Міністерство освіти і науки України
Харківський національний університет радіоелектроніки

Факультет Інформаційно-аналітичних технологій та менеджменту
(повна назва)
Кафедра Інформатики
(повна назва)

КВАЛІФІКАЦІЙНА РОБОТА
Пояснювальна записка

рівень вищої освіти перший (бакалаврський)

РОЗРОБКА МІКРОСЕРВІСНОЇ ВЕБПЛАТФОРМИ ДЛЯ ІНТЕЛЕКТУАЛЬНОГО РОЗПІЗНАВАННЯ ТА КАТЕГОРИЗАЦІЇ ВИТРАТ ЗА ДОПОМОГОЮ АНАЛІЗУ ФІНАНСОВИХ ДОКУМЕНТІВ
(тема)

Виконав:
студент 4 курсу, групи ІТІНФ-20-1
Іткін Д.О.
(прізвище, ініціали)

Спеціальності 122 Комп'ютерні науки
(код і повна назва спеціальності)

Тип програми освітньо-професійна

Освітня програма Інформатика
(повна назва освітньої програми)

Керівник доц. Любченко В.А.
(посада, прізвище, ініціали)

Допускається до захисту

Зав. кафедри _____ Кобилін О.А.
(підпис) (прізвище, ініціали)

2024 р.

Харківський національний університет радіоелектроніки

Факультет Інформаційно-аналітичних технологій та менеджменту
(повна назва)

Кафедра Інформатики
(повна назва)

Рівень вищої освіти перший (бакалаврський)

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

Тип програми освітньо-професійна

Освітня програма Інформатика
(повна назва освітньої програми)

ЗАТВЕРДЖУЮ:
Зав. кафедри
_____
(підпис)
«____»_____ 2024 р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ

студентові Іткіну Денису Олександровичу
(прізвище, ім'я, по батькові)

1. Тема роботи Розробка мікросервісної вебплатформи для інтелектуального розпізнавання та категоризації витрат за допомогою аналізу фінансових документівних

затверджена наказом університету від 20 травня 2024 року № 464 Ст

2. Термін подання студентом роботи до екзаменаційної комісії 24 травня 2024 р.

3. Вихідні дані до роботи науково-методична та науково-технічна література, матеріали конференцій, дані інтернет-мережі, клієнти великих мовних моделей GPT-4 та Gemini Pro, Google Vision API, Google Cloud Storage.

4. Перелік питань, що потрібно опрацювати в роботі
1. Аналіз сучасних технологій OCR та LLM для обробки фінансових документів.
2. Оцінка ефективності різних методів автоматизації обробки фінансових документів з використанням мікросервісної архітектури.
3. Розробка та впровадження мікросервісів для обробки, аналізу та категоризації фінансових даних.

5. Перелік графічного матеріалу із зазначенням креслеників, схем, плакатів, комп'ютерних ілюстрацій (п.5 включається до завдання за рішенням випускової кафедри) BPMN-діаграма роботи застосунку, схема роботи використовуваних технологій, діаграма архітектури застосунку.

6. Консультанти розділів роботи (п.6 включається до завдання за наявності консультантів згідно з наказом, зазначеним у п.1)

Найменування розділу	Консультант (посада, прізвище, ім'я, по батькові)	Позначка консультанта про виконання розділу	
		підпис	дата

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів роботи	Терміни виконання етапів роботи	Примітка
1	Отримання завдання на кваліфікаційну роботу	08.04.2024	
2	Аналіз завдання, підбір літератури	08.04.24-15.04.24	
3	Аналіз літератури з досліджуваної проблеми	16.04.24-20.04.24	
4	Аналіз технічних засобів	21.04.24-25.04.24	
5	Програмна реалізація	28.04.24-12.05.24	
6	Оформлення пояснювальної записки	20.05.24-26.05.24	
7	Перевірка на плагіат	27.05.24	
8	Рецензування	28.05.24	
9	Підготовка презентації та доповіді	29.05.24-02.06.24	
10	Занесення роботи в електронний архів	03.06.24	
11	Попередній захист кваліфікаційної роботи	05.06.24	

Дата видачі завдання 8 квітня 2024 р.

Студент _____
(підпис)

Керівник роботи _____ _____доц. Любченко В.А._____
(підпис) (посада, прізвище, ініціали)

РЕФЕРАТ/ABSTRACT

Пояснювальна записка до кваліфікаційної роботи: 76 с., 6 табл., 30 рис., 1 дод., 30 джерел.

ВЕБПЛАТФОРМА, МІКРОСЕРВІСИ, РОЗПІЗНАВАННЯ ТЕКСТУ, КАТЕГОРИЗАЦІЯ, VISION, SAGA, ПРОМПТ-ПРОЄКТУВАННЯ, LLM МОДЕЛІ, ІНТЕЛЕКТУАЛЬНИЙ АНАЛІЗ.

Об’єктом роботи є різноманітні фінансові документи.

Метою роботи є розробка платформи, що забезпечує автоматичне витягування і категоризацію інформації з документів для спрощення процесу ведення особистого бюджету. Значну увагу приділено промпт-інжинірінгу для створення ефективних запитів до LLM моделей, щоб оптимізувати обробку і аналіз текстових даних.

Використано технології Google Cloud, Google Vision API, та LLM для деталізованого аналізу текстів документів. Проведено розробку мікросервісів, що включають: сервіси прийому файлів, аналізу даних, верифікації інформації та управління транзакціями.

Результатом роботи є реалізація платформи, яка дозволяє користувачам управляти і аналізувати свої фінанси на основі оброблених документів.

WEBPLATFORM, MICROSERVICES, TEXT RECOGNITION, CATEGORIZATION, VISION, SAGA, PROMPT-ENGINEERING, LLM MODELS, INTELLIGENT ANALYSIS.

The object of the work is a variety of financial documents.

The aim of the project is to develop a platform that provides automatic extraction and categorization of information from receipts to simplify the process of personal budget management. Significant attention is devoted to prompt engineering for creating effective queries to LLM to optimize the processing and analysis of text data.

Technologies used include Google Cloud, Google Vision API, and LLM for detailed analysis of receipt texts. The development involved microservices including receipt receiving services, data analysis, information verification, and transaction management.

The result of the work is the implementation of a platform that allows users to manage and analyze their finances based on processed receipts.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ, СКОРОЧЕНЬ І ТЕРМІНІВ

API – Application Programming Interface (інтерфейс програмування застосунків)

ACID – Atomicity, Consistency, Isolation, Durability (Атомарність, Послідовність, Ізоляція, Довговічність)

GCP – Google Cloud Platform (платформа Google Cloud)

OCR – Optical Character Recognition (оптичне розпізнавання символів)

LLM – Large Language Models (великі моделі даних)

SAGA – Сага паттерн (паттерн для управління транзакціями у мікросервісних архітектурах)

JSON – JavaScript Object Notation (формат обміну даними)

DB – Database (база даних)

HTTP – Hypertext Transfer Protocol (протокол передачі гіпертексту)

REST – Representational State Transfer (архітектурний стиль взаємодії компонентів у мережі)

GPT – Generative Pre-trained Transformer (генеративна попередньо натренована трансформерна модель)

MQ – Message Queue (черга повідомлень)

CRUD – Create, Read, Update, Delete (створення, читання, оновлення, видалення)

AI – Artificial Intelligence (штучний інтелект)

BPMN – Business Process Model and Notation (модель бізнес–процесу та нотація)

ВСТУП

У сучасному світі обробка даних та автоматизація бізнес-процесів займають ключове місце в оптимізації повсякденного життя людей та їх взаємодії з різноманітними сервісами. Одним із важливих аспектів в цьому контексті є обробка та аналіз фінансових документів, таких як чеки та квитанції, що є невід'ємною частиною управління особистими та корпоративними фінансами. Ручне введення даних з документів є часозатратним і схильним до помилок, що зумовлює потребу в автоматизації цього процесу.

Актуальність розробки мікросервісної вебплатформи для інтелектуального розпізнавання [1] та категоризації витрат на основі інформації з фінансових документів обумовлена зростаючим попитом на технологічні рішення, які можуть спростити обробку великої кількості інформації, забезпечуючи точність та ефективність обробки даних. Це особливо важливо в контексті зростання електронної комерції та онлайн-платежів, де швидке та точне витягування даних із фінансових документів є критично важливим.

Проєкт має на меті вирішити проблему швидкого та точного розпізнавання, аналізу та категоризації інформації з чеків за допомогою сучасних технологій обробки зображень та обробки природної мови. Використання мікросервісної архітектури разом з моделями штучного інтелекту [2, 3], зокрема LLM, та промпт-проєктування для оптимізації запитів до цих моделей, дозволяє створити гнучку, масштабовану систему, яка може адаптуватися до потреб різних користувачів.

1 ІННОВАЦІЙНІ ПІДХОДИ ДО АВТОМАТИЗАЦІЇ ОБРОБКИ ФІНАНСОВИХ ДОКУМЕНТІВ

1.1 Потреби бізнесу та індивідуальних користувачів

Сьогоднішній бізнес та індивідуальні користувачі стикаються з цілою низкою викликів: від жвавої конкуренції та стрімких технологічних змін до постійної економічної нестабільності. У такому динамічному середовищі, як для компаній, так і для особистого використання, виникає потреба у новітніх, ефективних методах управління фінансами.

Одна з ключових проблем у фінансовому управлінні – це необхідність точного та оперативного контролю витрат. Звичайні люди, які прагнуть слідкувати за своїми фінансами, також зіштовхуються з проблемою нестачі часу для ручного введення даних про кожну транзакцію у фінансові застосунки. Це особливо важливо для тих, хто веде активний спосіб життя та не бажає витрачати час на трудомісткі фінансові записи [4, 5].

У відповідь на ці виклики, розробка застосунку для розпізнавання на основі даних із фінансових документів стає надзвичайно актуальною. Використання штучного інтелекту і машинного навчання може дозволити автоматизувати аналіз фінансових даних, забезпечуючи швидкий і точний доступ до інформації про витрати.

Така технологічна інновація не тільки значно спростить процес ведення фінансів для компаній і приватних осіб, але й дозволить їм приймати обґрунтовані фінансові рішення без значних затрат часу.

1.2 Застосування OCR та LLM для вирішення актуальних проблем

Задача автоматизації передбачає ряд технічних викликів: від точного збору даних до їх ефективної обробки, збереження та інтеграції з іншими

системами. Технології оптичного розпізнавання символів (OCR) і машинне навчання грають ключову роль у створенні ефективних рішень для обробки зображень [6], зменшуючи кількість помилок і покращуючи загальну ефективність процесів.

1.2.1 Опис та порівняння технологій OCR

Основний принцип роботи OCR полягає у перетворенні зображення тексту на електронний формат. Спочатку зображення проходить через попередню обробку, що включає видалення шуму, покращення контрасту та приведення зображення до оптимального стану розпізнавання [7]. Потім відбувається сегментація зображення окремі символи чи слова. Після цього кожен символ або слово проходить через класифікацію [8], де використовуються алгоритми машинного навчання для визначення відповідного символу або слова [9]. Нарешті, розпізнаний текст виводиться як електронного документа.

Однією з найсучасніших технологій OCR, яка відома своєю високою продуктивністю і точністю, є Google Vision API. Цей інструмент розроблений Google і використовує передові методи машинного навчання та штучного інтелекту для розпізнавання тексту на зображеннях:

а) висока точність і надійність: Google Vision API відомий своєю здатністю точно розпізнавати текст навіть на зображеннях низької якості. Це досягається завдяки використанню передових алгоритмів штучного інтелекту, які постійно вдосконалюються на величезній кількості даних, що є у розпорядженні Google;

б) підтримка багатьох мов: на відміну від багатьох інших OCR систем, Google Vision API підтримує широкий спектр мов, що робить його ідеальним рішенням для міжнародних застосувань та проктів, що вимагають обробки документів різними мовами;

в) легкість інтеграції: як частина хмарних сервісів Google, Vision API легко інтегрувати з іншими продуктами та сервісами Google, що дозволяє розробникам створювати комплексні рішення для обробки даних. Крім того, доступ до API можливий через зручний вебінтерфейс або через прямі виклики API.

Ще однією популярною технологією OCR є Tesseract. Tesseract розроблений Google і надає можливість розпізнавання тексту різними мовами. Він заснований на нейронній мережі та використовує методи машинного навчання для покращення точності розпізнавання. Tesseract має відкритий вихідний код, що дозволяє розробникам налаштовувати та оптимізувати його під свої потреби.

1.2.1.1 Задачі та обмеження OCR технологій

OCR розшифровується як оптичне розпізнавання символів. Ця технологія використовується для перетворення друкованого чи рукописного тексту або відсканованих зображень у дані, які може використовувати комп’ютер [10]. Після сканування фізичного об’єкта програмне забезпечення може розпізнати літери, цифри та символи і зрозуміти їхній зміст.

Машинне навчання (ML) покращує OCR, дозволяючи йому адаптуватися до різних варіантів введення (тобто шрифтів, розмірів тексту, почерку, розмитих зображень тощо) і використовувати багатоетапні процеси для усунення або мінімізації потенційних проблем. OCR-системи зазвичай складаються з двох етапів:

– сегментація: Цей етап розбиває зображення на окремі символи або рядки тексту;

– розпізнавання: Цей етап використовує алгоритми машинного навчання для ідентифікації символів або рядків тексту.

На рисунку 1.1 можна побачити приклад обробки літери А на зображенні:

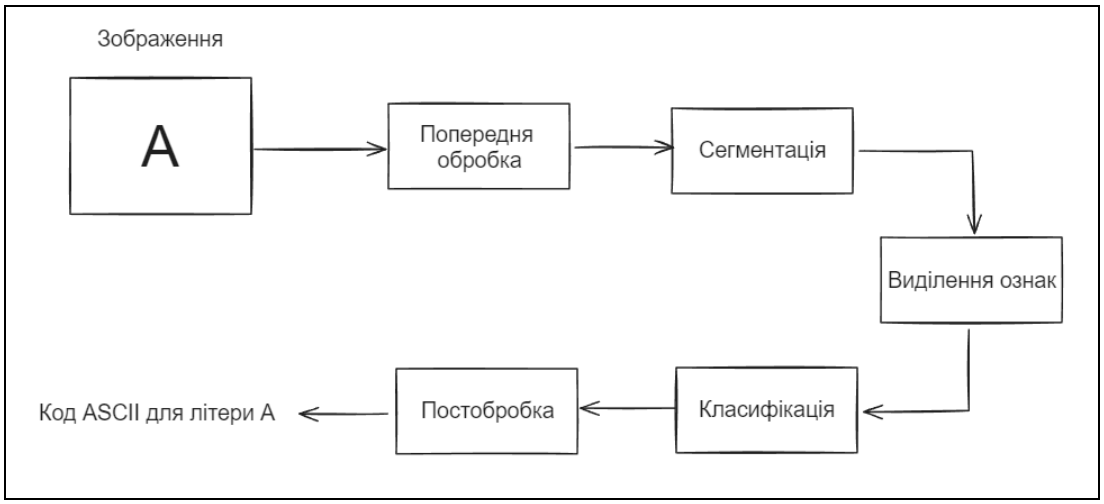


Рисунок 1.1 – Схема конвертації літери «А» з зображення у формат ASCII

OCR має свої обмеження. Наприклад, він може мати проблеми з розпізнаванням тексту з низькою якістю, розмитими символами або нестандартними шрифтами. Для покращення точності розпізнавання OCR може використовувати словники, мовні моделі та контекстну інформацію [11]. Також можуть бути застосовані методи машинного навчання для навчання OCR на великій кількості розмічених даних.

Однак OCR-технології значно покращилися за останні роки, і тепер вони можуть забезпечити високий рівень точності для багатьох типів документів.

1.2.2 Великі мовні моделі (LLM) у сучасному світі

LLM розшифровується як Large Language Model (велика мовна модель). Цей тип програм використовує штучний інтелект для розуміння тексту і створення свіжого контенту. Вони використовують нейронні мережі та великі масиви даних для навчання, щоб підвищити точність і

ефективність. Для управління ланцюгами поставок LLM можна налаштувати на основі галузевих знань [12].

LLM можуть ефективно вдосконалити процес обробки документів, наприклад чеків, у таких напрямках:

а) коригування помилок OCR: LLM можуть виявляти й коригувати помилки, які з’являються в результаті роботи систем оптичного розпізнавання символів;

б) класифікація витрат: LLM можуть аналізувати текст на чеках та автоматизувати процес розподілу витрат за категоріями;

в) збагачення інформації: LLM можуть збільшувати цінність даних з чеків, додаючи додаткові відомості, як-от описи товарів або назви торгових точок.

Однак LLM також має свої обмеження. Наприклад, він може виникнути з проблемами при аналізі складних макетів документів або наявності нестандартних елементів. Для підвищення точності та надійності LLM можуть бути застосовані методи машинного навчання, такі як глибокі нейронні мережі або ансамблі моделей [13].

1.2.2.1 Принципи ефективного формулювання запитів до LLM

Prompt у контексті LLM (Large Language Models) – це запит або вказівка, яку користувач подає моделі для отримання відповіді або згенерованого тексту [14]. Правильне проєктування запиту є ключовим для ефективної роботи LLM, оскільки якість та релевантність відповіді залежить від того, наскільки чітко та конкретно сформульовано запит.

Для ефективного проєктування запитів у LLM слід враховувати наступне:

а) специфічність: запит має бути достатньо конкретним, щоб модель могла зрозуміти, яку саме відповідь чи інформацію очікує користувач;

б) контекст: забезпечення достатньої контекстної інформації дозволяє моделі краще зорієнтуватися в запиті;

в) очікувані результати: якщо є певні очікування до формату відповіді, їх також слід включити у запит;

г) уникнення двозначності: неоднозначні або багатозначні запити можуть призвести до непередбачуваних результатів;

д) ітераційний підхід: проєктування запитів може вимагати декількох ітерацій з уточненням та тестуванням, щоб досягти оптимального результату;

е) дотримання етики та політики використання: завжди важливо стежити, щоб формулювання запиту відповідало етичним стандартам і політиці використання моделі.

Проєктування ефективного запиту вимагає розуміння можливостей та обмежень конкретної LLM, а також ясної визначеності цілей, яких хоче досягти користувач (табл. 1.1).

Таблиця 1.1 – Принципи формулювання запитів до LLM

Принцип	**Опис**	**Приклади**
1	**2**	**3**
Специфічність	Запит повинен чітко висловлювати запитувану інформацію або завдання.	«Перекладіть наступний текст з англійської на українську: ‘Hello, how are you?’»

Продовження таблиці 1.1

1	2	3
Контекст	Надати достатньо контексту, щоб модель могла правильно інтерпретувати запит.	«В контексті медицини, що означає термін «анамнез»?»
Очікувані результати	Вказати формат або структуру відповіді, якої очікуєте.	«Наведіть список з п’яти найбільш популярних книг про саморозвиток, випущених у 2023 році.»
Уникнення двозначності	Уникати запитів, які можуть мати кілька інтерпретацій.	«Які конкретні кроки слід вжити для оптимізації сайту під пошукові системи?»
Ітераційний підхід	Враховувати необхідність коригування запитів для досягнення кращих результатів.	Використовувати зворотний зв’язок для уточнення запитів: «Які зміни потрібно внести в мій останній запит, щоб отримати більш деталізовану відповідь?»
Дотримання етики та політики використання	Впевнитися, що ваш запит не порушує етичних норм і правил використання моделі.	Уникати запитів, які можуть вимагати порушення конфіденційності або використання авторського контенту.

1.2.3 Обмеження використання технологій OCR і LLM

Незважаючи на значні переваги, застосування технологій оптичного розпізнавання символів (OCR) та великих мовних моделей (LLM) має певні обмеження, особливо при роботі з фінансовими документами, такими як чеки.

Однією з основних проблем є потенційно низька точність розпізнавання тексту, особливо коли йдеться про чеки з нестандартними шрифтами або розмитими зображеннями. Це може призводити до помилок у розпізнаванні, що, у свою чергу, спотворює результати категоризації витрат.

Іноді чеки містять специфічні поля, такі як коди товарів або номери рахунків, які можуть бути важкими для розпізнавання з використанням стандартних налаштувань OCR і LLM. У таких випадках необхідно адаптувати та додатково навчати моделі для точного виявлення та обробки цих даних.

Використання OCR і LLM вимагає високоякісних зображень для ефективного розпізнавання. Погане освітлення, розмитість або пошкодження на чеку можуть істотно знизити точність розпізнавання. Важливо забезпечити адекватну якість зображень під час сканування або фотографування для забезпечення найкращих результатів.

У підсумку, хоча технології OCR і LLM є потужними інструментами для розпізнавання інформації з різних типів документів, але важливо враховувати їхні обмеження. Це дозволить оптимізувати процеси і забезпечити більш точне та ефективне управління даними.

1.3 Порівняльний аналіз монолітної та мікросервісної архітектури

Вибір архітектури програмного забезпечення є фундаментальним рішенням, яке визначає не тільки процеси розробки та розгортання застосунків, але й здатність системи до швидкої адаптації до змінних вимог ринку та технологій. Сучасні IT-проєкти зазвичай опираються на одну з двох основних архітектур: монолітну або мікросервісну. Кожна з них має свої унікальні характеристики, переваги та обмеження, що робить вибір між ними критичним.

Монолітна архітектура, традиційно використовувана в багатьох ранніх розробках програмного забезпечення, організує всі основні компоненти застосунка у єдиний, неділимий блок. Це створює сприятливі умови для простоти розгортання та оперативного управління. Проте, монолітні системи можуть швидко стати незграбними і важкими для масштабування, особливо у великих організаціях із швидко змінними технологічними потребами.

З іншого боку, мікросервісна архітектура пропонує модульний підхід, де кожен компонент, або «сервіс», функціонує незалежно і комунікує з іншими через легко визначені інтерфейси (API) [15]. Така декомпозиція дозволяє більш гнучке управління розробкою, тестуванням та впровадженням змін, покращуючи надійність та легкість впровадження нових технологій. На рисунку 1.2 представлено діаграму, яка порівнює монолітну та мікросервісну архітектури програмного забезпечення.

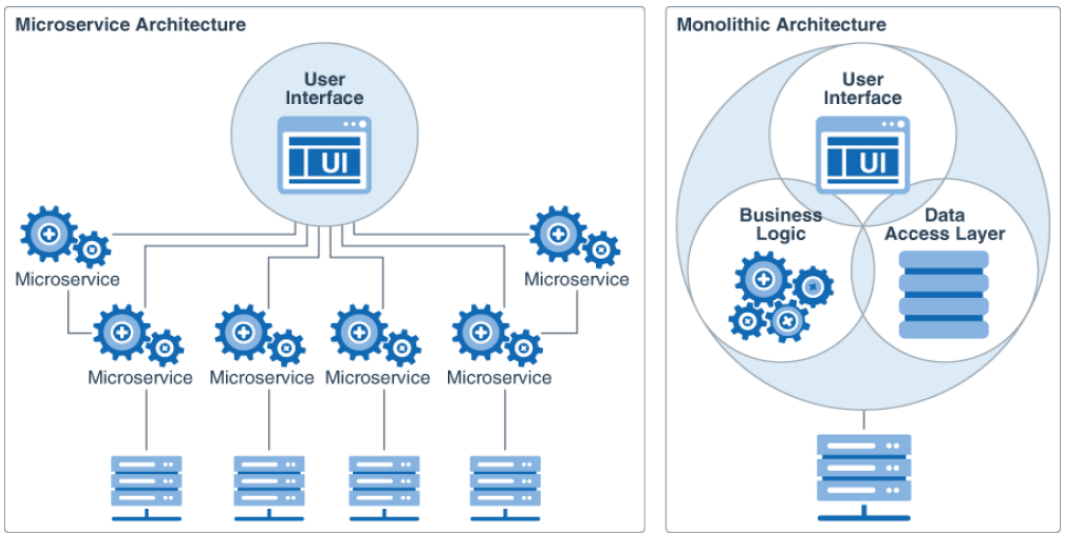


Рисунок 1.2 – Діаграма порівняння системних архітектур

1.4 Огляд ринкових рішень та оцінка їх недоліків

Сучасний бізнес все більше спирається на технології для оптимізації своєї діяльності та підвищення продуктивності. Однією з ключових технологій, яка знайшла широке застосування в області фінансів та бухгалтерського обліку, є оптичне розпізнавання символів (OCR). Використання цих технологій дозволяє автоматизувати процес обробки фінансових документів, що забезпечує ефективне управління фінансами підприємства.

Аналіз цих програмних засобів дозволить визначити їх функціональні недоліки, а також виявити потенційні області застосування в конкретних умовах бізнесу.

1.4.1 ABBYY FlexiCapture

ABBYY FlexiCapture забезпечує потужні можливості OCR для точного розпізнавання текстів з чеків та інших документів, підтримуючи мультимовність та можливість адаптації під конкретні потреби користувача.

Широко застосовується в банківському секторі та корпоративному бухгалтерському обліку для автоматизації введення даних та зниження людських помилок. На рисунку 1.3 можна побачити приклад інтерфейсу цього застосунку:

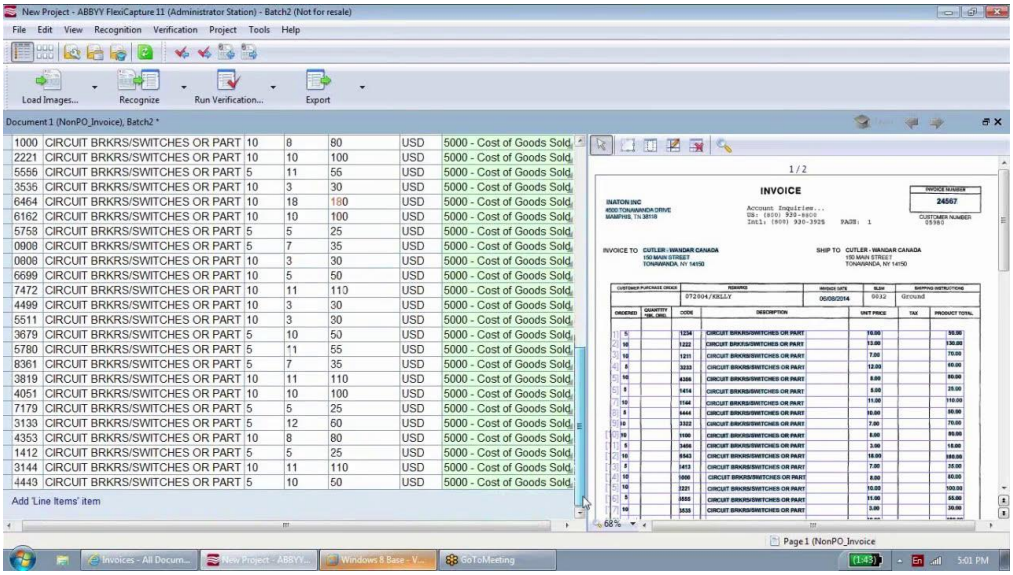


Рисунок 1.3 – Інтерфейс застосунку ABBYY FlexiCapture

Основні недоліки даного застосунку є:

– вартість: висока ціна рішення може бути обтяжливою для малого та середнього бізнесу;

– складність налаштування: для ефективного використання потрібні значні витрати часу на налаштування та тренування системи, що може виявитися складним без спеціалізованих знань.

1.4.2 Expensify

Expensify використовує OCR для автоматизації процесу ведення звітів про витрати, включаючи розпізнавання та категоризацію інформації з чеків. Були виділені такі недоліки цього застосунку:

– затримки в обробці: незважаючи на автоматизацію, користувачі Expensify іноді стикаються з затримками в обробці даних, особливо під час високого обсягу запитів. Це може стати проблемою для бізнесів, де швидке ведення звітності є критично важливим, зокрема в кінці фінансових періодів або під час подорожей;

– автоматична категоризація: система може помилково віднести покупки до невідповідних категорій, що потребує подальшого ручного втручання для виправлення помилок. Це знижує ефективність автоматизації та може призвести до неточностей у фінансовій звітності.

Нижче на рисунку 1.4 можна побачити інтерфейс застосунку Expensify:

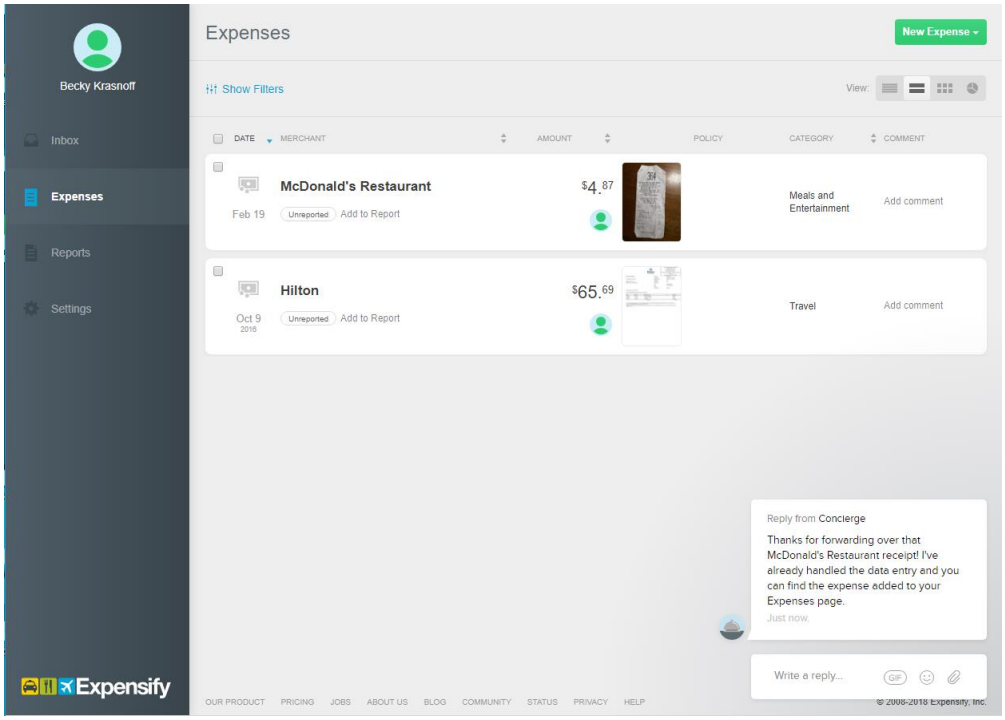


Рисунок 1.4 – Інтерфейс застосунку Expensify

1.5 Постановка задачі

У сучасному світі обробки даних із фінансових документів стає дедалі важливішим завданням, зважаючи на потребу автоматизації обліку та

управління особистими фінансами. Важливість точного та автоматизованого аналізу фінансових документів зростає, оскільки вона дозволяє користувачам ефективніше контролювати свої витрати та планувати бюджет.

Об’єктом дослідження є дані з різноманітних чеків і фінансових документів, отримані від користувачів.

Метою роботи є розробка мікросервісної вебплатформа, яка здатна розпізнавати, категоризувати та візуалізувати витрати на основі даних, отриманих із чеків, застосовуючи технології оптичного розпізнавання тексту (OCR) та велику мовну модель (LLM).

Для досягнення поставленої мети необхідно вирішити наступні завдання:

– провести аналіз існуючих технологій та методів розпізнавання тексту з зображень, зокрема, використання OCR;

– розробити мікросервіс для розпізнавання тексту з фінансових документів з використанням Google Vision API або іншого аналогічного інструменту;

– розробити мікросервіс для категоризації, створення транзакцій та обробки даних, використовуючи мовні моделі та техніки промпт-інжинірінгу для оптимізації запитів до моделей;

– інтегрувати мікросервіси у єдину вебплатформу, яка дозволяє користувачам завантажувати зображення чеків, переглядати результати розпізнавання та аналізувати свої витрати;

– тестування системи на реальних даних для перевірки точності та ефективності розпізнавання та категоризації даних;

– розробка інтерфейсу користувача, що забезпечує зручне та інтуїтивно зрозуміле взаємодія з платформою.

2 АНАЛІЗ ТЕХНОЛОГІЙ ДЛЯ АВТОМАТИЗАЦІЇ ФІНАНСОВИХ ДОКУМЕНТІВ

2.1 Розгляд масштабованої вебархітектури та її ключові аспекти

Масштабованість є критичною для успіху будь-якої вебзастосунку. Незалежно від розміру проєкту, важливо бути готовими до збільшення кількості користувачів та впевнитися, що система впорається з вищим навантаженням. Існує ризик, що система може виявитися недостатньо адаптивною і не зможе обробити збільшене навантаження. Тому важливо зосередитися на масштабуванні програм з самого початку процесу розробки.

Цей розділ присвячений масштабованому вебзастосунку, який може ефективно управляти великим потоком даних і уникати несподіваних збоїв. Розпочнемо з огляду масштабованості вебсистеми та окреслимо основні принципи її створення.

Масштабованість означає здатність вебпрограми витримувати ріст числа користувачів, які взаємодіють з програмою. Така програма повинна ефективно працювати з будь-якою кількістю користувачів і адаптуватися до коливань трафіку. Фактори, що впливають на масштабованість вебзастосунку, включають:

а) архітектура вебзастосунків і якісно написаний код, які є ключовими для створення масштабованих систем;

б) вибір фреймворку, який впливає на продуктивність програми, повинен враховувати різні бізнес-вимоги;

в) навантажувальне тестування для виявлення і вирішення слабких місць, щоб забезпечити стабільність;

г) обрання надійних сторонніх послуг для уникнення збоїв в роботі.

Не достатньо лише написати програмне забезпечення чи налаштувати сервери. Важливе регулярне обслуговування та управління як програмами,

так і серверами, за допомогою платформ і інструментів, які дозволяють легко відслідковувати та оновлювати програми, забезпечуючи їхню ефективність. У проєктуванні системи має бути враховано основні принципи масштабування. Багато з них можуть перетинатися або навіть конфліктувати, але вони життєво важливі для процесу проєктування та мають бути збалансовані. Оскільки ці принципи пов’язані з проєктуванням, ми розглянемо їх детальніше нижче.

2.1.1 Огляд архітектурних підходів

Вибір правильної архітектури для розробки та розгортання програм є важливим у сучасному світі програмування. Методи, такі як мікросервіси та сервіс-орієнтована архітектура (SOA), пропонують значну гнучкість для розробки програм, які не можуть працювати в рамках традиційних монолітних структур. Щоб вибрати найкращий для конкретного використання, важливо знати про відмінності цих архітектур. У цьому розділі ми розглянемо різні типи архітектур, а також їхні основні риси та приклади їх використання.

Монолітна програма побудована як єдиний блок, де бізнес-логіка, доступ до даних і інтерфейс користувача тісно інтегровані та працюють на одній платформі. У типовому моноліті можуть бути серверні програми, які взаємодіють з базою даних, а також клієнтський інтерфейс, створений за допомогою HTML і JavaScript. Масштабування є основним викликом монолітної архітектури, оскільки будь-яке додавання або оновлення функціоналу вимагає змін у всій системі, що може призводити до довгих відключень та складностей у тестуванні та впровадженні.

У відповідь на обмеження монолітних систем, була розроблена сервіс-орієнтована архітектура (SOA), яка дозволяє розділити програму на окремі сервісні компоненти. Ці компоненти взаємодіють один з одним за

допомогою визначених інтерфейсів для досягнення бізнес-цілей, забезпечуючи більшу гнучкість і масштабованість. SOA зазвичай використовує хмарні рішення для забезпечення необхідних платформ і сервісів, що сприяє легшому розгортанню та інтеграції різних бізнес-застосунків.

Застосування SOA та мікросервісів надає значні переваги у розробці та підтримці великих, розподілених програмних систем, що є особливо важливим у складних доменних областях, де потрібно забезпечити високий рівень масштабованості та надійності.

Мікросервісна архітектура, як логічне розвиток SOA, розділяє додаток на дрібніші, повністю незалежні компоненти, забезпечуючи кожному з них можливість бути розгорнутим окремо. Такий підхід спрощує розуміння, розробку і тестування окремих сервісів, а також підтримку неперервної доставки і поліпшень. Розгортання мікросервісів виконується у власних контейнерах, що дозволяє легко масштабувати і забезпечувати високу доступність і надійність системи [16].

Нижче наведено таблицю 2.1, яка порівнює монолітну, сервісно-орієнтовану та мікросервісну архітектуру за рядом критичних параметрів.

Таблиця 2.1 – Порівняльний аналіз архітектурних підходів

Критерії	**Монолітна архітектура**	**Сервісно–орієнтована архітектура**	**Мікросервісна архітектура**
1	**2**	**3**	**4**
Проєктування	Складність розуміння та зміни великих програм	Гнучка комбінація сервісів, що адаптується до змін	Невеликі, незалежні компоненти, що спрощують розуміння та розвиток

Продовження таблиці 2.1

1	2	3	4
Розгортання	Одноразове розгортання всієї системи	Менша гнучкість порівняно з мікросервісами	Незалежне розгортання кожного сервісу, що забезпечує гнучкість
Відмовостійкість	Падіння одного компоненту впливає на всю систему	Залежність від стабільності кожного сервісу	Незалежність сервісів забезпечує високу відмовостійкість
Масштабованість	Важко масштабувати через внутрішні залежності	Можливі ускладнення через спільні залежності сервісів	Висока масштабованість завдяки незалежності компонентів
Гнучкість	Обмежена можливість адаптації та оновлення	Середня, обмежена взаємними залежностями	Висока, кожен сервіс може використовувати різні технології
Розробка	Всі команди працюють над однією кодовою базою	Модульний підхід збільшує паралельність розробки	Незалежні команди працюють над своїми сервісами
Оновлення	Повільні оновлення через внутрішні залежності	Легше оновлювати окремі сервіси	Швидкі оновлення завдяки незалежності сервісів
Тестування	Комплексне тестування всієї системи	Тестування окремих сервісів зменшує ризики	Незалежне тестування кожного сервісу полегшує ідентифікацію проблем
Постачання	Складне і тривале через масштаб системи	Більш швидке постачання через незалежність компонентів	Дуже швидке завдяки автоматизації процесів

Продовження таблиці 2.1

1	2	3	4
Безпека	Єдиний захист для всієї системи може бути вразливим	Розподілений захист зменшує ризики	Вищий рівень безпеки завдяки ізоляції сервісів

2.1.2 Мікросервісна архітектура як рішення

Основним компонентом розробки системи, яка базується на використанні OCR (оптичного розпізнавання символів) і LLM (машинного навчання на основі міток), є дизайн мікросервісної вебплатформи для інтелектуального розпізнавання та категоризації витрат.

Основним завданням цієї вебплатформи є автоматичне розпізнавання та категоріювання витрат на основі даних з фінансових документів. Для цього використовується LLM, який навчається на основі даних для класифікації витрат за назвами магазинів, і OCR, який дозволяє витягувати текстову інформацію з зображень документів. Архітектура вебплатформи повинна забезпечувати ефективну роботу з великими кількостями даних, забезпечувати високу швидкість обробки та точність розпізнавання, а також бути масштабованою, щоб дозволити обробляти велику кількість запитів одночасно. Основними компонентами архітектури мікросервісної вебплатформи є:

а) сервіс розпізнавання тексту: сервіс забезпечує аналіз зображень фінансових документів за допомогою технологій оптичного розпізнавання символів (OCR). Він використовує передові алгоритми та бібліотеки для забезпечення високої точності екстракції тексту, що є критично важливим для точності подальшого аналізу;

б) сервіс великої мовної моделі (LLM): цей сервіс обробляє текстову інформацію, отриману з сервісу розпізнавання тексту, застосовуючи навчені мовні моделі для аналізу та витягу ключової інформації з фінансових

документів. Сервіс здатний паралельно обробляти численні запити, що забезпечує його масштабованість та ефективність;

в) сервіс зберігання файлів: відповідає за зберігання фінансових документів, завантажених користувачами, а також за управління інформацією про статус обробки кожного файлу. Цей сервіс гарантує безперервний доступ до даних і здатний витримувати великі обсяги інформації;

г) сервіс транзакцій: цей сервіс зберігає підтверджені транзакції користувача. У ньому зберігаються дані, здебільшого з фінансових документів;

д) інтерфейс користувача: ця частина є вебінтерфейсом, за допомогою якого користувач може завантажувати зображення чеків, переглядати результати розпізнавання та категорії та контролювати витрати. Інтерфейс користувача повинен бути простим у використанні, зручним і адаптованим до різних пристроїв.

Кожен із цих сервісів (рис. 2.1) відіграє важливу роль у забезпеченні функціональності та ефективності вебплатформи, а їхня комбінація створює надзвичайно міцну систему обробки фінансових документів, яка може бути масштабована та ефективна.

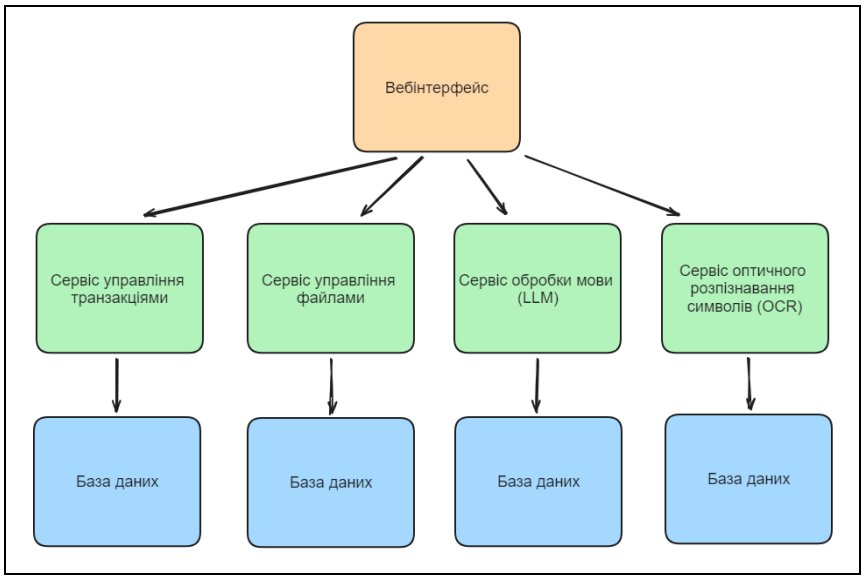


Рисунок 2.1 – Початкова версія мікросервісної архітектури

2.1.3 Шаблон розподілених транзакцій Saga

Тепер у нас є набір незалежних мікросервісів, кожен з яких може оновлюватися та масштабуватися окремо. Це призводить до проблем атомарності, цілісності та консистентності транзакцій. Як забезпечити, щоб усі мікросервіси, що беруть участь в одній великій транзакції, виконали свою роботу успішно та не порушили цілісність даних?

Саме тут нам на допомогу приходить цікавий патерн Saga. Суть його в тому, щоб розбити велику транзакцію на низку дрібніших, локальних транзакцій усередині кожного мікросервісу. Якщо щось іде не так, ми можемо застосувати компенсуючі дії для повернення системи до попереднього узгодженого стану. Тому Saga стає потужним інструментом для керування транзакціями у мікросервісній архітектурі.

Паттерн Saga є методологією розбиття складних і довгих транзакцій на дрібніші, керовані кроки. Кожен мікросервіс у ланцюжку виконує свою локальну транзакцію, яка може бути атомарною і повертати систему до попереднього узгодженого стану при помилці.

Паттерн Saga зосереджується на досягненні атомарності кожного кроку транзакції у мікросервісній архітектурі. Атомарність означає, що кожна локальна транзакція має бути виконана успішно, або повністю відкатана, не допускаючи жодного проміжного стану. Це критично важливо для забезпечення надійності та цілісності даних у системі [17].

Уявіть, у нас є складна транзакція, яка включає кілька етапів, що виконуються у різних мікросервісах. Кожен етап може змінювати дані або виконувати інші операції. Якщо в якомусь мікросервісі відбувається помилка на одному з етапів, і ми не застосовуємо атомарність, то система залишиться в неконсистентному стані, що може призвести до непередбачуваної поведінки та втрати цілісності даних.

Тому коли ми говоримо про те, що кожен крок транзакції має бути атомарним, це означає, що ми повинні гарантувати, що кожна операція

всередині мікросервісу виконується повністю і коректно. І якщо в якомусь мікросервісі відбувається помилка, нам потрібно застосувати дії, що компенсують, щоб повернути систему в стан, який був до початку цього кроку.

Це забезпечує надійність та цілісність даних, тому що якщо щось йде не так на одному з кроків, ми можемо застосувати компенсуючі дії, щоб скасувати всі зміни, внесені на попередніх етапах, та повернути систему до попереднього узгодженого стану. Таким чином, ми уникаємо непередбачуваних станів і гарантуємо, що наша система завжди залишається в узгодженому стані, навіть у разі виникнення помилок.

Паттерн Saga надає ефективний механізм для керування транзакціями у мікросервісній архітектурі, гарантуючи атомарність кожного кроку та забезпечуючи надійність та цілісність даних. Це дозволяє розробляти більш надійні та відмовостійкі системи, полегшуючи обробку помилок та виняткових ситуацій. У результаті застосування патерна Saga сприяє підвищенню якості та стабільності нашого програмного забезпечення в умовах мікросервісної архітектури.

На рисунку 2.2 зображено приклад бізнес-процесу, який включає замовлення товару та його оплату. У випадку виявлення помилок, транзакції скасовують всі зміни та повертають стан бази даних до початкового положення.

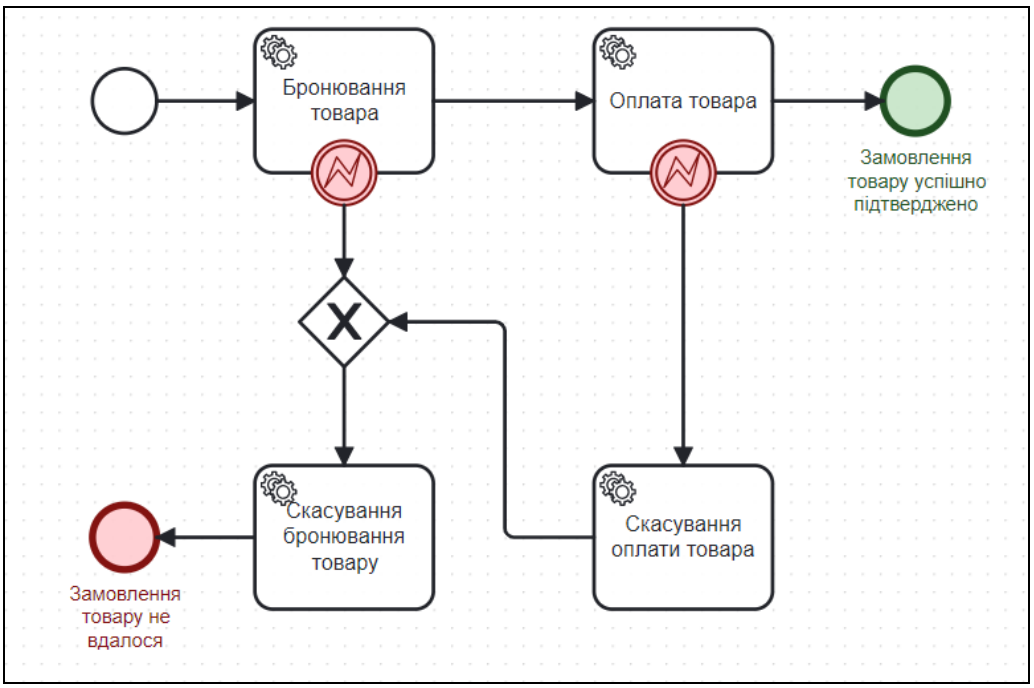


Рисунок 2.2 – BPMN–діаграма процесу замовлення товару та відображення компенсаційних транзакцій у разі виникнення помилок

2.1.3.1 Використання патерну SAGA: Оркестрація

Оркестрація підходу SAGA використовує централізований сервіс для організації транзакцій у розподілених системах. Цей сервіс також контролює та координує взаємодію між мікросервісами. Цей підхід схожий на диригента оркестру: кожен інструмент є мікросервісом, а диригент є центральним сервісом, який керує всім процесом.

У системі з оркестрацією центральний сервіс, також відомий як оркестратор, відправляє команди кожному сервісу, коли настає час виконати їхню частину процесу. Оркестратор запитує наявність товару на складі та розраховує його вартість, якщо він доступний. Оркестратор передає дані сервісу оплати після отримання цієї інформації, щоб він міг завершити покупку.

Оркестрація зменшує складність міжсервісної взаємодії та покращує здатність системи до швидкого відновлення після помилок, надаючи чіткий

контроль і централізоване управління процесами та потоком даних. Це гарантує високу ефективність і стабільність системи. Схема реалізації оркестрації в рамках SAGA представлена на рисунку 2.3.

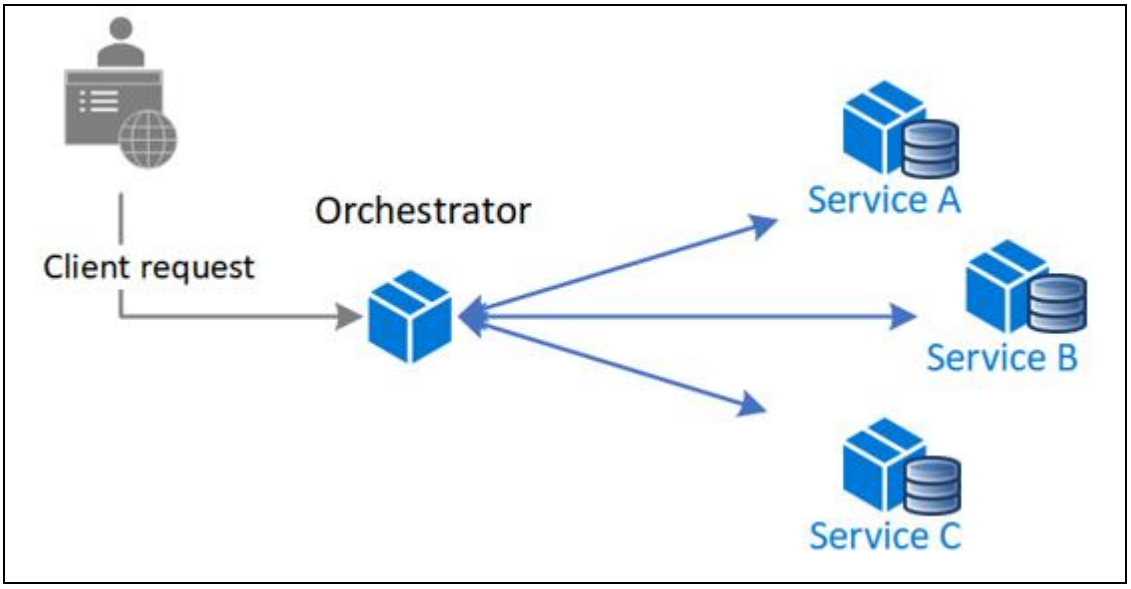


Рисунок 2.3 – Приклад SAGA підходу «Оркестрація»

2.1.3.2 Використання патерну SAGA: Хореографія

Хореографія патерна SAGA дозволяє керувати транзакціями в розподілених системах, таких як мікросервіси, де немає одного центрального елемента, який контролює все. Цей підхід нагадує танцювальний зал, де кожен танцюрист-мікросервіс у нашому випадку контролює свої власні рухи та реагує на рухи інших.

Уявімо ситуацію, коли ви замовляєте продукт через онлайн-магазин. У цьому процесі є кілька кроків:

Крок 1. Ви вибираєте товар.

Крок 2. Система перевіряє наявність товару на складі.

Крок 3. Ви вводите свої дані для доставки.

Крок 4. Система обраховує вартість доставки.

Крок 5. Ви підтверджуєте замовлення та проводите оплату.

У системі, яка використовує хореографію, кожен із цих кроків виконується різними сервісами, які взаємодіють між собою без центрального управління, а за допомогою брокерів повідомлень. Сервіси підписуються на події інших сервісів та відправляють повідомлення, коли потрібно ініціювати наступний крок процесу. Наприклад, коли сервіс складу виявляє, що товар наявний, він відправляє повідомлення сервісу доставки, щоб той розрахував вартість доставки. Потім сервіс доставки повідомляє сервіс оплати, що все готово для фінального підтвердження та оплати. На рисунку 2.4 можна побачити приклад реалізації цього підходу.

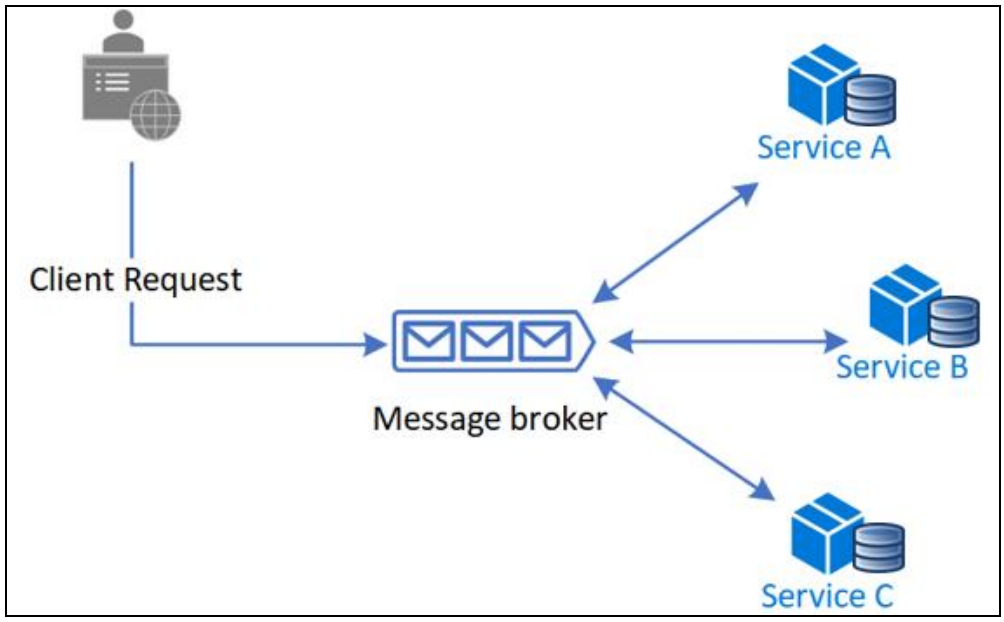


Рисунок 2.4 – Приклад SAGA підходу «Хореографія»

Для реаліцізації вебплатформи потрібно використовувати цей підхід, оскільки це обумовлює кількома ключовими причинами:

а) децентралізація контролю: хореографія дозволяє кожному мікросервісу самостійно вирішувати, коли і як реагувати на отримані події або повідомлення. Це зменшує залежність від центрального координуючого компонента (оркестратора), роблячи систему більш гнучкою і масштабованою;

б) мінімізація точок відмови: оскільки у хореографії не існує одного центрального компонента, який керує взаємодією всіх сервісів, ризик збою

всієї системи через проблеми в одному сервісі значно знижується. Це робить систему стійкішою до відмов;

в) спрощення масштабування: кожен мікросервіс у хореографії може бути незалежно масштабованим відповідно до його власних вимог та навантаження. Це важливо для систем, де окремі компоненти можуть мати різне навантаження, що змінюється у часі;

г) відповідність моделі подій: якщо система базується на обробці подій (наприклад, реакція на завантаження нових чеків чи їх обробка), хореографія може бути більш природною і ефективною моделлю, оскільки кожен сервіс реагує на події, не чекаючи команд від централізованого контролера;

д) зниження залежностей: у хореографічній моделі сервіси розробляються таким чином, щоб мінімізувати їхню залежність один від одного, що допомагає зменшити складність інтеграції та підтримки великої кількості сервісів.

2.2 Вибір оптимальної технології OCR для обробки фінансових документів

Перетворення зображень документів у машиночитаний текст є важливим у сучасному світі автоматизації бізнес-процесів. Багато сфер бізнесу потребують оптичного розпізнавання символів (OCR), особливо коли обробляються фінансові документи, такі як чеки, рахунки та інші важливі папери. Технологія OCR значно спрощує процеси обробки, знижує кількість помилок, пов’язаних з ручним введенням даних, і підвищує загальну продуктивність системи.

Оскільки якість подальшої обробки та аналізу інформації безпосередньо залежить від точності розпізнавання тексту, вибір найкращої технології OCR є критично важливим для успіху будь-якої системи обробки

документів. У цьому розділі ми розглянемо різні технології OCR, які зараз доступні на ринку, оцінивши їхні основні характеристики та те, наскільки вони можуть бути включені в мікросервісну архітектуру вебплатформи. Аналіз буде включати огляд передових практик, порівняльний аналіз їхніх переваг і недоліків, а також розгляд вимог до точності, швидкості обробки та можливості масштабування кожної технології. Це допоможе вибрати найкращу технологію, яка задовольнить особливі потреби в системі, а саме оптимізує процес обробки фінансових документів [18].

У цьому розділі розглянемо дві найбільш популярні на сьогодні бібліотеки OCR (Google Vision API та Tesseract OCR), зроблю оцінку про їхні ключові характеристики та потенціал для інтеграції у мікросервісну архітектуру вебплатформи.

2.2.1 Google Cloud Vision API

Google Vision API використовує передові алгоритми машинного навчання та нейронних мереж для аналізу зображень, що дозволяє ідентифікувати текстовий контент із фотографій, сканів та інших візуальних даних. API під капотом використовує моделі глибокого навчання, які треновані на великих даних зображень. Це дозволяє точно визначати текстові фрагменти на зображеннях і інтерпретувати їх у відповідному мовному середовищі [19].

2.2.1.1 Робота OCR у Google Vision API

Оптичне розпізнавання символів (OCR) у Google Vision API діє на основі розпізнавання зразків тексту. API аналізує структуру зображення, ідентифікуючи області, які містять текст, а потім застосовує алгоритми

розпізнавання для перетворення зображення тексту у цифровий текстовий формат. Окремі символи розпізнаються та об’єднуються у слова та речення з урахуванням контексту та мовних особливостей. Основні переваги цього API такі:

а) точність: Google Vision API використовує передові технології машинного навчання для точного розпізнавання тексту;

б) гнучкість і масштабованість: API масштабується залежно від потреб користувача, дозволяючи обробляти великі кількості даних;

в) швидкість обробки: Швидкість обробки даних дозволяє миттєво розпізнавати текст, що важливо для динамічних застосунків;

г) підтримка багатьох мов: API підтримує багато мов, що робить його ідеальним для застосунків за кордоном. На рисунку 2.5 а) наведено приклад застосування із словацькою мовою, 2.5 б) з німецькою мовою та 2.5 в) з українською мовою.

а) б) в)

Рисунок 2.5 – Варіанти аналізу чеків на різних мовах за допомогою Google Vision API

а) словацька мова; б) німецька мова; в) українська мова

2.2.2 Tesseract OCR

Tesseract OCR є одним із найвідоміших відкритих програм для оптичного розпізнавання символів. Tesseract, який спочатку був

розроблений HP, а потім розширений за підтримки Google, широко використовується в сучасному світі завдяки своїй здатності розпізнавати текст на понад 100 мовах.

Для розпізнавання текстів Tesseract використовує комбінований підхід, який включає класичні та нейронні мережеві технології. Його робота складається з аналізу зображень на наявність тексту, розпізнавання символів у зображеннях і перетворення цих символів у слова та рядки тексту. Обмеження Tesseract порівняно з Google Vision API:

а) точність розпізнавання: незважаючи на широкі можливості, Tesseract часто поступається за точністю Google Vision API, особливо при роботі з нестандартними шрифтами або погано освітленими зображеннями;

б) потреба в попередній обробці зображень: для досягнення оптимальних результатів з Tesseract, зображення часто потребують ретельної попередньої обробки (наприклад, корекція освітлення, вирівнювання та видалення шумів), тоді як Google Vision API ефективніше справляється з «сирими» зображеннями;

в) обмежена багатомовна підтримка: хоча Tesseract підтримує багато мов, він може виявитися менш ефективним при розпізнаванні мов із складнішими гліфами або правописними особливостями, на відміну від Google Vision API, який краще адаптований до глобальних вимог;

г) масштабування та інтеграція: оскільки Tesseract не надає готових хмарних рішень, його інтеграція та масштабування в більші системи може вимагати додаткових ресурсів і налаштувань. У порівнянні, Google Vision API легко масштабується завдяки вбудованій хмарній інфраструктурі.

Хоча Tesseract OCR залишається важливим інструментом у сфері оптичного розпізнавання символів, особливо для розробників з обмеженим бюджетом або потребою у відкритому програмному забезпеченні, він має певні обмеження, що робить Google Vision API більш привабливим варіантом для комерційного використання та великомасштабних

застосувань. Приклад використання Tesseract OCR можна побачити на рисунку 2.6.

Рисунок 2.6 – Приклад застосування Tesseract OCR для знаходження тексту з квитанції

2.3 Вибір оптимальної технології LLM для обробки фінансових документів

Великі мовні моделі (LLM) стають все більш популярними завдяки прогресу в галузі штучного інтелекту. Це пов’язано з тим, що вони можуть ефективно аналізувати, інтерпретувати та витягувати корисні дані з текстових даних. Це особливо важливо для автоматизації фінансових документів, де швидкість і точність мають вирішальне значення [20].

У цьому розділі розглядаються наявні великі мовні моделі, а також їхні можливості та обмеження щодо обробки фінансових документів. Мета полягає в тому, щоб визначити, які технології найкраще підходять для створення мікросервісної вебплатформи, яка дозволить забезпечити високу якість і точність обробки, а також автоматизувати процес розпізнавання та аналізу даних.

Буде проведено порівняльний аналіз лідерів у цій галузі, включаючи такі платформи як OpenAI GPT, Gemini та інші сучасні мовні моделі. Розглядаються такі аспекти, як адаптивність моделей до особливостей фінансових документів, те, наскільки легко їх використовувати та наскільки легко їх використовувати в реальних бізнес–процесах.

У рамках даного дослідження мною будуть використані промпти, спеціалізовані для мовних моделей, які здатні аналізувати та структурувати дані з чеків у деталізований формат JSON [21]. Цей процес включатиме екстракцію важливої інформації з текстових даних чеків і їх подальше форматування в такі категорії:

– store: назва магазину;
– country: країна, яку потрібно визначити з міста, згаданого у чеку;
– city: місто розташування магазину;
– address: повна адреса магазину;
– total_amount: загальна сума покупок;
– currency: код валюти у форматі ISO 4217;
– datetime: дата та час покупки у форматі ISO 8601;
– items: масив предметів покупки, кожен з яких містить опис, кількість, ціну за одиницю, та загальну вартість.

На рисунку 2.7 а) представлено зразок чеку з чіткою структурою, що демонструє як мовна модель вилуче інформацію у стандартних умовах. Рисунок 2.7 б) показує чек без структури, ілюструючи виклики, які виникають при роботі з неорганізованими даними. Нарешті, на рисунку 2.7 в) відображено пошкоджений чек, на якому видно, як модель стикається з труднощами через фізичне пошкодження або погіршення якості зображення. Будуть розглянуті дві популярні мовні моделі станом на сьогоднішній день, для визначення, яка з них забезпечує більш якісний та коректний результат. Результати цього порівняння допоможуть вибрати найбільш ефективну технологію для подальшої реалізації [22].

а) б) в)

Рисунок 2.7 – Зразки чеків, використані для перевірки великих мовних моделей

а) чек з гарною структурою; б) чек без структури; в) пошкоджений чек

2.3.1 OpenAI GPT

Завдяки своїй здатності до глибокого навчання на великих кількостях текстових даних і здатності генерувати зміст, що відповідає контексту, моделі OpenAI GPT стали еталоном у галузі обробки природної мови. Ця особливість робить їх особливо привабливими для автоматизації фінансових документів, де точність і контекстуальна релевантність інформації мають вирішальне значення.

API ChatGPT надає розробникам інструменти для інтеграції цих мовних моделей у різні застосунки, вебсайти та інші сервіси. Ось деякі ключові аспекти, пов’язані з API:

а) ціни: OpenAI пропонує декілька тарифних планів для використання ChatGPT, що включають як безкоштовний варіант із обмеженою кількістю запитів, так і платні опції, які розширюють кількість можливих запитів та доступ до додаткових функцій. Ціни можуть

варіюватися залежно від кількості згенерованих символів або відповідей, що робить цю платформу гнучкою з точки зору вартості для стартапів та великих компаній;

б) гнучкість: ChatGPT можна інтегрувати в чат-боти, цифрових помічників, інструменти для автоматизації відповідей на запитання клієнтів тощо;

в) масштабованість: API підтримує високу масштабованість, що дозволяє впоратися з великою кількістю запитів без втрати продуктивності;

г) багатомовність: підтримка багатьох мов розширює можливості використання ChatGPT для міжнародних ринків;

д) постійні оновлення: OpenAI регулярно оновлює моделі, щоб покращити точність і реалістичність відповідей;

е) використання: API ChatGPT може бути використаний для різноманітних завдань, включаючи автоматизацію обслуговування клієнтів, створення контенту, освітні застосунки, і навіть для допомоги у технічній підтримці та розробці [23].

На рисунку 2.8 а), б) та в) показано результат обробки документів з різною структурою моделю GPT 3.5 та Google Vision OCR:

а) б) в)

Рисунок 2.8 – Результати обробки чеків у форматі JSON за допомогою GPT 3.5 та Vision API

а) чек з гарною структурою; б) чек без структури; в) пошкоджений чек

Як було встановлено у дослідженні, GPT 3.5 та Google Vision OCR ефективно витягують основну інформацію з чеків, таку як дата, час, місто, та назву магазину. Зокрема, Google Vision виявляється ефективним навіть у випадках з пошкодженими чеками, адаптуючись до умов і намагаючись виокремити потрібний текст з зображення. Проте обидві технології стикаються з труднощами при обробці товарів з чеків, що мають нечітку структуру, призводячи до неточностей у визначенні ціни за товари.

2.3.2 Gemini Pro

У цьому розділі ми розглянемо найновішу технологію обробки даних від Google, Gemini Pro. Цей програмний продукт є одним із найбільш передових у галузі оптичного розпізнавання символів (OCR), який поєднує традиційні методи OCR із найсучаснішими технологіями штучного інтелекту та машинного навчання.

Gemini Pro покращує точність і продуктивність обробки текстової інформації з великої кількості документів, включаючи фінансові документи та чеки. Ця технологія може аналізувати великі кількості даних з високою точністю розпізнавання тексту навіть у випадках, коли текст представлений у складних форматах або має низьку якість [24].

Розглянемо результати обробки чеків за допомогою Gemini Pro 1.5, які представлені на рисунках 2.9 а), б) та в):

```
{
  "store": "ЄВРАЗІЯ",
  "country": "Ukraine",
  "city": "Київ",
  "address": "вул. Рогнідинська, 5/14",
  "total_amount": 251.2,
  "currency": "UAH",
  "datetime": "2024-04-28T13:19:00+03:00",
  "items": [
    {
      "description": "Свинина з овочами",
      "quantity": 1,
      "unit_price": 175,
      "total_price": 175
    },
    {
      "description": "Грецький салат",
      "quantity": 1,
      "unit_price": 160,
      "total_price": 160
    },
    {
      "description": "Карта 25%",
      "quantity": 1,
      "unit_price": -83.8,
      "total_price": -83.8
    }
  ]
}
```

```
{
  "store": "Магазин ПІДКОВА",
  "country": "Ukraine",
  "city": "Гриців",
  "address": "вул. Центральна, 5",
  "total_amount": 314.82,
  "currency": "UAH",
  "datetime": "2024-03-24T16:59:42+02:00",
  "items": [
    {
      "description": "Вода Регіна",
      "quantity": 1,
      "unit_price": 15,
      "total_price": 15
    },
    {
      "description": "Тіст. Напол еон ябл. Бі скотті",
      "quantity": 0.642,
      "unit_price": 211.99,
      "total_price": 136.1
    },
    {
      "description": "Зерн. Алмаз схід 80г",
      "quantity": 3,
      "unit_price": 23,
      "total_price": 69
    },
    {
      "description": "Кар-ль Мент олевк. Рош ем",
      "quantity": 0.53,
      "unit_price": 124,
      "total_price": 65.72
    },
    {
      "description": "Шок.бат. РО ШЕН мигд. ко кос 38 г",
      "quantity": 2,
      "unit_price": 14.5,
      "total_price": 29
    }
  ]
}
```

```
{
  "store": "BENU SK 151, a.s.",
  "country": "Slovakia",
  "city": "Bratislava",
  "address": "Metodova 8 82108 Bratislava",
  "total_amount": 19.18,
  "currency": "EUR",
  "datetime": "2024-04-06T15:29:51+02:00",
  "items": [
    {
      "description": "Tussirex sirup 1x120 ml",
      "quantity": 1,
      "unit_price": 9.99,
      "total_price": 9.99
    },
    {
      "description": "Neo angin bez cukru",
      "quantity": 1,
      "unit_price": 9.19,
      "total_price": 9.19
    }
  ]
}
```


а) б) в)

Рисунок 2.9 – Результат обробки чека у форматі JSON за допомогою Gemini Pro 1.5

а) чек з гарною структурою; б) чек без структури; в) пошкоджений чек

Як показали результати тестування, Gemini Pro відмінно справляється з обробкою документів, коректно розпізнаючи інформацію з усіх типів чеків. Однією з ключових особливостей Gemini Pro є її здатність до глибокої інтеграції з іншими сервісами і платформами, що дозволяє розробникам вбудовувати функціонал OCR безпосередньо у власні застосунки. Це включає інтеграцію з системами управління документообігом, фінансовими платформами та іншими бізнес-застосунками.

3 СТАДІЇ РОЗРОБКИ МІКРОСЕРВІСНОЇ ВЕБПЛАТФОРМИ ДЛЯ РОЗПІЗНАВАННЯ ФІНАНСОВИХ ДОКУМЕНТІВ

3.1 Обґрунтування технологій для розробки вебплатформи

У рамках кваліфікаційної роботи було зосереджено увагу на розробці архітектури вебплатформи для ефективної обробки фінансових документів. Завдання полягало в створенні детально спланованої структури, яка була б масштабованою та відповідала сучасним вимогам технологічної індустрії. Особлива увага приділялася забезпеченню гнучкості та адаптивності системи, що є критично важливим для успішного проєктування.

На першому етапі було вивчено, як можна інтегрувати мікросервіси в єдину систему, здатну ефективно обробляти великі обсяги даних, забезпечуючи при цьому їхню безпеку і цілісність. Різноманітні підходи до проєктування, які розглядалися, включають масштабування та управління окремими сервісами для підтримки високої продуктивності при змінних навантаженнях.

Додатково, були розроблені процедури впровадження та тестування компонентів системи. Це забезпечило стабільність і надійність роботи платформи під час виконання різноманітних бізнес-операцій. Такий підхід дозволяє не тільки вирішувати поточні завдання, але й адаптуватися до майбутніх викликів у цій швидко змінюваній галузі.

3.1.1 Опис стейт-машини з використанням BPMN

Стейт-машини, або автомати зі станами, є фундаментальною концепцією в теорії обчислень та програмуванні, що дозволяє моделювати поведінку систем через визначені стани та переходи між ними. Ці машини використовуються для дизайну та аналізу програмних та інженерних

систем, де поведінка може бути ясно описана як ряд станів з переходами, зумовленими подіями або умовами.

У контексті розробки мікросервісної вебплатформи для обробки фінансових документів, стейт-машини використовуються для моделювання та керування бізнес-процесами, такими як обробка чеків. Діаграма BPMN (Business Process Model and Notation) [25], представлена в цьому розділі, є прикладом такого використання, де кожен крок процесу описаний як частина стейт машини. Процес включає такі етапи:

Крок 1. Завантаження чеку (рис. 3.1) – це початковий крок, де користувач завантажує чек для обробки. Система перевіряє, чи було завантаження успішним, та відповідає клієнту успіхом або невдачею.

Крок 2. Сканування та екстракція даних (рис. 3.2) – після успішного завантаження чеку, документ сканується, і з нього екстрагуються дані.

Крок 3. Аналізування даних у LLM (рис. 3.3) – після сканування та екстракції, екстраговані дані аналізуються за допомогою великої мовної моделі (LLM). Цей крок включає подальше уточнення та валідацію даних..

Крок 4. Перевірка даних клієнтом (рис. 3.4) – екстраговані дані представляються користувачу для перевірки. На цьому етапі користувач має можливість модифікувати, відхилити або затвердити отриману інформацію.

Крок 5. Підтвердження транзакції (рис. 3.5) – одразу після затвердження користувачем даних, система вносить інформацію до бази даних, створюючи нові транзакції за вказаною користувачем категорією. Цей крок гарантує, що всі зміни збережені та категоризовані відповідно до вибору користувача, що забезпечує належне ведення фінансового обліку.

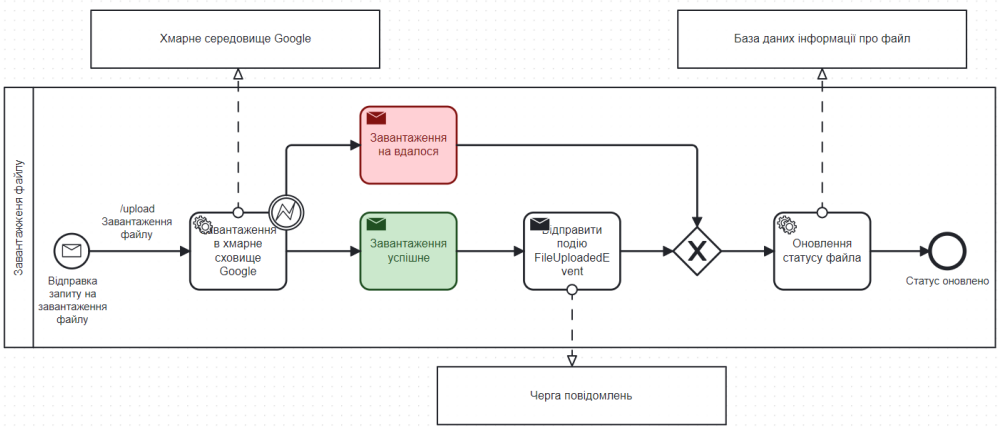


Рисунок 3.1 – Бізнес-процес завантаження файлу у хмарне середовище Google

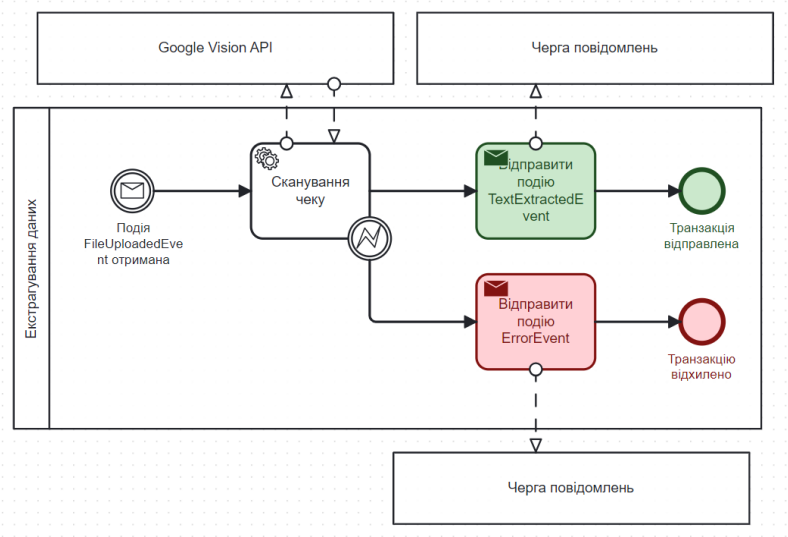


Рисунок 3.2 – Бізнес-процес сканування чеку за допомогою Google Vision API

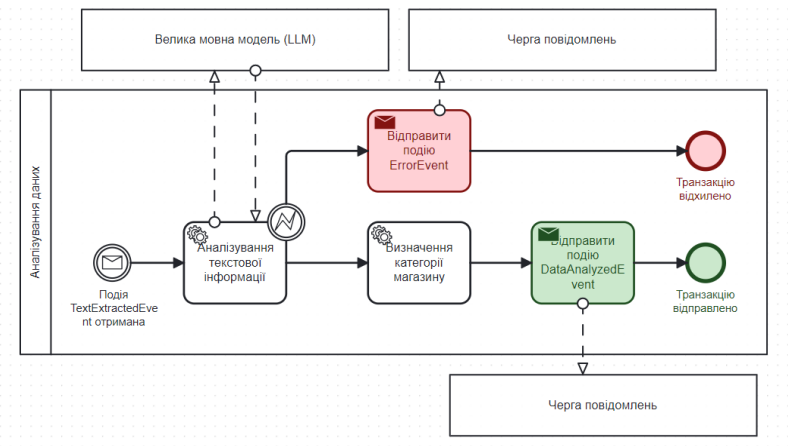


Рисунок 3.3 – Бізнес-процес аналізування даних за допомогою LLM

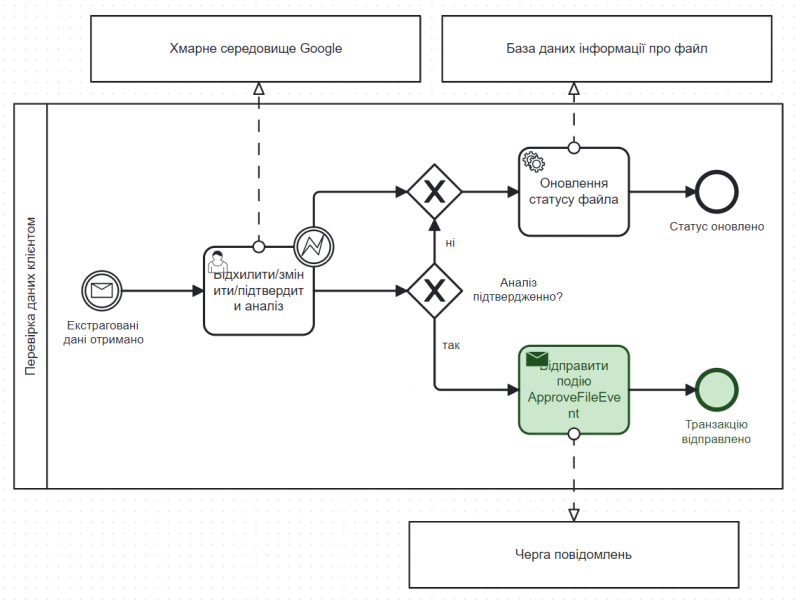


Рисунок 3.4 – Бізнес-процес перевірки даних клієнтом

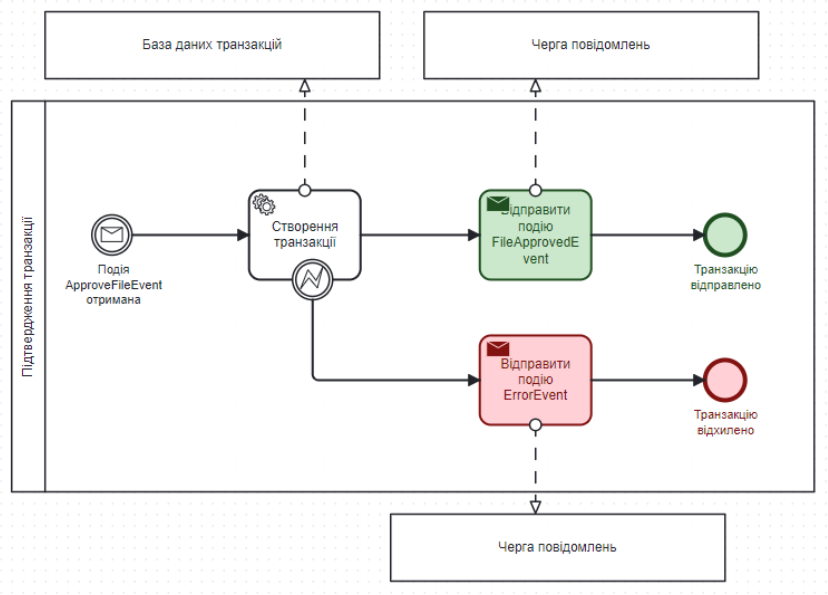


Рисунок 3.5 – Бізнес-процес затвердження транзакції

Цей процес відображає як моделювання стейт-машини може бути ефективно використане для автоматизації та оптимізації бізнес-процесів у мікросервісній архітектурі, забезпечуючи високий рівень адаптивності та масштабованості необхідних для сучасних вебплатформ [26].

3.1.2 Черги повідомлень для взаємодії між мікросервісами

Apache Kafka та RabbitMQ – це дві платформи для обміну повідомленнями з відкритим кодом, засновані на підписках, які часто інтегруються в різні проєкти. RabbitMQ, заснований у 2007 році, довгий час був ключовим елементом в архітектурах, заснованих на обміні повідомленнями та SOA, і досі використовується для потокової обробки даних. З іншого боку, Kafka, запущений у 2011 році, був розроблений спеціально для потокових даних з самого початку.

RabbitMQ функціонує як універсальний брокер повідомлень (рис. 3.6), підтримуючи протоколи MQTT, AMQP та STOMP, і здатний справлятися з високонавантаженими задачами, такими як обробка онлайн-платежів. Цей інструмент може також керувати асинхронними завданнями в фоновому режимі або слугувати як зв'язок між мікросервісами.

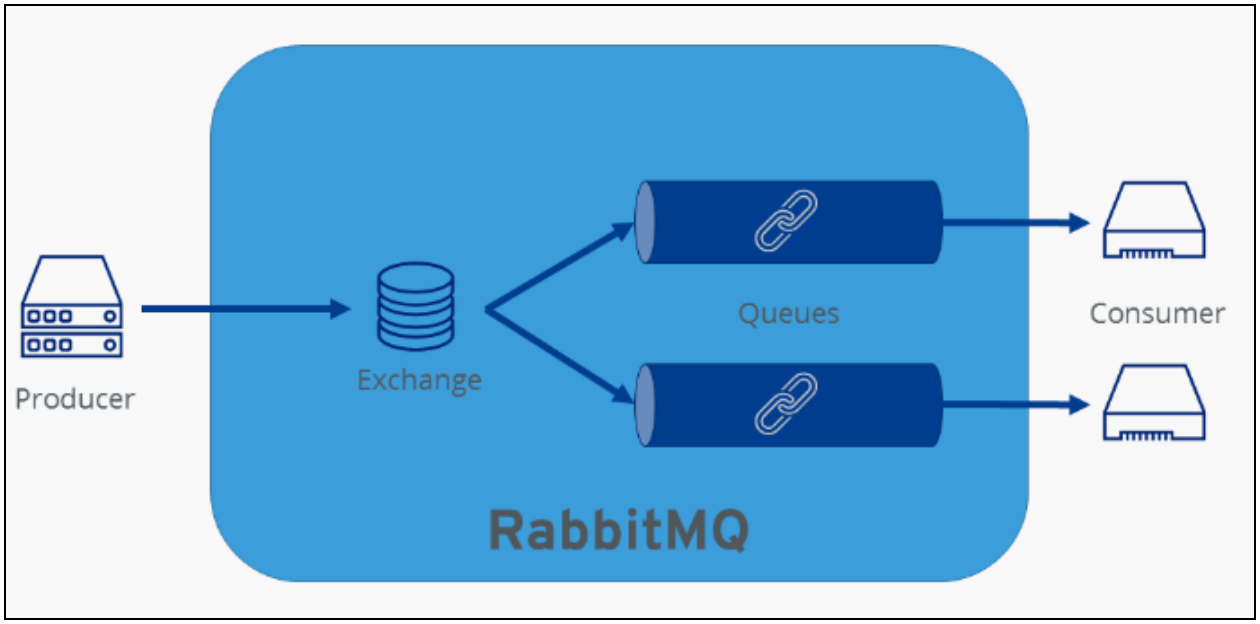


Рисунок 3.6 – Схема роботи брокера повідомлень RabbitMQ

Kafka, у свою чергу, була спеціально розроблена для оптимізації потокової передачі даних, використовуючи модель публікації-підписки, яка дозволяє ефективно масштабуватися та обробляти великі обсяги даних (рис.

3.7). Завдяки своїм властивостям, Kafka ідеально підходить для систем, де потрібна швидка обробка та здатність до витривалості в умовах великої кількості транзакцій [29].

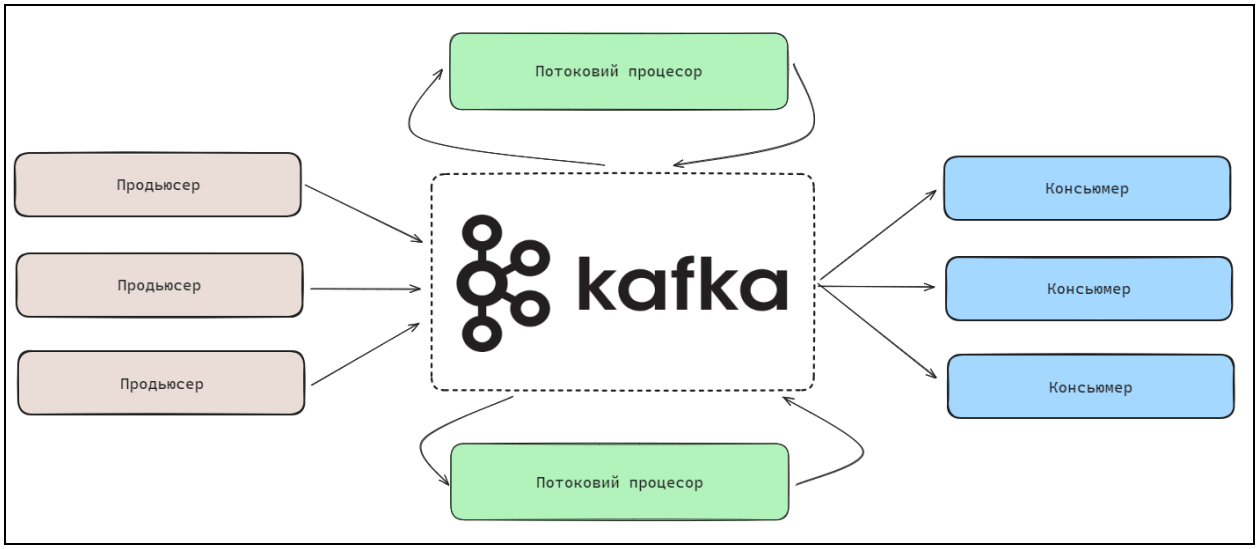


Рисунок 3.7 – Схема роботи брокера повідомлень Kafka

Це дозволяє зробити висновок про необхідність застосування Kafka у наступних випадках:

– значна кількість отримувачів одного повідомлення;

– велика пропускна здатність, що досягає мільйонів повідомлень за секунду;

– необхідність потокової обробки;

– можливість резервного копіювання черг;

– висока надійність системи;

– важливість збереження порядку повідомлень.

RabbitMQ використовується в ситуаціях, коли необхідно:

– один отримувач на повідомлення;

– гнучкість маршрутизації;

– черга з пріоритетами;

– використання стандартних протоколів повідомлень.

Результати дослідження показують, що продуктивність RabbitMQ значно знижується під час надмірного навантаження. Реплікація черг також впливає на продуктивність при високих пропускних здатностях. Використання базових налаштувань без додаткового дзеркалювання дало найкращі показники затримки.

Зокрема, порівняння результатів за ключовими параметрами представлено в таблиці 3.1:

Таблиця 3.1 – Аналіз пропускної здатності

	Kafka	**RabbitMQ**
Пропускна здатність	605 МБ/с	38 МБ/с
P99 затримка	5 мс	1 мс

3.1.3 Мова програмування для мікросервісів

Вибір мови програмування, яка буде використана для написання сервісів, є важливим вибором під час розробки мікросервісної архітектури. Мова повинна бути легкою для масштабування, високопродуктивною та здатною забезпечити сильні інструменти для управління залежностями та розподіленими системами. В цьому випадку великою перевагою буде використання Go (або Golang) для розробки мікросервісів, зокрема його придатність для цього типу архітектури.

а) простота та читабельність: Go має чіткий синтаксис, який спрощує написання та зрозуміння коду, що є важливим для великих команд розробників і швидкого впровадження нових інженерів у проєкт;

б) конкурентність: мова включає в себе примітиви конкурентності, такі як горутини та канали, що дозволяють легко реалізувати асинхронну обробку даних і ефективне використання багатоядерних процесорів для високопродуктивних операцій;

в) швидкість виконання: програми, написані на Go, компілюються в машинний код, що забезпечує високу швидкість виконання та ефективність, порівнянну з такими мовами, як C++;

г) вбудована підтримка мікросервісів: Go має розширені можливості для роботи з мережею та розподіленими системами, зокрема підтримку RESTful API та легкість створення мікросервісів з використанням різних фреймворків, таких як Gorilla Mux та Gin;

д) спільнота та інструменти: Golang має велику та активну спільноту розробників, а також набір потужних інструментів для розробки, моніторингу та розгортання застосунків. Це спрощує роботу з кодом та його відладку, а також забезпечує додаткові ресурси для навчання та підтримки.

3.1.4 Контейнеризація за допомогою Docker

Контейнеризація відіграє ключову роль у розвитку сучасних мікросервісних архітектур, оскільки дозволяє ізолювати сервіси на рівні операційної системи, що сприяє легкій віртуалізації. Такий підхід дозволяє запускати застосунки разом із необхідними системними бібліотеками в уніфікованому контейнері, гарантуючи стандартизацію виконання незалежно від розгортального середовища.

Ця технологія являється передовою тенденцією у розробці програмного забезпечення, пропонуючи альтернативу традиційній віртуалізації. Вона інкапсулює програмний код та всі його залежності у пакет, що забезпечує його консистентне виконання на будь-якій інфраструктурі. Швидкий розвиток цієї технології принесло значні переваги як розробникам, так і оперативним командам, підвищуючи ефективність інфраструктури.

Коли мова заходить про контейнеризацію, часто на думку приходить Docker – технологія з відкритим вихідним кодом, яка використовується для

розробки, тестування, розгортання та запуску вебзастосунків у контейнеризованих середовищах. Docker забезпечує більш ефективне використання системних ресурсів, дозволяє швидко розгортати готові програмні рішення, а також масштабувати та переносити їх між різними середовищами, забезпечуючи стабільність роботи (рис. 3.8). Завдяки широкому розповсюдженню, Docker став практично синонімом поняття «контейнер» [30].

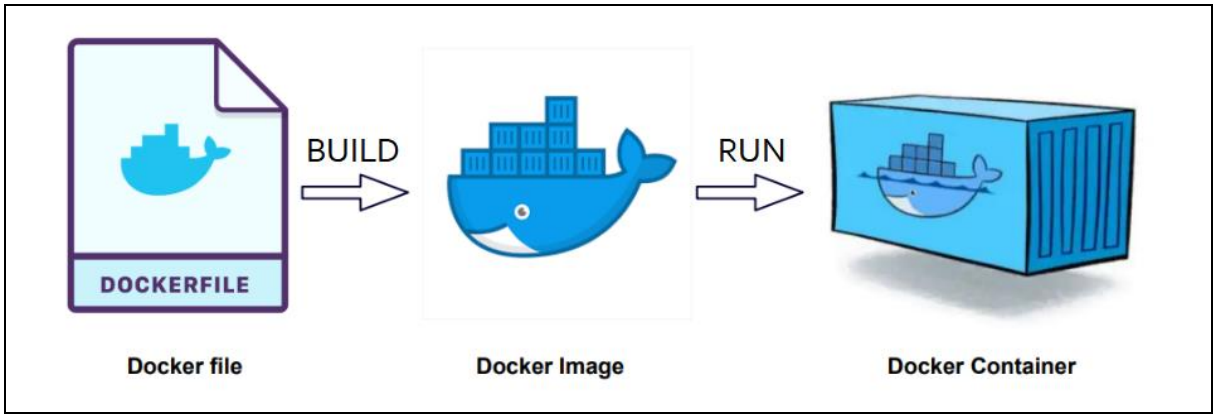


Рисунок 3.8 – Схема принципу роботи Docker

Запуск Docker потребує лише декілька системних викликів, після яких операційна система створює ізольований простір для нового процесу, включаючи відокремлену віртуальну мережу та обмеження ресурсів. Процес, який «запущений у Docker», насправді виконується на тій самій фізичній машині за допомогою того ж ядра ОС, не вимагаючи віртуальної машини або будь-якого емулятора апаратного забезпечення.

Docker Compose є невід'ємною частиною екосистеми Docker, що дозволяє описати та управляти множинними контейнерами за допомогою одного конфігураційного файлу. Цей інструмент ідеально підходить для організації мікросервісних архітектур, забезпечуючи простоту конфігурації та відмінну інтеграцію з Docker.

Привілегії використання Docker Compose включають можливість визначення, оркестрації та масштабування сервісів одночасно за допомогою

простих команд. Він дозволяє керувати циклом життя цілого застосунку, включно з його створенням, запуском та зупинкою через єдиний інтерфейс.

Давайте розглянемо приклад файлу docker–compose.yml для інтеграції Kafka та декількох мікросервісів. Файл декларує не тільки окремі сервіси, але й їхні залежності та мережеві налаштування:

а) Zookeeper і Kafka: ці сервіси ініціюються з використанням офіційних образів Bitnami. Zookeeper виступає у ролі координації для Kafka, який є розподіленим системом обміну повідомленнями. Параметри конфігурації дозволяють налаштувати Kafka на роботу в середовищі Docker;

б) бази даних: використання Postgres як бази даних для кожного сервісу з власними налаштуваннями забезпечує ізоляцію даних між мікросервісами. Волюми дозволяють зберігати дані незалежно від контейнерів;

в) мікросервіси: кожен мікросервіс, такий як сервіс обробки чеків, сервіс візуалізації, транзакційний сервіс та інші, інтегрується в систему з власними налаштуваннями. Вони включають побудову Docker образів з локальних Dockerfile та залежність від відповідних баз даних.

Docker Compose дозволяє централізовано управляти всіма цими компонентами, забезпечуючи єдиний точку входу для розгортання та масштабування. Такий підхід значно спрощує локальну розробку та тестування, дозволяючи розробникам зосередитись на інноваціях, а не на управлінні інфраструктурою.

3.2 Реалізація мікросервісної вебплатформи

На основі проведеного аналізу та обґрунтування вибору технологій у попередньому розділі, можна приступати до безпосередньої реалізації мікросервісної вебплатформи. Всі технології були ретельно обрані для забезпечення максимальної ефективності, масштабованості та надійності

системи. У цьому розділі буде описано процес розробки вебплатформи, що включає проєктування архітектури, інтеграцію обраних технологій, створення інтерфейсу користувача та забезпечення належного функціонування кожного мікросервісу.

Першочергове завдання полягає у створенні стабільної та масштабованої архітектури, яка буде здатна обробляти великі обсяги даних та підтримувати високий рівень продуктивності. Для цього будуть використані сучасні інструменти та фреймворки, такі як Docker для контейнеризації, Kafka для обміну повідомленнями між мікросервісами та Golang для розробки мікросервісів. Особлива увага приділяється розробці користувацького інтерфейсу з використанням React, Next.js та TailwindCSS для створення зручного та інтуїтивно зрозумілого UI.

3.2.1 Опис технічної архітектури мікросервісної вебплатформи

Основним компонентом розробки даної системи є проєкт мікросервісної вебплатформи, яка використовує OCR і LLM для інтелектуального розпізнавання та категорії витрат на основі інформації з фінансових документів.

Перед тим, як перейти до детального опису архітектури, слід зазначити, що мікросервісна архітектура є модульним підходом до розробки програмного забезпечення, де програма розділяється на окремі незалежні сервіси, які взаємодіють один з одним. Цей метод дозволяє зробити розробку, масштабування та підтримку системи більш простими. Вебплатформа складатиметься з наступних основних елементів архітектури:

а) вебклієнт: це передня частина інтерфейсу, через який користувачі взаємодіють із системою. Він забезпечує зручне середовище, де користувачі можуть завантажувати чеки, переглядати оброблені результати та управляти

своїми фінансовими документами. Клієнт спілкується із backend-сервісами через API-виклики;

б) сервіс файлів: сервіс відповідальний за управління всіма операціями, пов’язаними з файлами, включаючи завантаження, зберігання та отримання зображень фінансових документів. Він забезпечує, що файли безпечно зберігаються і ефективно управляються;

в) сервіс розпізнавання оптичних символів (OCR): цей сервіс обробляє зображення фінансових документів для отримання текстової інформації за допомогою технології OCR. OCR-сервіс може бути реалізований на основі сторонніх бібліотек або API, таких як Tesseract або Google Cloud Vision;

г) сервіс транзакцій: після підтвердження транзакції користувач реєструє транзакції, класифікує їх відповідно до налаштувань користувача або визначених правил, а також оновлює фінансові записи користувача відповідно до цих налаштувань;

д) черга повідомлень: ця черга дозволяє сервісам обмінюватися повідомленнями асинхронно через платформу. Він керує темами та чергами, щоб розділити сервіси та покращити масштабованість і відмовостійкість, забезпечуючи плавний потік даних через систему.

Кожен сервіс працює самостійно (рис. 3.9), але взаємодіє через чіткі API, зазвичай через HTTP або інтерфейси на основі повідомлень, що дозволяє безперешкодно взаємодіяти та інтегруватися. Обробляючи пікові навантаження та розподіляючи повідомлення між сервісами, використання посередника повідомлень зокрема сприяє підтримці реактивності та ефективності системи.

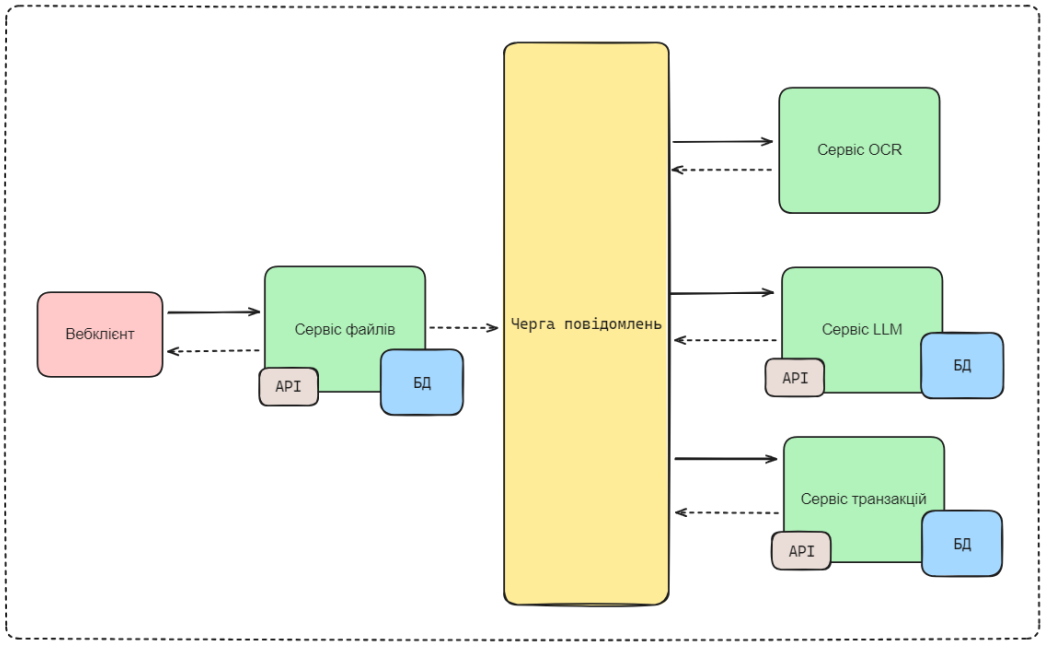


Рисунок 3.9 – Архітектура мікросервісної вебплатформи «Financier»

Ця архітектура не тільки підтримує динамічну природу обробки фінансових документів, але й адаптується до змінних умов навантаження та запитів користувачів, що є вирішальним для обробки даних у реальному часі та аналізу у надійний та масштабований спосіб.

3.2.2 Інтеграція клієнтського інтерфейсу з мікросервісною архітектурою

Ефективна взаємодія між клієнтами та мікросервісами є ключовою для безперебійного функціонування сучасних програмних систем. Центральним елементом цієї взаємодії є ретельне планування та виконання стратегії API. API (інтерфейс прикладного програмування) служить каналом, через який клієнти отримують доступ і взаємодіють з мікросервісами, що робить його стрижнею в архітектурі розподілених систем. Тому розробка добре продуманої стратегії API є обов’язковою для забезпечення не лише функціональності, але й довговічності та адаптивності програмної екосистеми.

Структура API розроблена таким чином, щоб оптимізувати взаємодію між фронтендом та бекендом, забезпечуючи ефективну обробку та відправку запитів. Вона включає в себе декілька основних клієнтів, які відповідають за різні аспекти обробки інформації:

а) File Client: відповідає за управління чеками, включаючи їх завантаження, отримання інформації та управління статусами;

б) LLM Client: займається взаємодією з аналізованми даними за допомогою LLM з фінансових документів;

в) Transaction Client: управляє фінансовими транзакціями, що базуються на обробленій інформації з фінансових документів.

Існують два поширені підходи до архітектури API – REST та SOAP. Ось їхні основні відмінності:

– простий протокол доступу до об’єктів (SOAP): офіційний протокол із суворими правилами. Він працює з протоколами прикладного рівня, такими як HTTP, UDP і SMTP, вимагає детальних договорів API та включає вбудовані механізми безпеки, обробки помилок та авторизації. Використовує об’ємний формат даних XML, що споживає більше пропускної здатності;

– передавання репрезентативного стану (REST): гнучкий стиль архітектури з декількома загальними рекомендаціями. REST працює лише з HTTP, дозволяє кешування запитів і не потребує детальних контрактів. Безпека, обробка помилок та авторизація залишаються на розсуд розробників. REST підтримує різні формати даних, включаючи JSON, XML, HTML та простий текст, і відомий своєю високою продуктивністю, масштабованістю та гнучкістю. Зазвичай REST використовується для веб та мобільних застосунків.

На даний момент, REST є найпопулярнішим підходом для створення API, охоплюючи понад 70% публічних API. Він славиться своєю простотою в роботі, кращою продуктивністю та масштабованістю. Отже, у таблиці 3.2 наведено структуру API для взаємодії із мікросервісами File Service, LLM

Service та Transaction Service, яка забезпечує ефективну комунікацію та інтеграцію різних компонентів системи.

Таблиця 3.2 – Опис API клієнтів

API	**Метод**	**Точка доступу**	**Опис**	**Відповідь**
File Client	GET	/receipts	Завантажує всі документи зібрані в системі.	Масив документів з даними.
	POST	/receipts	Завантаження нового документа у систему.	Статус завантаження.
	POST	/receipts /{id}/approve	Затвердження інформації документа, вказаного за ID.	Статус затвердження.
	POST	/receipts /{id}/reject	Відхилення документа.	Статус відхилення.
LLM Client	GET	/receipts/{id}	Завантажує детальну інформацію про документ.	Дані документа.
Transaction Client	GET	/transactions	Завантажує інформацію про всі транзакції в системі.	Деталі транзакцій.

3.2.3 Інтеграція Kafka за допомогою Golang

В сучасних мікросервісних архітектурах ключовим аспектом є ефективна обробка та маршрутизація повідомлень. У цьому контексті,

використання Apache Kafka разом з мовою програмування Golang є ідеальним рішенням для забезпечення високої продуктивності та надійності в обробці потокових даних. Бібліотека Sarama, розроблена для Golang, надає потужний інструментарій для взаємодії з Kafka, підтримуючи розширені можливості цієї системи обробки повідомлень.

Sarama – це клієнтська бібліотека Kafka для мови програмування Go. Вона підтримує повний функціонал Kafka та забезпечує зручний API для реалізації продюсерів та консюмерів. Sarama дозволяє детально налаштовувати поведінку черги, включно з управлінням партиціями та збереженням стабільності обробки повідомлень у масштабованій системі.

а) producers: сервіси, що відправляють дані до Kafka, використовуючи Sarama, можуть визначати куди і як часто дані будуть відправлятися. Це включає здатність визначати топіки для кожного виду подій. У лістингу 3.1 наведено приклад створення продюсера за допомогою бібліотеки Sarama;

б) consumers: сервіси, що отримують дані з Kafka, мають можливість підписатися на специфічні топіки. Завдяки Sarama, консюмери можуть ефективно обробляти повідомлення, що надходять, і забезпечувати асинхронну обробку даних. У лістингу 3.2 наведено приклад створення консьюмера за допомогою бібліотеки Sarama;

в) partitions та topics: партиції в Kafka дозволяють паралелізувати обробку та збільшити пропускну спроможність, адже кожен консюмер може читати з своєї партиції. Sarama дозволяє детально управляти цим процесом, забезпечуючи ефективну масштабованість і відмовостійкість.

Лістинг 3.1 Реалізація Kafka Producer за допомогою Golang

```
func NewAsyncProducer(cfg *config.Config, logger *slog.Logger)
(*AsyncProducer, error) {
    kafkaCfg := sarama.NewConfig()
    kafkaCfg.Producer.RequiredAcks = sarama.WaitForAll
```

```
    kafkaCfg.Producer.Retry.Max = 5
    kafkaCfg.Producer.Return.Successes = true
    sarama.Logger = log.New(log.Writer(), "sarama-producer: ",
log.LstdFlags)
    producer, err :=
sarama.NewAsyncProducer([]string{cfg.KafkaCfg.Broker}, kafkaCfg)
    if err != nil {
        return nil, fmt.Errorf("failed to create Sarama producer: %w", err)
    }
    return &AsyncProducer{producer: producer, logger: logger}, nil
}
```

Лістинг 3.2 Реалізація Kafka Consumer за допомогою Golang

```
func NewConsumerGroup(groupId, broker string) (*ConsumerGroup, error)
{
    cfg := sarama.NewConfig()
    cfg.Consumer.Offsets.Initial = sarama.OffsetNewest
    cfg.Consumer.Return.Errors = true
    sarama.Logger = log.New(log.Writer(), "sarama-consumer: ",
log.LstdFlags)
    brokers := []string{broker}
    consumerGroup, err := sarama.NewConsumerGroup(brokers, groupId,
cfg)
    if err != nil {
        log.Fatalf("Error creating consumer group: %v", err)
        return nil, errors.New("error creating consumer group")
    }
    return &ConsumerGroup{consumerGroup}, nil
}
```

У таблиці 3.3 можна побачити взаємодію мікросервісів з чергою Kafka за допомогою топіків.

Таблиця 3.3 – Взаємодія мікросервісів через Kafka

Мікросервіс	**Роль**	**Топіки, на які підписується**	**Топіки, в які публікує**
File Service	Consumer	DataAnalyzedEvent, FileApprovedEvent, ErrorEvent	
	Producer		FileUploadedEvent
OCR Service	Consumer	FileUploadedEvent	
	Producer		TextExtractedEvent, ErrorEvent
LLM Service	Consumer	TextExtractedEvent	
	Producer		DataAnalyzedEvent, ErrorEvent
Transaction Service	Consumer	ApproveFileEvent	
	Producer		FileApprovedEvent, ErrorEvent

3.2.4 Створення Docker-контейнерів

У сучасній розробці програмного забезпечення, особливо в контексті мікросервісної архітектури, важливо забезпечити швидке та ефективне розгортання сервісів. Docker та docker-compose стали потужними інструментами для вирішення цього завдання. Вони дозволяють розробникам створювати, тестувати та розгортати програми у контейнерах, забезпечуючи при цьому ізоляцію сервісів, узгодженість середовищ та

легкість у масштабуванні. У таблиці 3.4 представлено опис майбутнього контейнера та його основні функції.

Таблиця 3.4 – Опис майбутніх Docker-контейнерів

Сервіс	**Ім'я контейнера**	**Опис та функції**
1	**2**	**3**
Zookeeper	kafka-like-zookeeper	Zookeeper є координатором для розподілених систем, керує конфігурацією Kafka брокерів, забезпечує високу доступність.
Kafka	kafka-like	Kafka брокер, відповідає за обробку потоків даних, зберігання та передачу повідомлень між різними частинами системи.
Receipt Database	receipt-database	PostgreSQL база даних для мікросервісу обробки квитанцій, зберігає дані про квитанції, користувачів та транзакції.
Vision Database	vision-database	PostgreSQL база даних для мікросервісу комп'ютерного зору, зберігає дані, пов'язані з аналізом зображень та відео.
Transaction Database	transaction-database	PostgreSQL база даних для мікросервісу обробки транзакцій, зберігає фінансові дані та історію транзакцій.

Продовження таблиці 3.4

1	2	3
LLM Database	llm-database	PostgreSQL база даних для мікросервісу обробки природної мови, зберігає дані, необхідні для роботи мовної моделі.
File Service	file-microservice	Мікросервіс обробки квитанцій, відповідає за аналіз, зберігання та управління даними про квитанції.
Vision Service	vision-microservice	Мікросервіс комп'ютерного зору, відповідає за обробку зображень та відео, використовує машинне навчання для аналізу.
Transaction Service	transaction-microservice	Мікросервіс обробки транзакцій, відповідає за управління фінансовими операціями та їх безпеку.
LLM Service	llm-microservice	Мікросервіс обробки природної мови, використовує мовну модель GPT для генерації та аналізу тексту.

Docker Compose дозволяє вам визначити декілька сервісів (контейнерів) у файлі docker-compose.yml. Рисунок 3.10 вказує, як саме ці сервіси будуть взаємодіяти між собою, якими портами користуватись, які томи використовувати тощо.

```yaml
llm-service:
  container_name: llm-microservice
  build:
    context: .
    dockerfile: llm-service.dockerfile
  ports:
    - "50054:50054"
  env_file: .env
  depends_on:
    - llm-database
  tty: true
```

```yaml
llm-database:
  image: postgres:latest
  container_name: gpt-database
  environment:
    POSTGRES_USER: ${LLM_USER_DB}
    POSTGRES_PASSWORD: ${LLM_PASSWORD_DB}
    POSTGRES_DB: ${LLM_NAME_DB}
  ports:
    - "5553:5432"
  volumes:
    - ./llm-service/gpt-database:/var/lib/postgresql/data
  tty: true
```

а) б)

Рисунок 3.10 – Створення Docker-контейнера за допомогою docker-compose
а) контейнер мікросервісу; б) контейнер бази даних

Dockerfile визначає, як створити Docker образ для мікросервісу. На рисунку 3.11 приклад Dockerfile, де вказується базовий образ, копіюються необхідні файли, встановлюються залежності і визначається команда для запуску застосунку.

```dockerfile
# Use the official Golang image as the base image
FROM golang:alpine

# Set the working directory inside the container
WORKDIR /app

# Copy the local package files to the container's workspace
COPY ./receipt-service .

# Build the Go application
RUN go build -o main ./cmd/app

# Command to run the executable
CMD ["./main"]
```

Рисунок 3.11 – Створення Dockerfile

3.2.5 Створення користувацького інтерфейсу

Розробка користувацького інтерфейсу та функціональності вебплатформи є одним з ключових етапів проєктування та реалізації

системи. Користувацький інтерфейс повинен забезпечувати зручність та ефективність використання системи, а також надавати всі необхідні функціональні можливості [27].

Одним з основних завдань користувацького інтерфейсу є забезпечення можливості завантаження та перегляду інформації з чеків. Для цього необхідно розробити зручний механізм завантаження чеків у систему. Користувач повинен мати можливість вибрати файл з зображенням чека та завантажити його на сервер. При цьому важливо передбачити можливість завантаження декількох чеків одночасно, щоб спростити процес для користувачів, які мають велику кількість чеків. На рисунку 3.12 представлено реалізацію інтерфейсу користувача для завантаження файлів.

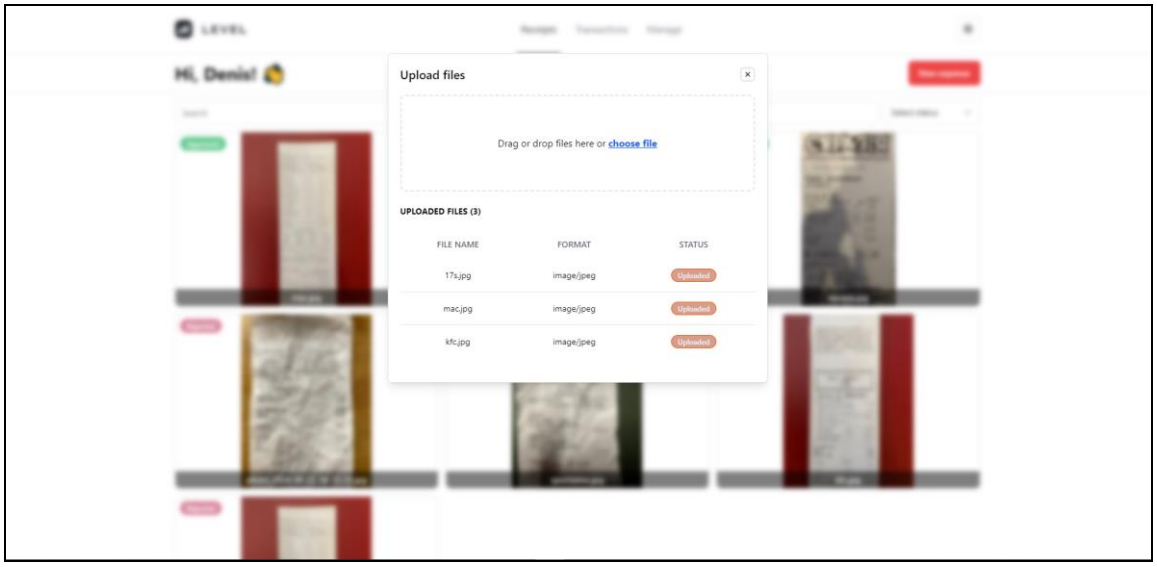


Рисунок 3.12 – Інтерфейс користувача для завантаження файлів

Після того, як користувач завантажує чеки, він має можливість ознайомитись з інформацією, яка була автоматично витягнута та категоризована системою. Для цього створено інтерфейс, який уможливлює перегляд списку всіх завантажених чеків. Користувач може вибрати будь-який чек, щоб детально переглянути відповідну інформацію, таку як сума покупки, дата та місце покупки. Важливою особливістю є також можливість перегляду категорії кожного чеку, і в разі помилок у автоматичній

категоризації, користувач має можливість внести необхідні корективи. Також інтерфейс підтримує фільтрацію чеків за статусом обробки та пошук конкретного чеку за назвою або іншими критеріями. На рисунку 3.13, 3.14, 3.15 представлено, як виглядає цей інтерфейс з переглядом усіх доступних документів.

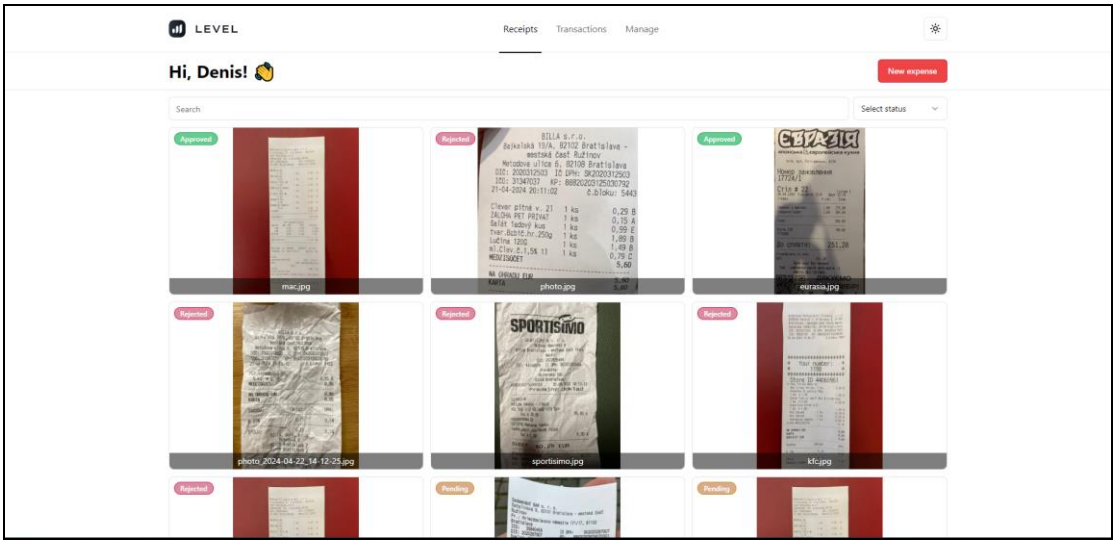


Рисунок 3.13 – Інтерфейс користувача для перегляду файлів

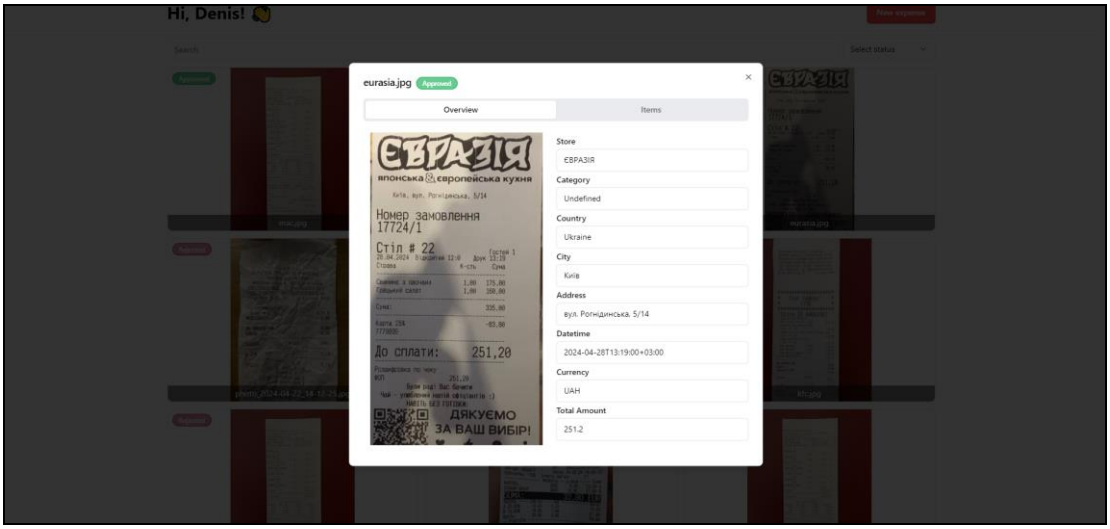


Рисунок 3.14 – Інтерфейс користувача для перегляду даних

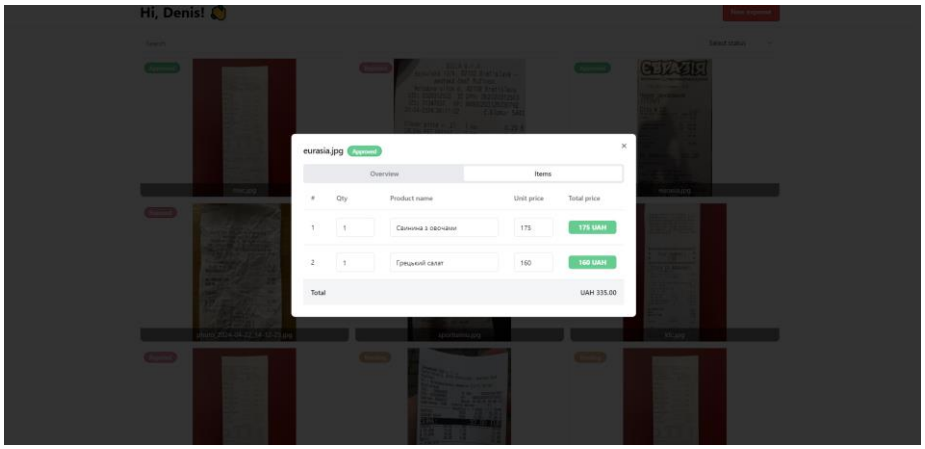


Рисунок 3.15 – Інтерфейс користувача для перегляду товарів

Далі розглядається реалізація інтерфейсу користувача, який може відображати та переглядати всі транзакції та категорії, відсортовані за валютою на основі сканованих фінансових документів. Для того, щоб користувачі могли легко керувати своїми фінансовими даними, цей елемент інтерфейсу є важливим [28].

Інтерфейс (рис. 3.16) спроєктований таким чином, щоб користувачі могли легко переключатися між різними сторінками та відстежувати свої витрати в різних валютах. Функціональність полягає в тому, щоб провести детальний огляд кожної транзакції з можливістю перегляду всіх пов’язаних витрат. Це допомагає користувачам більш точно аналізувати свої гроші, оскільки вони можуть бачити, на що саме витрачали гроші у певній валюті.

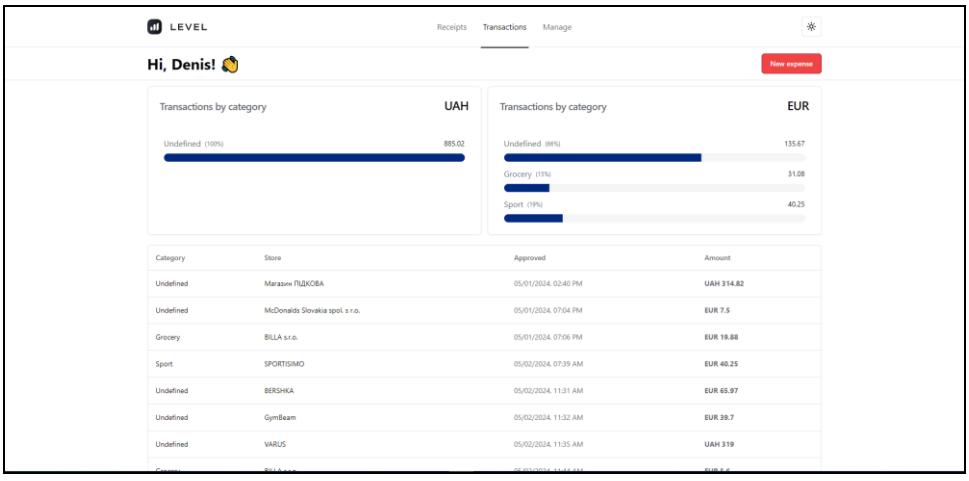
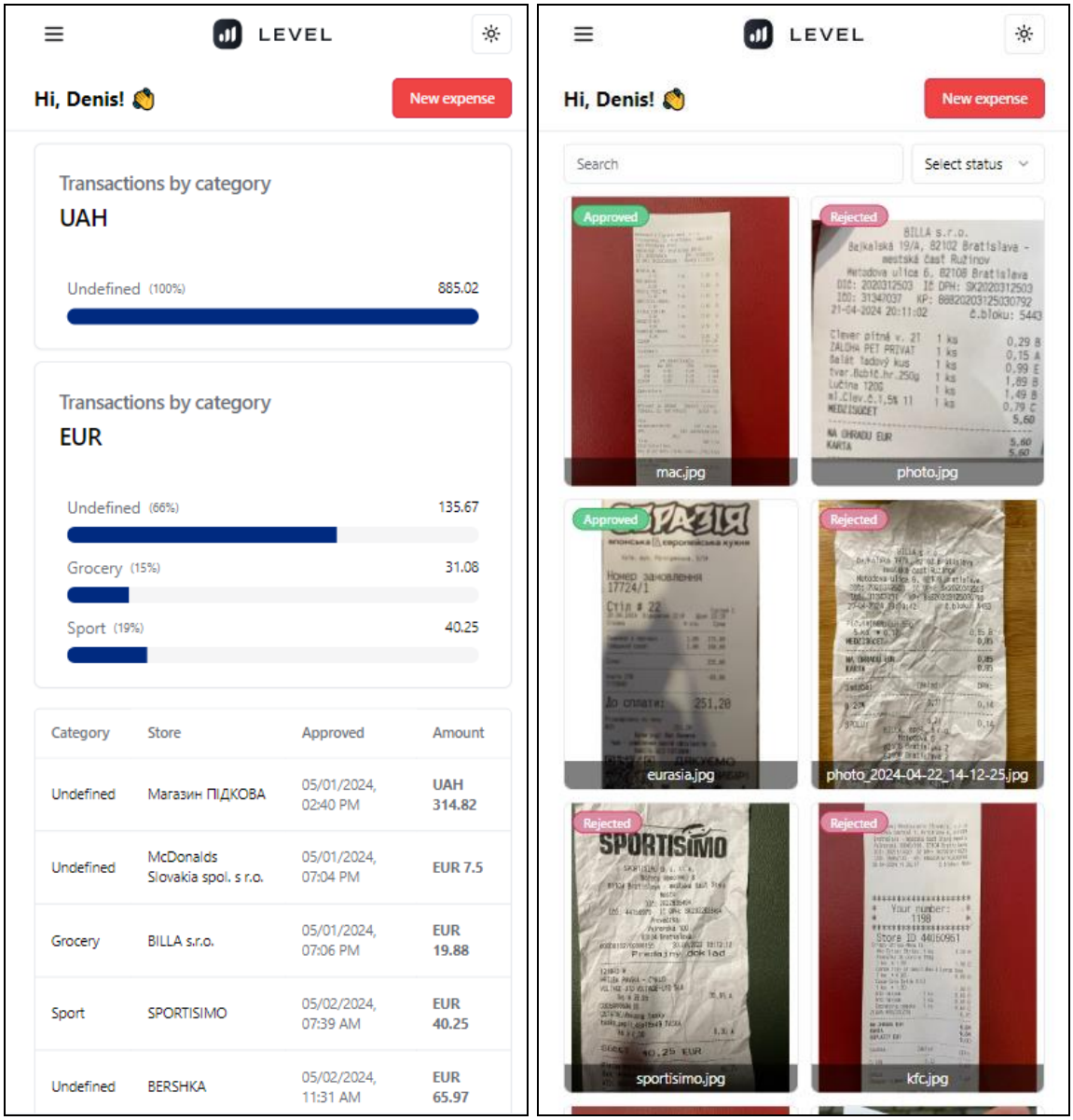


Рисунок 3.16 – Інтерфейс користувача для аналізу транзакцій за валютою

Важливим аспектом розробки користувацького інтерфейсу є його адаптивність. Платформа має підтримувати роботу на різних пристроях, включаючи комп’ютери, планшети та смартфони. Для цього необхідно розробити адаптивний дизайн, який буде коректно відображатися на різних екранах і забезпечувати зручність використання незалежно від розміру пристрою для інтерфейсу транзакцій, який показано на рисунку 3.17 а) та інтерфейсу завантажених файлів продемонстрований на рисунку 3.17 б).



а) б)

Рисунок 3.17 – Адаптивний інтерфейс користувача

а) інтерфейс транзакцій; б) інтерфейс завантажених файлів

Для розробки користувацького інтерфейсу (UI) вебплатформи буде використовуватися комбінація сучасних технологій та фреймворків, зокрема React, Next.js, TypeScript, та Tailwind CSS для стилізації.

Використання цих технологій та інструментів дозволить створити надійний, масштабований та легко підтримуваний користувацький інтерфейс для вебплатформи, орієнтований на забезпечення оптимального користувацького досвіду та ефективної взаємодії з системою.

ВИСНОВКИ

У рамках виконання даної кваліфікаційної роботи було розроблено і реалізовано мікросервісну вебплатформу для обробки фінансових документів з використанням технологій OCR та LLM. Реалізація включала в себе створення надійної архітектури, оптимізацію взаємодії мікросервісів через Kafka, інтеграцію з базами даних та розробку зручного користувацького інтерфейсу з використанням React, Next.js та TypeScript.

Протягом роботи було впроваджено передові підходи контейнеризації за допомогою Docker, що забезпечило легкість розгортання та масштабування системи. Також, було використано Golang як основну мову програмування, що дозволило ефективно вирішувати задачі асинхронної обробки даних завдяки вбудованим можливостям конкурентності.

Результати реалізованої системи демонструють значні переваги використання мікросервісної архітектури для комплексної обробки фінансових даних, забезпечуючи високу швидкість обробки, гнучкість налаштування сервісів та здатність до швидкого масштабування. Отримані дані та методики були представлені і високо оцінені на декількох наукових форумах та конференціях, що свідчить про актуальність та ефективність вибраних рішень.

Таким чином, виконана кваліфікаційна робота не тільки вирішує актуальну задачу обробки та категоризації фінансових документів у реальному часі, але й вносить значний вклад у розвиток технологій розробки програмного забезпечення, відкриваючи нові можливості для подальших досліджень та розвитку в даній області.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Ковтуненко, А. Р., Яковлева, О. В., Любченко, В. А., & Янголенко, О. В. (2020). Дослідження сумісного використання математичної морфології та згорткових нейронних мереж для вирішення задачі розпізнавання цінників.

2. Lyashenko, V., Babker, A., & Lyubchenko, V. (2017). Wavelet Analysis of Cytological Preparations Image in Different Color Systems.

3. Любченко, В. А., Яковлева, О. В., & Передрій, Є. О. (2008). Нормалізація перспективних перетворень проектно спотворених зображень. Вісник Національного технічного університету «ХПІ». Серія: Системний аналіз, управління та інформаційні технології, (26), 174–184.

4. Hezretov, M. (2021). Budget Tracker Highly Customizable Budgeting Mobile Application (Doctoral dissertation).

5. Kaye, J. J., McCuistion, M., Gulotta, R., & Shamma, D. A. (2014, April). Money talks: tracking personal finances. In Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (pp. 521–530).

6. Кобилін, О.А., & Творошенко, І.С. (2021). Методи цифрової обробки зображень: навч. посібник. *Харків: ХНУРЕ*.

7. Яковлева, О. В., & Кускова, І. В. (2006). Дослідження результатів сегментації зображень методом матриць збігів.

8. Daradkeh, Y.I., Tvoroshenko, I., Gorokhovatskyi, V., Latiff, L.A., and Ahmad, N. (2021) Development of Effective Methods for Structural Image Recognition Using the Principles of Data Granulation and Apparatus of Fuzzy Logic, *IEEE Access*, 9, pp. 13417–13428.

9. Яковлева, О. В., & Ковтуненко, О. Р. (2019). Пошук та розпізнавання цінників товарів на зображеннях.

10. Daradkeh, Y.I., Gorokhovatskyi, V., Tvoroshenko, I., Gadetska, S., and Al–Dhaifallah, M. (2021) Methods of Classification of Images on the Basis of the

Values of Statistical Distributions for the Composition of Structural Description Components, *IEEE Access*, 9, pp. 92964–92973.

11. Gorokhovatskyi, V.O., Tvoroshenko, I.S., and Peredrii O.O. (2020) Image classification method modification based on model of logic processing of bit description weights vector, *Telecommunications and Radio Engineering*, 79(1), pp. 59–69.

12. Daradkeh Y.I., Gorokhovatskyi V., Tvoroshenko I., and Zeghid M. (2022) Cluster representation of the structural description of images for effective classification, Computers, Materials & Continua, 73(3), pp. 6069–6084.

13. Tvoroshenko I., and Gorokhovatskyi V. (2022) The Application of Hybrid Intelligence Systems for Dynamic Data Analysis, *International Journal of Engineering and Information Systems*, 6(2), pp. 40–48.

14. White, J., Fu, Q., Hays, S., Sandborn, M., Olea, C., Gilbert, H., ... & Schmidt, D. C. (2023). A prompt pattern catalog to enhance prompt engineering with chatgpt.

15. Richardson, C. (2018). *Microservices patterns: with examples in Java*. Simon and Schuster.

16. Taibi, D., Lenarduzzi, V., & Pahl, C. (2020). Microservices anti–patterns: A taxonomy. Microservices: Science and Engineering, 111–128.

17. Speth, S., Stieß, S., & Becker, S. (2022, March). A saga pattern microservice reference architecture for an elastic SLO violation analysis. In 2022 IEEE 19th International Conference on Software Architecture Companion (ICSA–C) (pp. 116–119). IEEE.

18. Mori, S., Suen, C. Y., & Yamamoto, K. (1992). Historical review of OCR research and development. Proceedings of the IEEE, 80(7), 1029–1058.

19. Bosch, O. J., Revilla, M., & Paura, E. (2019). Answering mobile surveys with images: an exploration using a computer vision API. Social Science Computer Review, 37(5), 669–683.

20. Brynjolfsson, E., Li, D., & Raymond, L. R. (2023). Generative AI at work (No. w31161). National Bureau of Economic Research.

21. Feuerriegel, S., Hartmann, J., Janiesch, C., & Zschech, P. (2024). Generative ai. Business & Information Systems Engineering, 66(1), 111–126.

22. Carlà, M. M., Gambini, G., Baldascino, A., Giannuzzi, F., Boselli, F., Crincoli, E., ... & Rizzo, S. (2024). Exploring AI–chatbots' capability to suggest surgical planning in ophthalmology: ChatGPT versus Google Gemini analysis of retinal detachment cases. British Journal of Ophthalmology.

23. Floridi, L., & Chiriatti, M. (2020). GPT–3: Its nature, scope, limits, and consequences. Minds and Machines, 30, 681–694.

24. Liu, X., Zheng, Y., Du, Z., Ding, M., Qian, Y., Yang, Z., & Tang, J. (2023). GPT understands, too.

25. Chinosi, M., & Trombetta, A. (2012). BPMN: An introduction to the standard. Computer Standards & Interfaces, 34(1), 124–134.

26. White, S. A., & Miers, D. (2008). BPMN modeling and reference guide: understanding and using BPMN. Future Strategies Inc..

27. Nasution, W. S. L., & Nusa, P. (2021). UI/UX design web–based learning application using design thinking method. ARRUS Journal of Engineering and Technology, 1(1), 18–27.

28. Indriana, M., & Adzani, M. L. (2017, November). UI/UX analysis & design for mobile e–commerce application prototype on Gramedia. com. In 2017 4th International Conference on New Media Studies (CONMEDIA) (pp. 170–173). IEEE.

29. Kreps, J., Narkhede, N., & Rao, J. (2011, June). Kafka: A distributed messaging system for log processing. In Proceedings of the NetDB (Vol. 11, No. 2011, pp. 1–7).

30. Miell, I., & Sayers, A. (2019). Docker in practice. Simon and Schuster.