wiki/Filename_extension.

2. Jeremy H. gRPC vs. REST: Key Similarities and Differences [Електронний ресурс] / Jeremy. – 2023. – Режим доступу до ресурсу: https://blog.dreamfactory.com/grpc-vs-rest-how-does-grpc-compare-with-traditional-rest-apis.

3. gRPC [Електронний ресурс]. – 2023. – Режим доступу до ресурсу: https://en.wikipedia.org/wiki/GRPC

4. What's the Difference Between gRPC and REST? [Електронний ресурс] // Amazon Web Services, Inc. – 2023. – Режим доступу до ресурсу: https://aws.amazon.com/compare/the-difference-between-grpc-and-rest/?nc1=h_ls

5. PostgreSQL [Електронний ресурс]. – 2023. – Режим доступу до ресурсу: https://en.wikipedia.org/wiki/PostgreSQL

6. Comparing Database Management Systems: MySQL, PostgreSQL, MSSQL Server, MongoDB, Elasticsearch, and others [Електронний ресурс] // altexsoft. – 2023. – Режим доступу до ресурсу: https://www.altexsoft.com/blog/comparing-database-management-systems-mysql-postgresql-mssql-server-mongodb-elasticsearch-and-others/

7. Golang vs Python: Full Difference Explained [Електронний ресурс] // Interview Bit. – 2023. – Режим доступу до ресурсу: https://www.interviewbit.com/blog/golang-vs-python/#:~:text=Python%20is%20considered%20the%20best,glitches%20later%20in%20the%20production

8. Dziubałka P. Golang vs. Python – Which One to Choose? [Електронний ресурс] / P. Dziubałka, S. Karasiewicz // Soft Kraft. – 2023. – Режим доступу до ресурсу: https://www.softkraft.co/golang-vs-python/#:~:text=Applications%3A%20Python%20

shines%20when%20used,computing%20and%20cluster%20computing%20applications.

9. Introduction to Docker [Електронний ресурс] // Geeks for Geeks. – 2023. – Режим доступу до ресурсу: https://www.geeksforgeeks.org/introduction-to-docker/

10. Anshul G. Introduction to Docker [Електронний ресурс] / Ganvir Anshul // Medium. – 2023. – Режим доступу до ресурсу: https://medium.com/@anshulganvir/introduction-to-docker-337b9d09a079

11. Docker (software) [Електронний ресурс]. – 2023. – Режим доступу до ресурсу: https://en.wikipedia.org/wiki/Docker_(software)

12. Kubernetes [Електронний ресурс]. – 2023. – Режим доступу до ресурсу: https://en.wikipedia.org/wiki/Kubernetes

13. Дроздов Д.О. Аналіз сучасних мов програмування для обробки даних в інфокомунікаційних системах / Д.О. Дроздов, Д.В. Чеботарьова // Тези доповідей одинадцятої міжнародної науково-технічної конференції «Проблеми інформатизації», 16 – 17 листопада 2023 р., Баку – Харків – Бельсько-Бяла. 2023. – Том 1. – С. 69.