НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
“КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО”

Факультет інформатики та обчислювальної техніки

Кафедра обчислювальної техніки

До захисту допущено:
В. о. зав. кафедри
_____ Артем ВОЛОКИТА
(підпис)
“__” _____2026 р.

Дипломний проєкт

на здобуття ступеня бакалавра
за освітньо-професійною програмою “Інженерія програмного забезпечення комп’ютерних систем”
спеціальності 121 “Інженерія програмного забезпечення”

на тему: Система інтелектуальної обробки та категоризації банківських виписок

Виконав : студент __4__ курсу, групи ІМ-21
(шифр групи)

_____Ковальов Євгеній Русланович_____
(прізвище, ім’я, по батькові) (підпис)

Керівник _____проф., д.т.н., проф. Стіренко С. Г._____
(посада, науковий ступінь, вчене звання, прізвище та ініціали) (підпис)

Консультант (нормоконтроль) ст. викл., д.ф. комп’ютерної інженерії,_____
Міщенко Л. Д._____
(назва розділу) (посада, вчене звання, науковий ступінь, прізвище та ініціали) (підпис)

Рецензент ___доц., к.т.н., доцент кафедри АУТС Павло Катін_____
(посада, науковий ступінь, вчене звання, прізвище та ініціали) (підпис)

Засвідчую, що у цьому дипломному проєкті немає запозичень з праць інших авторів без відповідних посилань.
Студент _____
(підпис)

Київ – 2026 р.

НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
“КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО”

Факультет інформатики та обчислювальної техніки

Кафедра обчислювальної техніки

Рівень вищої освіти – перший (бакалавр)

Освітньо-професійна програма

“Інженерія програмного забезпечення комп’ютерних систем”

спеціальності 121 “Інженерія програмного забезпечення”

ЗАТВЕРДЖУЮ
В. о. зав. кафедри
_____ Артем ВОЛОКИТА
(підпис)
“__” _____2026 р.

ЗАВДАННЯ
на бакалаврський дипломний проєкт студента

Ковальова Євгенія Руслановича

1. Тема проєкту *Система інтелектуальної обробки та категоризації банківських виписок*
керівник проєкту Стіренко Сергій Григорович, проф., д.т.н., проф.
(прізвище, ім’я, по батькові, науковий ступінь, вчене звання)
затверджені наказом по університету від 3 червня 2026 року №2055-с

2. Термін здачі студентом закінченого проєкту *9 червня 2026 р.*

3. Вихідні дані до проєкту *технічна документація, теоретичні дані.*

4. Зміст розрахунково-пояснювальної записки (перелік питань, які розробляються): Огляд систем інтелектуальної обробки та категоризації банківських виписок. Аналіз алгоритмів і технологій для розробки системи. Деталі розробки системи. Тестування та аналіз результатів роботи системи.

5. Перелік графічного матеріалу (з точним позначенням обов’язкових креслень): Структурна схема системи. Схема бази даних. Алгоритм дій програмного забезпечення.

6. Консультанта проєкту, з вказівкою розділів проєкту, які до них вносяться

Розділ	Консультант	Підпис, дата	
		Завдання видав	Завдання прийняв
Нормоконтроль	Міщенко Л. Д.		

7. Дата видачі завдання «1» вересня 2025 р.

Календарний план

№ п/п	Найменування етапів дипломного проєкту	Терміни виконання етапів проєкту	Примітки
1.	*Затвердження теми проєкту*	*01.12.2025-12.12.2025*	
2.	*Вивчення та аналіз завдання*	*12.12.2025-13.03.2026*	
3.	*Розробка архітектури та загальної структури системи*	*13.03.2026-03.04.2026*	
4.	*Розробка структур окремих підсистем*	*03.04.2026-17.04.2026*	
5.	*Програмна реалізація системи*	*17.04.2026-15.05.2026*	
6.	*Оформлення пояснювальної записки*	*15.05.2026-29.05.2026*	
7.	*Захист програмного продукту*	*29.05.2026*	
8.	*Передзахист*	*9.06.2026*	
9.	*Захист*	*16.06.2026*	

Студент-дипломник _____ Євгеній КОВАЛЬОВ
(підпис)

Керівник проєкту _____ Сергій СТІРЕНКО
(підпис)

АНОТАЦІЯ

У дипломному проєкті розглянуто проблему ручної обробки та категоризації банківських виписок у малому та середньому бізнесі. Проаналізовано сучасні підходи до автоматизації цього процесу, зокрема правила на основі ключових слів і методи промпт-інжинірингу для великих мовних моделей. На основі порівняльного аналізу трьох стратегій обрано оптимальну конфігурацію для застосунку.

Розроблено вебзастосунок для бухгалтерів, що дозволяє завантажувати PDF-файли банківських виписок, автоматично витягувати транзакції через нативний документний API Claude та категоризувати їх у двоетапному конвеєрі: спочатку за налаштовуваними правилами на основі ключових слів, потім - засобами штучного інтелекту. Реалізовано можливість перегляду та ручного редагування результатів, а також експорту у форматах CSV та Excel.

Ключові слова: банківська виписка, автоматична категоризація, великі мовні моделі, Claude API, промпт-інжиніринг, Next.js, автоматизація бухгалтерського обліку.

ANNOTATION

This project addresses the problem of manual processing and categorization of bank statements in small and medium-sized businesses. Modern approaches to automating this process were analyzed, including keyword-based rules and prompt engineering methods for large language models. Based on a comparative analysis of three strategies, an optimal configuration for the application was selected.

A web application for accountants was developed, enabling users to upload PDF bank statements, automatically extract transactions via the Claude native document API, and categorize them in a two-stage pipeline: first by configurable keyword rules, then by AI. The application supports manual review and editing of results, and export in CSV and Excel formats.

Key words: bank statement, automated categorization, large language models, Claude API, prompt engineering, Next.js, bookkeeping automation.

справки	Формат	*Значення*	*Найменування*	Кіл. листів	№ екземпля	*Додаток*
			Документація загальна			
			Знову розроблена			
	А4	***ІАЛЦ.467200.002 ТЗ***	Система інтелектуальної	4		
			обробки та категоризації			
			банківських виписок			
			Технічне завдання			
	А4	***ІАЛЦ.467200.003 ПЗ***	Система інтелектуальної	65		
			обробки та категоризації			
			банківських виписок			
			Пояснювальна записка			
	А4	***ІАЛЦ.467200.004 Д1***	Система інтелектуальної	1		
			обробки та категоризації			
			банківських виписок			
			Структурна схема			
			системи			
	А4	***ІАЛЦ.467200.005 Д2***	Система інтелектуальної	1		
			обробки та категоризації			
			банківських виписок			
			Схема бази даних			
	А4	***ІАЛЦ.467200.006 Д3***	Система інтелектуальної	1		
			обробки та категоризації			
			банківських виписок			
			Алгоритм дій програмного			
			забезпечення			
	А4	***ІАЛЦ.467200.007 Д4***	Система інтелектуальної	12		
			обробки та категоризації			
			банківських виписок			
			Текст програмного коду			

					ІАЛЦ.467200.001 ОА
Зм	*Лист*	*№ докум.*	*Підп*	*Дата*	
Розроб		Ковальов Є.Р.			*Система інтелектуальної обробки та категоризації банківських виписок* Опис альбому
Перев		Стіренко С.Г.			

Літ.	Аркуш	Аркушів
	1	1

КПІ ім. Ігоря Сікорського, ФІОТ, ІМ-21

ТЕХНІЧНЕ ЗАВДАННЯ
ДО ДИПЛОМНОГО ПРОЄКТУ

на тему: «*Система інтелектуальної обробки та категоризації банківських виписок*»

Київ – 2026

ЗМІСТ

1 НАЙМЕНУВАННЯ ТА ОБЛАСТЬ ЗАСТОСУВАННЯ

Дане технічне завдання поширюється на розробку системи інтелектуальної обробки та категоризації банківських виписок, а також на подальшу підтримку та вдосконалення розробленої системи.

Областю застосування цієї системи є автоматизація бухгалтерського обліку на малих та середніх підприємствах, у бухгалтерських фірмах та фінансових відділах організацій, де виникає потреба у регулярній обробці великих обсягів банківських транзакцій та їх класифікації за статтями витрат і доходів.

2 ПІДСТАВИ ДЛЯ РОЗРОБКИ

Підставою для розробки даної системи є завдання для виконання роботи кваліфікаційно-освітнього рівня «бакалавр інженерії програмного забезпечення», який був затверджений факультетом “Інформатики та обчислювальної техніки” кафедрою обчислювальної техніки Національного технічного Університету України «Київський Політехнічний інститут ім. Ігоря Сікорського».

3 МЕТА ТА ПРИЗНАЧЕННЯ РОЗРОБКИ

Метою та призначенням даного дипломного проєкту є розробка вебзастосунку для автоматизованої обробки та категоризації банківських виписок із застосуванням великих мовних моделей, що дозволить бухгалтерам значно скоротити час на ручне введення та класифікацію транзакцій.

4 ДЖЕРЕЛА РОЗРОБКИ

Джерелами розробки даного дипломного проєкту є офіційна документація Anthropic Claude API, Next.js та Supabase, публікації та науково-

технічні статті з тематики автоматизації бухгалтерського обліку та промпт-інжинірингу, а також відкриті публікації в мережі Інтернет.

5 ТЕХНІЧНІ ВИМОГИ

5.1 Вимоги до розробленого продукту

Розроблена система має виконувати такі вимоги:

- Простий та інтуїтивно зрозумілий інтерфейс, орієнтований на бухгалтерів без технічного досвіду.
- Надати можливість завантажувати PDF-файли банківських виписок.
- Автоматично витягувати список транзакцій із завантаженого PDF.
- Автоматично категоризувати транзакції.
- Надати можливість переглядати, фільтрувати та вручну редагувати результати категоризації.
- Надати можливість налаштовувати правила категоризації на основі ключових слів.
- Надати можливість експортувати результати у форматах CSV та Excel.
- Забезпечити автентифікацію користувачів та захист даних.
- Відображати детальний журнал обробки з інформацією про кожен етап конвеєру.

5.2 Вимоги до програмного забезпечення

- Операційна система Windows, macOS або Linux.
- Сучасний веббраузер Google Chrome, Mozilla Firefox або Microsoft Edge актуальної версії.
- Стабільне підключення до мережі Інтернет.

5.3 Вимоги до апаратної частини

- Процесор: не нижче Intel® Core™ i7-1365U 13-го покоління.
- Обсяг постійної пам’яті (ROM): не менше 64 ГБ.
- Обсяг оперативної пам’яті (RAM): не менше 8 ГБ.

6 ЕТАПИ РОЗРОБКИ

Назва етапів виконання	Термін виконання
Затвердження теми роботи	01.12.2025-12.12.2025
Вивчення та аналіз завдання	12.12.2025-13.03.2026
Розробка архітектури та загальної структури системи	13.03.2026-03.04.2026
Розробка структур окремих частин системи	03.04.2026-17.04.2026
Програмна реалізація системи	17.04.2026-15.05.2026
Виправлення помилок	15.05.2026-29.05.2026
Оформлення пояснювальної записки	29.05.2026

ПОЯСНЮВАЛЬНА ЗАПИСКА
ДО ДИПЛОМНОГО ПРОЄКТУ

на тему: «*Система інтелектуальної обробки та категоризації банківських виписок*»

Київ – 2026

ЗМІСТ

ПЕРЕЛІК СКОРОЧЕНЬ

МСБ	Малий та середній бізнес
ШІ	Штучний інтелект
API	Application Programming Interface (Інтерфейс програмування застосунків)
CoT	Chain-of-Thought (Ланцюжок міркувань)
JWT	JSON Web Token (Вебтокен JSON)
LLM	Large Language Model (Велика мовна модель)
OCR	Optical Character Recognition (Оптичне розпізнавання символів)
RLS	Row Level Security (Безпека на рівні рядків)
SaaS	Software as a Service (Програмне забезпечення як послуга)
SSE	Server-Sent Events (Події, надіслані сервером)
SVM	Support Vector Machine (Метод опорних векторів)
TF-IDF	Term Frequency-Inverse Document Frequency (Частота терміна - обернена частота документа)

ВСТУП

У сучасних умовах ведення малого та середнього бізнесу бухгалтерський облік залишається одним із найбільш рутинних і ресурсомістких аспектів операційної діяльності. Зокрема, регулярна обробка банківських виписок - ідентифікація та класифікація транзакцій за статтями витрат і доходів - досі здебільшого виконується вручну. Це призводить до значних витрат часу фахівців, підвищеного ризику помилок і сповільнення фінансової звітності.

Актуальною проблемою в галузі автоматизації бухгалтерського обліку є обмеженість традиційних підходів до категоризації транзакцій. Правила на основі ключових слів, які є найпоширенішим рішенням, ефективні лише для однозначних та регулярних платежів, проте потребують ручного налаштування, не справляються з нестандартними описами і не здатні розв'язувати неоднозначні випадки, де один опис може відповідати різним категоріям залежно від контексту. Таким чином, значна частина транзакцій усе одно потребує ручного втручання.

У рамках дипломного проєкту пропонується розроблення вебзастосунку для бухгалтерів, що автоматизує завантаження PDF (Portable Document Format) файлів банківських виписок і категоризацію транзакцій за допомогою великих мовних моделей. Для цього реалізовано двоетапний конвеєр обробки: спочатку транзакції перевіряються за налаштовуваними правилами на основі ключових слів, після чого решта передається до штучного інтелекту. Як компонент ШІ (штучного інтелекту) використовується Claude API з нативною підтримкою PDF-документів, що дозволяє витягувати транзакції безпосередньо з файлу без проміжного OCR (Optical Character Recognition, оптичне розпізнавання символів) перетворення. Для вибору оптимальної конфігурації порівнюються три стратегії промпт-інжинірингу - базова інструкція, few-shot навчання та ланцюжок міркувань - за критеріями точності, вартості та часу відповіді.

Таким чином, робота поєднує задачу практичної автоматизації рутинних бухгалтерських процесів із дослідженням сучасних методів промпт-інжинірингу для великих мовних моделей, що визначає її актуальність та практичну цінність.

РОЗДІЛ 1 ОГЛЯД СИСТЕМ ІНТЕЛЕКТУАЛЬНОЇ ОБРОБКИ ТА КАТЕГОРИЗАЦІЇ БАНКІВСЬКИХ ВИПИСОК

1.1 Предметна область та актуальність задачі

Банківська виписка - це офіційний документ, що надається фінансовою установою і містить перелік усіх транзакцій за певний період: надходжень, видатків, переказів, банківських комісій тощо. Для підприємств малого та середнього бізнесу (МСБ) регулярна обробка банківських виписок є обов'язковою частиною бухгалтерського обліку: кожна транзакція має бути ідентифікована та віднесена до відповідної статті витрат або доходів.

На практиці підприємства МСБ щомісяця отримують виписки з одного або декількох банківських рахунків, кожна з яких може містити від кількох десятків до кількох сотень транзакцій. Традиційний процес обробки передбачає завантаження PDF-файлу виписки, ручне або напівавтоматичне перенесення даних у бухгалтерську систему, а потім - класифікацію кожної транзакції за відповідною категорією [1, 2]. Цей процес є монотонним, трудомістким і схильним до людських помилок.

За оцінками дослідників у сфері автоматизації бухгалтерського обліку, фахівці витрачають значну частину робочого часу на ручне введення та класифікацію транзакцій. У масштабах навіть невеликого підприємства це становить кілька годин на місяць - час, який міг би бути спрямований на аналіз фінансових даних, планування та прийняття управлінських рішень. Крім того, ручна обробка призводить до неузгодженості у категоризації: один і той самий вид витрат може бути класифікований по-різному різними фахівцями або навіть одним фахівцем у різний час.

Таким чином, автоматизація обробки та категоризації банківських виписок є актуальною задачею в галузі інженерії програмного забезпечення, що має безпосередній практичний вплив на ефективність бухгалтерської діяльності в підприємствах МСБ.

1.2 Методи категоризації банківських транзакцій

Підходи до автоматизації категоризації банківських транзакцій еволюціонували разом із розвитком технологій - від простих детерміністичних правил до моделей глибокого навчання та великих мовних моделей. Нижче розглянуто основні методи в порядку зростання складності.

1.2.1 Ручна обробка як базовий підхід

Ручна обробка є відправною точкою і досі залишається найпоширенішою практикою серед малих підприємств. Бухгалтер вручну переглядає кожну транзакцію у банківській виписці та призначає їй відповідну категорію в обліковій системі. У простих випадках цей підхід є цілком прийнятним - він не потребує жодних технічних засобів і дає бухгалтеру повний контроль над результатом.

Проте ручна обробка має суттєві обмеження в контексті масштабу та повторюваності. Зі зростанням обсягів транзакцій і кількості банківських рахунків час, витрачений на категоризацію, збільшується лінійно. Крім того, ручний процес не гарантує узгодженості: категоризація одних і тих самих типів транзакцій може відрізнятися залежно від фахівця або від часу. Це ускладнює формування достовірної фінансової звітності та порівняння даних між звітними періодами [1].

1.2.2 Правила на основі ключових слів

Перший рівень автоматизації - це системи правил, що зіставляють опис транзакції з наборами ключових слів і автоматично призначають категорію при збігу. Наприклад, правило може визначати: якщо опис транзакції містить рядок "GOOGLE WORKSPACE", категорія - "Програмне забезпечення та підписки".

Правилові системи мають очевидні переваги: вони прозорі, детерміністичні, не потребують навчання та є безкоштовними з точки зору обчислювальних ресурсів. Для регулярних, однозначних транзакцій - оренда офісу, відомі постачальники, банківські комісії - вони забезпечують стовідсоткову точність після первинного налаштування.

Водночас правилові підходи мають фундаментальне обмеження: вони не здатні узагальнювати. Правило для відомого постачальника не допоможе при першій появі нового SaaS-інструменту або нового контрагента. Описи транзакцій часто скорочені, містять коди або назви платіжних процесорів замість назви реального постачальника. Крім того, підтримка бази правил потребує постійних зусиль: зі зміною постачальників і появою нових категорій витрат правила стають застарілими [2].

1.2.3 Класичне машинне навчання

Для подолання обмежень правилових систем дослідники запропонували використовувати методи класичного машинного навчання - зокрема, алгоритми класифікації тексту. Опис транзакції розглядається як текстовий документ, а категорія - як клас. Серед найпоширеніших підходів: Наївний Байєс, метод опорних векторів (SVM), випадковий ліс та градієнтний бустинг. Вхідні ознаки формуються на основі TF-IDF-векторизації або n-грамних представлень тексту опису транзакції.

Перевага ML-підходів полягає в здатності до узагальнення: модель, навчена на прикладах, може правильно класифікувати нові, раніше не бачені описи, якщо вони семантично подібні до навчальних. При достатньому обсязі навчальних даних такі моделі демонструють суттєво вищу точність порівняно з правиловими системами на різноманітних та зашумлених описах транзакцій [3].

Однак класичне ML має власні обмеження. По-перше, воно потребує значного обсягу розмічених навчальних даних, які у більшості малих підприємств відсутні або неякісні. По-друге, при зміні системи категорій або появі нових класів модель потребує перенавчання. По-третє, класичні ML-моделі погано справляються з контекстуальною неоднозначністю: транзакція "UBER" може бути як витратою на транспорт, так і оплатою їжі через Uber Eats - і класифікатор без контекстуальних ознак не може розрізнити ці випадки.

1.2.4 Великі мовні моделі

Поява великих мовних моделей (LLM - Large Language Models) ознаменувала якісний стрибок у задачах розуміння природної мови, до яких належить і категоризація транзакцій. Архітектурною основою сучасних LLM є механізм самоуваги, запропонований у роботі Vaswani та ін. [4]

LLM попередньо навчені на масштабних корпусах тексту і містять широкі знання про компанії, галузі, типи витрат і фінансові операції. Це дозволяє їм категоризувати транзакції без будь-якого додаткового навчання - у режимі zero-shot або few-shot.

На відміну від класичних ML-моделей, LLM здатні:

- розуміти скорочені або закодовані назви постачальників через семантичний аналіз;
- враховувати знак суми транзакції (позитивна - надходження, негативна - витрата) при виборі категорії;

- застосовувати покрокове логічне міркування для неоднозначних випадків;
- адаптуватися до нових категорій без перенавчання - шляхом зміни інструкції у промпті.

Ключовим питанням при застосуванні LLM є вибір стратегії промпт-інжинірингу - способу формулювання запиту до моделі. Різні стратегії суттєво впливають на точність, вартість і час відповіді, що є предметом дослідження у даній роботі.

1.3 Мотивації для автоматизації

Автоматизація категоризації банківських транзакцій розглядається не лише як засіб зменшення рутинного навантаження, а й як стратегічний інструмент підвищення якості бухгалтерського обліку загалом. Основні мотивації можна згрупувати за трьома вимірами: операційна ефективність, якість даних та стратегічна цінність.

З точки зору операційної ефективності, автоматизація безпосередньо скорочує час на повторювані операції. Для підприємства з обсягом 200-500 транзакцій на місяць ручна категоризація займає кілька годин щомісяця. Автоматизована система здатна обробити ту саму кількість транзакцій за секунди, звільняючи фахівця від рутини та даючи змогу зосередитися на аналізі й прийнятті рішень. У бухгалтерських фірмах, що обслуговують десятки клієнтів, цей ефект множиться пропорційно кількості клієнтів [1, 2].

З точки зору якості даних, автоматизована система забезпечує узгодженість категоризації: одні й ті самі правила або модель застосовуються до всіх транзакцій незалежно від часу або виконавця. Це усуває варіативність, притаманну ручній обробці, і підвищує достовірність фінансової звітності.

Узгоджена категоризація є особливо важливою при порівнянні фінансових показників між звітними періодами або при підготовці до аудиту.

З точки зору стратегічної цінності, автоматизація відкриває можливість для більш глибокого аналізу фінансових даних. Коли категоризація виконується автоматично і надійно, бухгалтер отримує структуровані дані, придатні для аналізу витрат, виявлення аномалій та побудови фінансових прогнозів.

1.4 Штучний інтелект у бухгалтерській автоматизації

Застосування штучного інтелекту у фінансовому секторі охоплює широкий спектр задач: від виявлення шахрайства та оцінки кредитних ризиків до прогнозування грошових потоків і автоматизації бухгалтерського обліку. У контексті категоризації транзакцій ШІ-підходи набули особливого розвитку разом із появою загальнодоступних великих мовних моделей.

Ключовою перевагою LLM для задач класифікації фінансових операцій є їхня здатність до узагальнення без спеціалізованого навчання. Моделі, навчені на масштабних корпусах текстів, накопичили знання про широкий спектр компаній, галузей і типів витрат. Завдяки цьому вони здатні коректно інтерпретувати описи транзакцій - навіть скорочені або закодовані - без попереднього ознайомлення з конкретним підприємством або його обліковою системою [3].

Важливим напрямом у застосуванні LLM є промпт-інжиніринг - практика формулювання запитів до мовних моделей для отримання найбільш точних і корисних відповідей. Для задач класифікації виділяють три основні стратегії. Перша - базова інструкція (zero-shot) - передбачає лише перелік категорій і пряму вказівку класифікувати транзакцію. Цей підхід є найшвидшим і найдешевшим, але поступається точністю у неоднозначних випадках. Друга - few-shot навчання - включає до запиту кілька прикладів правильних

класифікацій, що підвищує узгодженість відповідей. Третя - ланцюжок міркувань (chain-of-thought) - інструктує модель міркувати покроково перед присвоєнням категорії, що підвищує точність для складних випадків ціною більших витрат токенів [5, 6, 7].

Окремо варто відзначити нативну підтримку документів у сучасних LLM. Деякі моделі - зокрема, Claude від Anthropic - приймають PDF-файли безпосередньо як вхідні дані, що дозволяє витягувати структуровані дані з банківських виписок без проміжного перетворення в текст. Це усуває залежність від сторонніх OCR-сервісів і зменшує кількість точок відмови в системі [8, 9].

1.5 Огляд існуючих систем

1.5.1 QuickBooks Online

QuickBooks Online - одна з найпоширеніших бухгалтерських SaaS-платформ у світі, розроблена компанією Intuit. У контексті обробки банківських виписок платформа пропонує імпорт транзакцій через пряме підключення до банківського рахунку або завантаження файлу у форматах CSV/OFX/QFX. Автоматична категоризація базується на алгоритмі машинного навчання, що аналізує історію транзакцій користувача і пропонує категорії на основі попередніх класифікацій аналогічних описів.

QuickBooks також підтримує визначені користувачем правила (bank rules), що дозволяють автоматично категоризувати транзакції за умовами збігу тексту, суми або банківського рахунку. Ці правила виконуються до машинного навчання і мають пріоритет над автоматичними пропозиціями.

Обмеженням платформи є відсутність нативної підтримки завантаження PDF-файлів банківських виписок для автоматичного витягування транзакцій -

імпорт з PDF потребує попереднього перетворення у структурований формат стороннім інструментом. Крім того, платформа не надає можливості налаштування алгоритму категоризації або вибору стратегії класифікації [1].

1.5.2 Xero

Xero - провідна бухгалтерська платформа, орієнтована переважно на малий і середній бізнес. Підхід Xero до категоризації транзакцій подібний до QuickBooks: платформа підтримує пряме підключення до банківських рахунків і застосовує автоматичні пропозиції категорій на основі навченої моделі, яка враховує поведінку конкретного користувача та агреговані дані від усіх користувачів платформи.

Ключовою особливістю Xero є функція "bank rules" з розвиненими умовами: правила можуть включати логічні оператори AND/OR, умови на суму, назву контрагента та тип транзакції. Платформа також підтримує автоматичне зіставлення банківських транзакцій із попередньо введеними рахунками-фактурами.

Аналогічно до QuickBooks, Xero не підтримує нативне витягування транзакцій з PDF-файлів. Платформа орієнтована на пряме підключення до банку або імпорт структурованих файлів. Алгоритм категоризації є закритим і не підлягає налаштуванню на рівні стратегії або моделі [2].

1.5.3 Finmap

Finmap - українська SaaS-платформа для фінансового обліку, орієнтована на малий і середній бізнес. На відміну від QuickBooks та Xero, Finmap спрямована переважно на управлінський облік: облік грошових потоків, статей витрат і доходів. Платформа підтримує ручне введення транзакцій та імпорт з файлів, проте автоматична ШІ-категоризація у базовому функціоналі відсутня.

Finmap є актуальним прикладом рішення, орієнтованого на українські реалії: підтримка гривні, інтеграція з популярними українськими банками (ПриватБанк, monobank, Ощадбанк) та адаптований план рахунків. Однак відсутність автоматичного витягування транзакцій з PDF і ШІ-категоризації є суттєвим обмеженням порівняно з рішеннями на базі LLM [10].

1.5.4 Порівняльний аналіз існуючих систем

Для систематизації результатів огляду було проведено порівняльний аналіз розглянутих систем за ключовими критеріями, що визначають ефективність обробки та категоризації банківських виписок. Результати представлено у таблиці 1.1.

Таблиця 1.1 - Порівняльний аналіз систем обробки та категоризації банківських транзакцій

Критерій	QuickBooks	Xero	Finmap	Розроблювана система
Нативна підтримка PDF-виписок	Ні	Ні	Ні	Так
Автоматичне витягування транзакцій	Тільки CSV/OFX	Тільки CSV/OFX	Ні	Так (LLM)
Правила на основі ключових слів	Так	Так	Ні	Так
ШІ-категоризація	Так (ML, закрита)	Так (ML, закрита)	Ні	Так (ML, відкрита)

Продовження таблиці 1.1

Вибір стратегії класифікації	Ні	Ні	Ні	Так
Ручне редагування результатів	Так	Так	Так	Так
Експорт CSV/Excel	Так	Так	Так	Так
Адаптованість до укр. ринку	Обмежена	Обмежена	Так	Так

Як видно з таблиці 1.1, жодна з розглянутих комерційних систем не підтримує нативну обробку PDF-файлів банківських виписок і не надає користувачу контролю над стратегією ШІ-категоризації. Існуючі системи пропонують закриті алгоритми машинного навчання, які не піддаються налаштуванню, що унеможливлює оцінку або порівняння різних підходів до класифікації в межах однієї платформи.

Таким чином, порівняльний аналіз підтверджує наявність незайнятої ніші: системи, що поєднує нативне витягування транзакцій з PDF засобами LLM, гнучке правилове фільтрування та конфігуровану ШІ-категоризацію з можливістю вибору й порівняння стратегій промпт-інжинірингу.

ВИСНОВОК ДО РОЗДІЛУ 1

У першому розділі було проведено комплексний огляд предметної області та актуальних підходів до автоматизації обробки і категоризації банківських виписок. Визначено, що ручна обробка транзакцій є значущою проблемою для підприємств МСБ: вона є трудомісткою, призводить до неузгодженості класифікації та обмежує можливості для стратегічного фінансового аналізу.

Розглянуто еволюцію технічних методів категоризації - від ручної обробки та правил на основі ключових слів до класичного машинного навчання та великих мовних моделей. Встановлено, що LLM мають принципові переваги перед попередніми підходами завдяки здатності до семантичного розуміння тексту, нативній підтримці документів і можливості роботи без попереднього навчання на специфічних даних підприємства.

Аналіз мотивацій для автоматизації показав, що впровадження ШІ у бухгалтерські процеси є не лише засобом скорочення рутинного навантаження, а й стратегічним рішенням, що підвищує якість фінансових даних і відкриває нові можливості для аналітики.

Огляд існуючих систем - QuickBooks Online, Xero та Finmap - виявив спільне обмеження: жодна з них не підтримує нативне витягування транзакцій з PDF-файлів і не надає користувачу контролю над алгоритмом категоризації. Порівняльний аналіз підтверджує актуальність розробки системи, що поєднує нативну обробку PDF через LLM, гнучке правилове фільтрування та конфігуровану ШІ-категоризацію з можливістю вибору і порівняння стратегій промпт-інжинірингу. Саме ця концепція складає основу розроблюваної в рамках дипломного проєкту системи.

РОЗДІЛ 2 АНАЛІЗ АЛГОРИТМІВ І ТЕХНОЛОГІЙ ДЛЯ РОЗРОБКИ СИСТЕМИ

2.1 Аналіз архітектурних підходів до побудови конвеєру обробки

Центральним алгоритмічним рішенням у проєктуванні системи є вибір підходу до категоризації транзакцій. На етапі аналізу було розглянуто три можливі варіанти побудови конвеєру обробки, кожен з яких має відмінні характеристики за точністю, вартістю та прозорістю.

Перший варіант - повністю правилова категоризація - передбачає, що всі транзакції класифікуються виключно на основі наборів ключових слів, визначених користувачем. Підхід є прозорим і не потребує витрат на API, однак, як показано у Розділі 1, він не здатний опрацьовувати нові або нестандартні описи транзакцій. Оскільки обслуговування бази правил є постійним завданням і не масштабується зі зростанням різноманітності транзакцій, цей варіант не відповідає вимогам до повноти категоризації.

Другий варіант - повністю ШІ-категоризація - передбачає, що кожна транзакція без винятку передається до великої мовної моделі. Такий підхід забезпечує найвищу гнучкість і мінімальні вимоги до налаштування з боку користувача, проте призводить до необґрунтованих витрат: однозначні та повторювані транзакції (банківські комісії, відомі постачальники, регулярні підписки) не потребують ШІ-аналізу і можуть бути надійно оброблені детерміністичними правилами. Передача таких транзакцій до LLM збільшує вартість і затримку без приросту точності.

Третій варіант - двоетапний конвеєр - поєднує переваги обох підходів. На першому етапі транзакції перевіряються правилами на основі ключових слів: збіги отримують категорію з впевненістю 100% і більше не обробляються. На другому етапі транзакції, що не збіглися з жодним

правилом, передаються до LLM. Такий підхід мінімізує витрати токенів за рахунок виключення тривіальних випадків і водночас зберігає здатність ШІ-компоненту опрацьовувати складні та неоднозначні описи.

Порівняльна характеристика розглянутих підходів наведена у таблиці 2.1.

Таблиця 2.1 - Порівняння підходів до побудови конвеєру категоризації

Критерій	Тільки правила	Тільки ШІ	Двоетапний конвеєр
Точність для відомих транзакцій	Висока	Висока	Висока
Точність для нових транзакцій	Низька	Висока	Висока
Вартість API	Відсутня	Висока	Низька (часткова)
Латентність	Низька	Висока	Середня
Прозорість результату	Повна	Часткова	Повна для правил, часткова для ШІ
Потреба у налаштуванні	Висока	Низька	Середня

На підставі аналізу для реалізації системи обрано двоетапний конвеєр як такий, що забезпечує оптимальний баланс між точністю, вартістю та зручністю для кінцевого користувача. Правила виконуються першими і мають абсолютний пріоритет; до LLM потрапляють лише транзакції, не охоплені правилами.

2.2 Алгоритм правилової категоризації

Правилова категоризація є першим етапом конвеєру обробки і реалізується як детерміністичний алгоритм зіставлення ключових слів. Кожне правило містить два поля: масив ключових слів і цільову категорію. Правила визначаються користувачем через окремий розділ інтерфейсу і зберігаються у базі даних.

Алгоритм обробляє транзакції у такій послідовності:

1. Для кожної транзакції опис приводиться до нижнього регістру.

2. Правила перевіряються у порядку їх збереження; для кожного правила перевіряється, чи містить опис транзакції хоча б одне з ключових слів правила (як підрядок, без урахування регістру).

3. При першому збігу транзакції призначається відповідна категорія, встановлюється впевненість 100% і прапорець “ruleApplied = true”. Перевірка решти правил припиняється.

4. Транзакції, для яких жодне правило не спрацювало, передаються на другий етап - до ШІ-категоризатора.

Принцип "перший збіг виграє" є навмисним: порядок правил дозволяє користувачу задавати пріоритети. Більш специфічні правила розміщуються вище і перекривають загальніші. Наприклад, правило для "UBER EATS" (категорія "Харчування") матиме пріоритет над правилом для "UBER" (категорія "Транспорт"), якщо розміщене першим.

Перевагою цього підходу є його повна передбачуваність і нульова вартість обчислень: правилова перевірка виконується локально, без API-викликів, і не залежить від доступності зовнішніх сервісів. Це також підвищує надійність системи: навіть за умови недоступності Claude API раніше оброблені правилами транзакції залишаться категоризованими.

2.3 Алгоритм витягування транзакцій з PDF

Витягування структурованих даних з PDF-файлів банківських виписок є технічно нетривіальним завданням: виписки різних банків мають відмінні макети, шрифти та структуру таблиць. Крім того, деякі виписки є сканованими - тобто є зображеннями без текстового шару.

На початковому етапі проєктування розглядався двокроковий підхід: PDF-файл передається до стороннього OCR-сервісу (PDF.co), який повертає текст, після чого текст передається до Claude для структурування у список транзакцій. Аналіз цього підходу виявив суттєві недоліки:

- залежність від стороннього сервісу з окремим ключем API, тарифами та потенційними збоями;
- OCR-артефакти у тексті (особливо для таблиць банківських виписок) знижували точність подальшого витягування;
- подвійна затримка: два послідовні мережеві запити замість одного;
- необхідність явної обробки сканованих PDF як окремого випадку.

Claude API підтримує нативний тип вмісту “document”, який приймає PDF-файл безпосередньо як base64-закодовані байти. Модель обробляє документ через власний механізм бачення, витягує текст, розпізнає структуру таблиці транзакцій і повертає структурований список у вигляді JSON. Сканований PDF обробляється так само, як і текстовий, - без додаткових налаштувань [8].

Обраний підхід - передача PDF безпосередньо до Claude - усуває всі зазначені недоліки: сторонній сервіс виключається з ланцюжка залежностей, кількість API-викликів скорочується до одного, а сканований і текстовий формати обробляються уніфіковано. Єдиним викликом API є запит до Claude з прикріпленим PDF та інструкцією повернути транзакції у форматі JSON-масиву з полями “date”, “description” та “amount”.

2.4 Стратегії промпт-інжинірингу для категоризації

Після витягування транзакцій і застосування правил некатегоризовані транзакції передаються до ШІ-компоненту. Ключовим параметром цього компоненту є стратегія промпт-інжинірингу - спосіб формулювання запиту до мовної моделі. У системі реалізовано три стратегії, кожна з яких відрізняється структурою промпту, обсягом токенів і очікуваною точністю.

2.4.1 Базова інструкція

Базова стратегія (zero-shot) є найпростішою за структурою. Промпт містить:

- перелік доступних категорій;
- список транзакцій для категоризації;
- пряму інструкцію призначити одну категорію кожній транзакції;
- специфікацію вихідного формату - JSON-масив з полями “transactionId”, “category” та “confidence”.

Модель отримує мінімальний контекст і виконує категоризацію на основі власних знань, накопичених під час попереднього навчання. Перевагою стратегії є найнижча кількість вхідних токенів і, відповідно, мінімальна вартість та затримка. Обмеження проявляється на неоднозначних транзакціях: без прикладів і без явного механізму міркування модель може по-різному інтерпретувати схожі описи [7].

2.4.2 Few-shot навчання

Few-shot стратегія розширює базову шляхом додавання до промпту п'яти заздалегідь підготовлених прикладів правильних категоризацій. Приклади

охоплюють різні категорії витрат і доходів, забезпечуючи модель демонстрацією очікуваного стилю відповіді.

Приклади у промпті виконують роль неявного "контракту": модель орієнтується на їхній формат і логіку вибору категорії при обробці нових транзакцій. Це підвищує узгодженість відповідей - особливо для транзакцій, семантично подібних до наведених прикладів. Недоліком є збільшення вхідних токенів приблизно на 30% порівняно з базовою стратегією за рахунок фіксованого обсягу прикладів [5].

2.4.3 Ланцюжок міркувань

Стратегія ланцюжка міркувань (chain-of-thought) інструктує модель виконувати покрокове логічне міркування перед присвоєнням категорії. Промпт визначає чотири явні кроки для кожної транзакції:

1. Ідентифікувати постачальника або тип операції за описом.
2. Врахувати знак суми (позитивна - надходження, негативна - витрата).
3. Зіставити з найбільш відповідною категорією зі списку.
4. Призначити показник впевненості від 0 до 100.

Вихідний JSON додатково містить поле “reasoning” - коротке обґрунтування обраної категорії. Це поле корисне для відлагодження та перевірки результатів: користувач може переглянути, чому модель обрала ту чи іншу категорію.

Стратегія демонструє найвищу точність для неоднозначних транзакцій, де поверхневого аналізу опису недостатньо. Наприклад, транзакція з описом платіжного процесора без назви реального постачальника може бути коректно інтерпретована через аналіз суми та контексту. Компромісом є найвища кількість вихідних токенів серед трьох стратегій, що збільшує вартість і затримку [6].

2.4.4 Порівняльна характеристика стратегій

Узагальнена порівняльна характеристика трьох стратегій наведена у таблиці 2.2.

Таблиця 2.2 - Порівняння стратегій промпт-інжинірингу

Критерій	Базова інструкція	Few-shot	Ланцюжок міркувань
Вхідні токени	Низькі	Середні (~+30%)	Середні
Вихідні токени	Низькі	Низькі	Високі (+reasoning)
Латентність	Низька	Середня	Висока
Точність (однозначні транзакції)	Висока	Висока	Висока
Точність (неоднозначні транзакції)	Середня	Середня-висока	Висока
Поле reasoning у відповіді	Ні	Ні	Так
Придатність для налагодження	Обмежена	Обмежена	Висока

За результатами порівняння для використання за замовчуванням у системі обрано стратегію ланцюжка міркувань як таку, що забезпечує найвищу точність на неоднозначних транзакціях - найскладнішому класі вхідних даних. Порівняльна оцінка всіх трьох стратегій у поєднанні з двома моделями виконується у рамках дослідницького модуля бенчмаркінгу, результати якого описано у Розділі 4.

2.5 Вибір технологій реалізації

2.5.1 Фреймворк для веброзробки

При виборі фреймворку для реалізації вебзастосунку розглядалися три варіанти: Next.js 16 з App Router, Remix та класична комбінація Express.js із окремим Vite-застосунком на React.

Підхід Express + Vite передбачає розділення серверної та клієнтської частин на два окремих проєкти. Це ускладнює спільне використання типів TypeScript між клієнтом і сервером, потребує дублювання конфігурації та окремого налаштування SSE (Server-Sent Events) для потокового виводу прогресу обробки.

Remix є сучасною альтернативою з хорошою підтримкою серверного рендерингу, проте має менш зрілу екосистему порівняно з Next.js і не забезпечує нативної підтримки SSE у route handlers.

Next.js 16 з App Router обрано з таких причин:

- єдина кодова база для клієнта і сервера дозволяє використовувати спільні TypeScript-типи (зокрема, “Transaction”, “ProcessLog”, “QuickBooksCategory”) без дублювання;
- route handlers підтримують “ReadableStream” нативно, що дозволяє реалізувати SSE для потокового виводу прогресу обробки без додаткової інфраструктури;
- React 19 з конкурентними можливостями (Suspense, streaming) підключається без додаткового налаштування;
- безконфігураційне розгортання на Vercel із автоматичним визначенням середовища виконання [11].

2.5.2 ШІ-модель та провайдер

Для задачі категоризації транзакцій розглядалися три провідні LLM-провайдери: Anthropic Claude, OpenAI GPT та Google Gemini.

OpenAI GPT-4o підтримує обробку зображень, однак не має нативного типу вмісту для PDF-документів. Передача PDF потребує або конвертації сторінок у зображення (з втратою якості та зростанням токенів), або попередньої конвертації у текст стороннім інструментом. Це суперечить вимозі мінімізувати кількість зовнішніх залежностей.

Google Gemini підтримує завантаження файлів через Files API, однак ця функція реалізована як окремий двокроковий процес (завантаження + посилання), а не нативний тип вмісту в повідомленні. Крім того, на момент проєктування системи відповідні функції мали менш зрілу документацію і стабільність.

Claude API від Anthropic обрано з двох ключових причин:

- нативний тип вмісту “document” приймає base64-закодований PDF безпосередньо у запиті, що дозволяє реалізувати весь процес витягування транзакцій одним API-викликом без сторонніх сервісів;
- модель стабільно повертає структурований JSON відповідно до інструкції без додаткових обгорток або пояснювального тексту, що є критичним для парсингу відповіді у виробничому коді [8].

У системі використовуються дві моделі: “claude-haiku-4-5” (оптимізована за вартістю і швидкістю) та “claude-sonnet-4-5” (вища якість міркування). Вибір між ними є параметром бенчмаркінгу.

2.5.3 Система зберігання даних

Для зберігання облікових даних, правил категоризації та шаблонів промптів розглядалися три варіанти: Supabase, Firebase Firestore та PlanetScale.

Firebase Firestore є NoSQL базою даних з документно-орієнтованою моделлю. Для структурованих бухгалтерських даних (правила, категорії) SQL-запити є природнішими, ніж NoSQL-колекції. Крім того, Firestore використовує пропрієтарний API запитів замість стандартного SQL, що ускладнює міграцію. Модель ціноутворення базується на кількості операцій читання/запису, що є менш передбачуваним для додатків зі змінним навантаженням.

PlanetScale надає managed MySQL, проте не включає вбудованого сервісу автентифікації, що потребує підключення окремого рішення (наприклад, NextAuth) і збільшує складність архітектури.

Supabase обрано як основний сервіс зберігання даних з таких причин:

- managed PostgreSQL із стандартним SQL та підтримкою Row Level Security (RLS) на рівні бази даних - контроль доступу не потребує реалізації на рівні застосунку;
- вбудований сервіс автентифікації з підтримкою сесій та JWT, що дозволяє реалізувати захищений вхід без стороннього auth-провайдера;
- TypeScript-клієнт із типами, згенерованими зі схеми бази даних, забезпечує типову безпеку при роботі з даними [12].

Транзакційні дані (витягнуті та категоризовані) не зберігаються в базі даних - вони утримуються у “sessionStorage” браузера і доступні лише протягом поточної сесії. Це навмисне архітектурне рішення, що спрощує систему і виключає необхідність реалізовувати ізоляцію даних між користувачами.

2.5.4 Зведена таблиця технологічних рішень

Узагальнений огляд усіх ключових технологічних рішень із обраними варіантами та відхиленими альтернативами наведено у таблиці 2.3.

Таблиця 2.3 - Зведена таблиця технологічних рішень

Компонент	Обрана технологія	Розглянуті альтернативи	Ключова перевага
Повний стек	Next.js 16 App Router	Express + Vite, Remix	Спільні TypeScript-типи, нативний SSE
Мова	TypeScript 5.7	JavaScript	Типова безпека на межі клієнт-сервер
UI-компоненти	shadcn/ui + Radix UI	MUI, Ant Design	Власний код, без design debt
Стилізація	Tailwind CSS	CSS Modules, styled-components	Утилітарний підхід, швидка розробка
ШІ-провайдер	Anthropic Claude	OpenAI GPT, Google Gemini	Нативна підтримка PDF-документів
Auth + DB	Supabase	Firebase, PlanetScale	PostgreSQL + Auth + RLS в одному сервісі
Обробка PDF	Claude document API	PDF.co + OCR	Один API-виклик, без сторонніх сервісів

Продовження таблиці 2.3

Експорт	xlsx	ExcelJS, Papa Parse	Легковагий, підтримує CSV та XLSX
Деплой	Vercel	Railway, Render	Нульова конфігурація для Next.js

ВИСНОВОК ДО РОЗДІЛУ 2

У другому розділі проведено аналіз алгоритмічних і технологічних рішень, що складають основу розроблюваної системи.

Порівняльний аналіз підходів до побудови конвеєру обробки показав, що двоетапна архітектура - правила на основі ключових слів із подальшим ШІ-аналізом - забезпечує оптимальний баланс між точністю та вартістю. Правилова категоризація обробляє однозначні та регулярні транзакції з нульовими витратами токенів, тоді як ШІ-компонент зосереджується на складних і неоднозначних випадках.

Описано алгоритм правилової категоризації на основі пошуку підрядків та принципу першого збігу, а також обґрунтовано вибір нативної обробки PDF через Claude document API замість двокрокового OCR-підходу - як такий, що усуває сторонню залежність і зменшує кількість точок відмови.

Детальний розгляд трьох стратегій промпт-інжинірингу - базової інструкції, few-shot навчання та ланцюжка міркувань - показав, що стратегії відрізняються за точністю, вартістю та латентністю. Для використання за замовчуванням обрано ланцюжок міркувань як такий, що забезпечує найвищу точність для неоднозначних транзакцій; порівняльне оцінювання всіх стратегій виноситься до дослідницького розділу.

Аналіз технологічного стеку підтвердив доцільність обраних рішень: Next.js 16 для повного стеку, Claude API для обробки PDF і категоризації, Supabase для зберігання даних і автентифікації. Кожен вибір обґрунтовано порівнянням з альтернативами за критеріями, що відповідають вимогам системи.

РОЗДІЛ 3 ДЕТАЛІ РОЗРОБКИ СИСТЕМИ

3.1 Загальна архітектура системи

Система інтелектуальної обробки та категоризації банківських виписок побудована як повностековий вебзастосунок з чіткою трирівневою архітектурою: рівень представлення, рівень застосунку та рівень даних. Таке розмежування забезпечує незалежність рівнів, полегшує підтримку коду та дозволяє тестувати бізнес-логіку ізольовано від транспортного шару.

3.1.1 Рівень представлення

Рівень представлення реалізований на базі Next.js 16 з App Router та React 19. Компоненти інтерфейсу розроблені з використанням бібліотеки shadcn/ui, яка побудована на доступних примітивах Radix UI, та стилізовані через Tailwind CSS. Такий підхід забезпечує узгодженість візуального оформлення без прив'язки до конкретної design system з фіксованими стилями.

Застосунок надає автентифікованим користувачам доступ до таких маршрутів:

Таблиця 3.1 - Маршрути застосунку та їх призначення

Маршрут	Призначення
/	Завантаження PDF, перегляд витягнутих транзакцій, експорт у CSV/XLSX
/accounts	Управління планом рахунків (CRUD)
/rules	Управління правилами ключових слів для категоризації (CRUD)

Продовження таблиці 3.1

`/prompts`	Редагування шаблонів промптів для стратегій ШІ
`/benchmark`	Запуск бенчмарку стратегій ШІ, перегляд порівняльного звіту
`/login`	Автентифікація за email/паролем

Головна сторінка "`/`" є ключовою точкою взаємодії: вона реалізує повний робочий цикл - від завантаження файлу до отримання категоризованих результатів. Форма завантаження приймає PDF-файли з валідацією типу та розміру на клієнтській стороні (не більше 10 МБ). Після обробки транзакції відображаються в інтерактивній таблиці з можливістю ручного коригування категорій та подальшого експорту.

3.1.2 Рівень застосунку

Серверна логіка реалізована у вигляді API-маршрутів Next.js у директорії "`app/api/`". Ключовим маршрутом є "`POST /api/process`", який реалізує основний конвеєр обробки та повертає результати у форматі Server-Sent Events (SSE). Потоковий підхід обраний навмисно: обробка PDF-файлу великим мовним моделем може тривати від 5 до 30 секунд, і синхронна відповідь призвела б до таймауту або видимого "зависання" інтерфейсу. SSE-потік дозволяє клієнту отримувати оновлення про хід обробки в реальному часі - по завершенні кожного етапу конвеєру [13].

Модуль штучного інтелекту `lib/ai/` виконує всю бізнес-логіку обробки і є повністю ізольованим від HTTP-шару. Він оперує виключно типізованими TypeScript-об'єктами і не містить жодних посилань на об'єкти запиту чи

відповіді. Така ізоляція дозволяє бенчмарк-раннеру викликати ті самі ШІ-функції безпосередньо, без HTTP-запитів.

3.1.3 Рівень даних

Рівень даних реалізований на базі Supabase - хмарного сервісу, що надає managed PostgreSQL з вбудованою автентифікацією та підтримкою Row Level Security (RLS). У базі даних зберігаються три сутності: план рахунків (`accounts`), правила категоризації (`category_rules`) та шаблони промптів (`prompts`).

Транзакційні дані - витягнуті та категоризовані рядки виписки - навмисно не зберігаються у базі даних. Вони утримуються у `sessionStorage` браузера і доступні лише протягом поточної сесії. Це архітектурне рішення спрощує систему та виключає необхідність реалізовувати ізоляцію даних між користувачами на рівні бази даних.

Загальну схему архітектурних рівнів та взаємозв'язки між ними показано на рисунку 3.1.

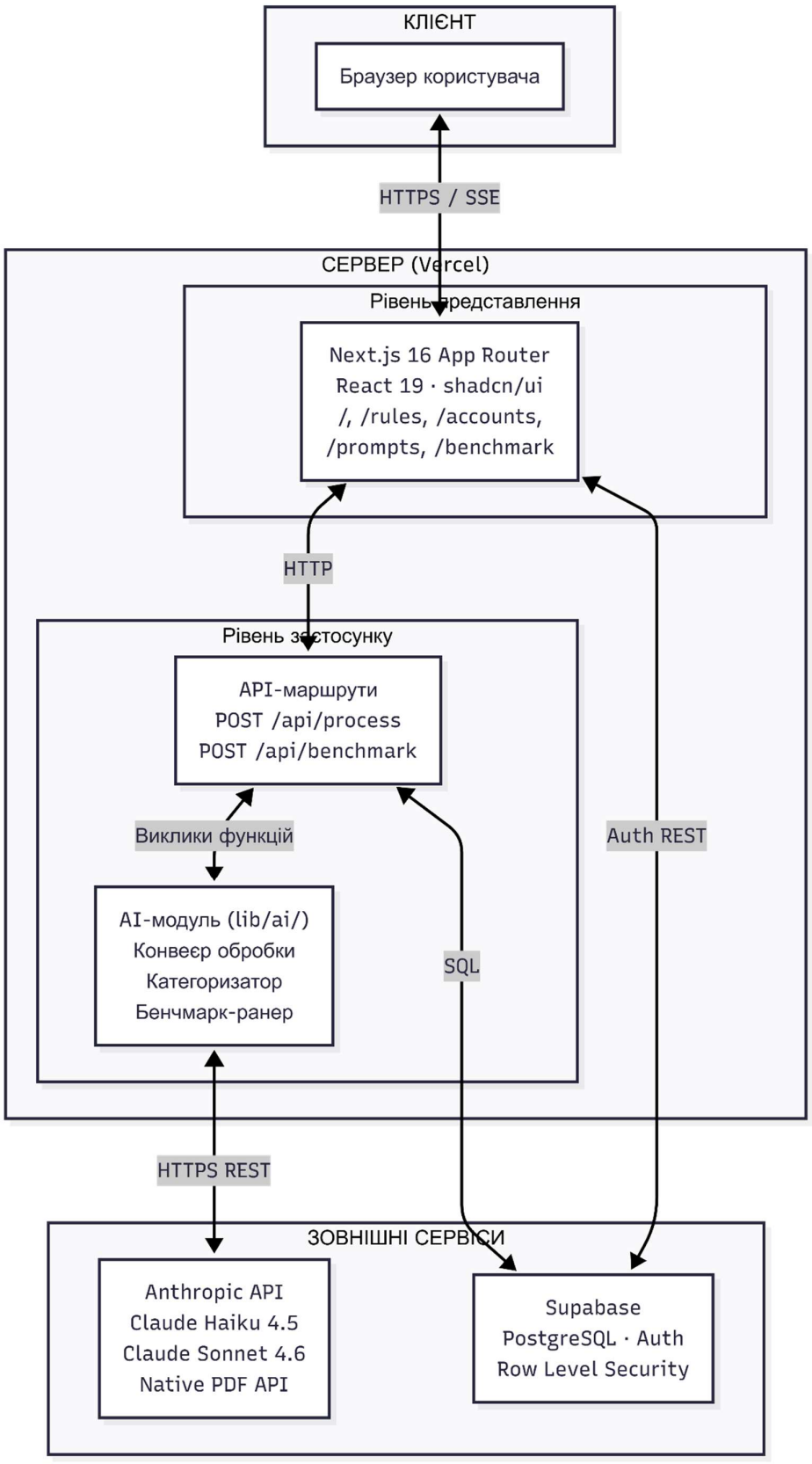


Рисунок 3.1 - Загальна архітектура системи

3.2 Схема бази даних

База даних містить три таблиці, структура яких визначена відповідно до вимог системи.

Таблиця “accounts” зберігає план рахунків користувача. Кожен запис містить унікальний ідентифікатор, назву рахунку, тип рахунку (активи, зобов'язання, власний капітал, дохід, витрати), а також ідентифікатор користувача для ізоляції даних через RLS. Записи цієї таблиці відповідають категоріям QuickBooks, що є стандартом бухгалтерського обліку для МСБ.

Таблиця “category_rules” зберігає правила категоризації на основі ключових слів. Структура запису включає: масив ключових слів (“keywords TEXT[]”), цільову категорію (“category TEXT”), ідентифікатор користувача та позначку активності. Зберігання ключових слів як масиву PostgreSQL дозволяє виконувати пошук підрядка в масиві одним SQL-запитом без нормалізації у підтаблицю.

Таблиця “prompts” зберігає шаблони промптів для кожної зі стратегій ШІ. Структура містить ідентифікатор стратегії (“strategy_id TEXT”), текст промпту (“prompt_text TEXT”) та ідентифікатор користувача. Надання можливості редагувати промпти через інтерфейс дозволяє бухгалтеру налаштовувати формулювання під особливості своєї галузі без зміни коду.

Row Level Security налаштована на рівні бази даних для всіх трьох таблиць. Кожна RLS-політика перевіряє відповідність “user_id” запису ідентифікатору з активного JWT-токена Supabase. Таким чином, навіть якщо помилка в логіці застосунку пропустить неавторизований запит, база даних відхилить його автоматично [12].

Таблиця 3.2 - Схема бази даних

Таблиця	Ключові поля	Тип	RLS
accounts	id, name, type, user_id	UUID, TEXT, TEXT, UUID	Так
category_rules	id, keywords, category, user_id, active	UUID, TEXT[], TEXT, UUID, BOOL	Так
prompts	id, strategy_id, prompt_text, user_id	UUID, TEXT, TEXT, UUID	Так

3.3 Реалізація конвеєру обробки PDF

3.3.1 Огляд конвеєру

Основний конвеєр обробки запускається при завантаженні користувачем PDF-файлу банківської виписки та складається з п'яти послідовних етапів:

1. Валідація файлу на клієнті (тип, розмір)
2. Витягування транзакцій з PDF через Claude document API
3. Правилова категоризація (детерміністичний етап)
4. ШІ-категоризація транзакцій, що не збіглися з правилами
5. Об'єднання результатів та відправка клієнту через SSE

На рисунку 3.2 показано детальну схему потоку даних конвеєру.

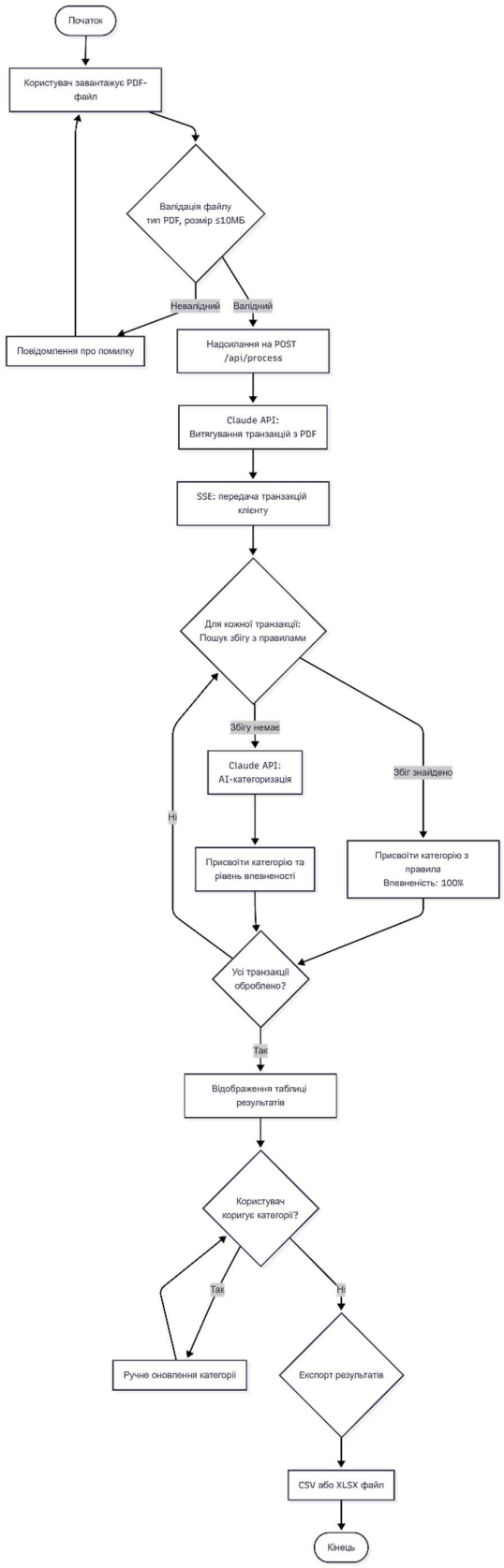


Рисунок 3.2 - Потік даних основного конвеєру обробки

3.3.2 Обробник PDF (“lib/ai/pdf-processor.ts”)

Функція “processPDF” приймає “Buffer” з байтами PDF-файлу, кодує його у base64 і надсилає до Claude API з використанням типу вмісту “document”. Це нативний спосіб передачі PDF у Claude, що відрізняється від передачі зображень або текстових рядків і дозволяє моделі безпосередньо зчитувати структуру документа.

Запит до Claude формується таким чином, що повідомлення містить два елементи: блок типу “document” з base64-кодованим PDF та блок типу “text” з інструкцією витягування. Інструкція задає точний формат відповіді - масив JSON-об'єктів з полями “date”, “description” та “amount” - і забороняє будь-яке обрамлення у вигляді markdown або пояснювального тексту.

Після отримання відповіді функція “parseTransactions” витягує JSON-рядок за допомогою регулярного виразу “\[[\s\S]*\]”, що забезпечує стійкість до мінімальних відхилень у форматуванні виводу. Кожному витягнутому запису присвоюється унікальний UUID через “randomUUID()” для ідентифікації при подальшій категоризації.

Для підвищення надійності реалізовано механізм повторних спроб з експоненційним відступом: у разі помилки API виконується одна повторна спроба через 1 секунду. Якщо обидві спроби невдалі, помилка передається вище по стеку і транслюється клієнту через SSE-подію типу “error”.

3.3.3 Потоковий відгук через Server-Sent Events

Server-Sent Events обрано як механізм транспортування результатів замість звичайної HTTP-відповіді з трьох причин.

По-перше, Next.js 16 App Router підтримує “ReadableStream” у route handlers нативно, без потреби у WebSocket-сервері чи сторонніх бібліотеках. По-друге, SSE є однонаправленим протоколом (сервер → клієнт), що повністю

задовольняє потребам конвеєру: клієнт надсилає один запит, а сервер повертає послідовність подій у міру завершення кожного етапу. По-третє, браузери мають вбудовану підтримку SSE через “`EventSource`” API з автоматичним повторним підключенням.

Конвеєр надсилає події трьох типів: “`progress`” (хід виконання з описом поточного кроку), “`complete`” (фінальний масив категоризованих транзакцій) та “`error`” (деталі помилки, якщо конвеєр не виконаний). Клієнтський код прослуховує ці події та оновлює стан React-компонента відповідно.

3.4 Модуль штучного інтелекту

3.4.1 Патерн стратегій

Модуль “`lib/ai/`” побудований на патерні стратегій (Strategy Pattern) - поведінковому шаблоні, що дозволяє визначити сімейство алгоритмів, інкапсулювати кожен з них та зробити їх взаємозамінними без зміни клієнтського коду [14]. У даному контексті "алгоритм" - це спосіб формування промпту для Claude: базова інструкція, few-shot навчання або ланцюжок міркувань.

Інтерфейс “`CategorizationStrategy`” визначає єдиний метод:

```
interface CategorizationStrategy {
  name: string
  description: string
  categorize(
    transactions: Transaction[],
    model: string,
    availableCategories: QuickBooksCategory[]
  ): Promise<StrategyResult>
}
```

Де “`StrategyResult`” містить масив результатів категоризації, кількість використаних токенів (вхідних і вихідних) та час виконання у мілісекундах. Усі три реалізовані стратегії задовольняють цей інтерфейс. Модуль

“categorizer.ts” приймає рядок “StrategyName” і повертає відповідну реалізацію. Додавання нової стратегії потребує лише створення нового файлу - жодних змін у конвеєрі обробки або бенчмарк-раннері не потрібно.

Структура директорії ШІ-модуля:

```
lib/ai/
├── pdf-processor.ts        # Витягування транзакцій через Claude document API
├── categorizer.ts          # Застосування обраної стратегії до транзакцій
├── strategies/
│   ├── types.ts            # Спільні інтерфейси та перерахування
│   ├── basic.ts            # Стратегія прямої інструкції
│   ├── few-shot.ts         # Стратегія з прикладами
│   └── chain-of-thought.ts # Стратегія ланцюжка міркувань
└── benchmark.ts            # Бенчмарк-раннер всіх 6 комбінацій
```

3.4.2 Стратегія базової інструкції (“basic”)

Базова стратегія є найпростішою реалізацією і служить нижньою межею точності у порівняльному оцінюванні. Промпт містить чотири елементи: роль (бухгалтерський асистент), список доступних категорій, масив транзакцій для категоризації та специфікацію формату відповіді (JSON-масив без markdown-обрамлення).

Ця стратегія не надає Claude жодних прикладів або явних кроків міркування - лише перелік категорій і задачу. Очікувана характеристика: мінімальна кількість вхідних токенів, найнижча латентність, але нижча точність для неоднозначних транзакцій, де опис сам по собі не дає однозначної відповіді на питання категорії.

3.4.3 Стратегія з прикладами (“few-shot”)

Few-shot стратегія передує переліку транзакцій блоком з п'яти вручну підібраних прикладів правильної категоризації. Приклади охоплюють

різноманітні категорії витрат і служать неявним "контрактом" - Claude орієнтується на них при виборі категорій та формуванні відповіді.

Прикладами є транзакції: витрати на офісні матеріали (категорія "Office Expenses"), авіаквитки ("Travel"), ресторан ("Meals & Entertainment"), підписка на SaaS-продукт ("Software & Subscriptions") та банківська комісія ("Bank Fees"). Такий добір прикладів відображає найпоширеніші типи транзакцій для малого підприємства.

Стратегія додає фіксований обсяг вхідних токенів (приблизно 30% більше порівняно з базовою), що підвищує вартість кожного виклику. Водночас точність для транзакцій, схожих за типом на наведені приклади, очікується вищою завдяки механізму in-context learning [5].

3.4.4 Стратегія ланцюжка міркувань (“chain-of-thought”)

Chain-of-thought стратегія явно інструктує Claude виконати чотирикрокове міркування перед тим, як присвоїти категорію:

1. Визначити постачальника або тип витрат з опису транзакції
2. Проаналізувати знак суми (від'ємний - витрата, додатній - дохід)
3. Співставити з найбільш відповідною категорією зі списку
4. Визначити рівень впевненості (0-100)

Вихідний формат відповідно розширено полем “`reasoning`” - одним реченням, що підсумовує хід міркування для конкретної транзакції.

Ця стратегія генерує найбільший обсяг вихідних токенів через включення міркувань у відповідь, що збільшує як вартість, так і латентність. Водночас вона очікувано забезпечує найвищу точність для граничних випадків - транзакцій з неоднозначним описом, де підсумок суми або контекст є вирішальними [6].

Таблиця 3.3 - Порівняльні характеристики стратегій ШІ

Стратегія	Вхідні токени	Вихідні токени	Очікувана точність	Оптимальний сценарій
Базова (“basic”)	Мінімальні	Мінімальні	Базова	Однозначні описи
Few-shot	+30% до базової	Мінімальні	Вища	Типові витрати МСБ
Chain-of-thought	+20% до базової	+20% до базової	Максимальна (reasoning)	Неоднозначні транзакції

3.4.5 Вибір моделі

Для бенчмарку обрано дві моделі Anthropic з різним балансом між вартістю та якістю.

claude-haiku-4-5 - компактна модель, оптимізована для швидкості та низькою вартості (вхідні токени: $1.00/1M, вихідні: $5.00/1M). Очікується, що вона добре справляється зі стандартними транзакціями, де опис є достатньо виразним. Використовується як модель за замовчуванням у продуктивному режимі для мінімізації витрат на API.

claude-sonnet-4-5 - потужніша модель з кращим слідуванням інструкціям і глибшим міркуванням (вхідні: $3.00/1M, вихідні: $15.00/1M). Очікується, що вона перевершує Haiku на граничних випадках, особливо у поєднанні зі стратегією chain-of-thought. Включена до бенчмарку для кількісного визначення компромісу якість-вартість.

3.5 Правилова категоризація

3.5.1 Алгоритм зіставлення ключових слів

Правилова категоризація є першим і безкоштовним етапом конвеєру. Вона реалізована у “`lib/category-rules.ts`” як детерміністичний алгоритм зіставлення підрядків. Кожне правило у базі даних містить масив ключових слів і цільову категорію. Правила визначаються і впорядковуються користувачем.

Функція `applyRulesToTransactions` обробляє транзакції у такому порядку:

1. Опис транзакції приводиться до нижнього регістру
2. Правила перебираються у порядку їх збереження
3. Для кожного правила перевіряється наявність хоча б одного ключового слова у описі (пошук підрядка без урахування регістру)
4. При першому збігу транзакції присвоюється категорія, встановлюється “`confidence = 100`” та “`ruleApplied = true`”, перебір правил припиняється
5. Транзакції без збігу передаються до ШІ-категоризатора

Принцип "перший збіг виграє" є навмисним: більш специфічні правила розміщуються вище і мають перевагу над загальними. Наприклад, правило для ключового слова "UBER EATS" (категорія "Харчування") розміщується вище правила для "UBER" (категорія "Транспорт") і спрацює першим для відповідних транзакцій.

3.5.2 Економічна доцільність правилового етапу

Правилова категоризація усуває витрати токенів для регулярних, передбачуваних транзакцій. Типова банківська виписка містить значну частку повторюваних транзакцій: щомісячні підписки, регулярні постачальники, банківські комісії. Після первинного налаштування правил ці транзакції обробляються без жодних витрат на API.

У виробничому середовищі правильно налаштований набір правил може покрити від 40% до 70% транзакцій типового підприємства МСБ, зменшуючи відповідним чином витрати на Claude API. Бенчмарк вимірює лише ШІ-компонент (транзакції, що не збіглися з правилами), що відповідає реальному розподілу навантаження у системі.

3.6 Механізм бенчмаркінгу

3.6.1 Мета та структура

Бенчмарк-модуль (“`lib/ai/benchmark.ts`”) призначений для емпіричного порівняння всіх 6 комбінацій: 2 моделі × 3 стратегії. Його виклик доступний через маршрут “`/benchmark`” та відповідний API-ендпоінт “`POST /api/benchmark`”.

Конвеєр бенчмарку виконується у такій послідовності:

1. Користувач завантажує тестовий PDF на сторінці “`/benchmark`”

2. “`pdf-processor.ts`” витягує транзакції один раз (спільно для всіх 6 комбінацій)

3. Для кожної комбінації викликається відповідна стратегія з відповідною моделлю

4. Результати кожної комбінації оцінюються за чотирма метриками

5. Markdown-звіт автоматично зберігається до “`docs/research/benchmarks/`” з міткою часу

6. Зведена таблиця повертається на сторінку “`/benchmark`”

3.6.2 Метрики оцінювання

Кожна комбінація оцінюється за чотирма метриками:

Точність (Accuracy) - частка транзакцій, яким присвоєно очікувану категорію. Обчислюється відносно еталонної розмітки, наданої разом з тестовим PDF.

Латентність (Latency) - астрономічний час від початку виклику API до отримання повної відповіді у мілісекундах. Вимірюється за допомогою “`Date.now()`” безпосередньо навколо виклику “`client.messages.create()`”.

Вартість токенів (Token Cost) - розрахункова вартість у доларах США на основі кількості вхідних і вихідних токенів та опублікованих цін Anthropic. Дозволяє прямо порівнювати економічну ефективність комбінацій.

Розподіл впевненості (Confidence Distribution) - відсоток транзакцій з високою (≥ 80), середньою (50-79) та низькою (< 50) впевненістю. Ця метрика виявляє, чи модель самостійно сигналізує про невизначеність - це важлива характеристика для системи, де бухгалтер може переглядати позначені транзакції.

Таблиця 3.4 - Метрики бенчмарку та їх значення

Метрика	Одиниця	Очікуваний переможець
Точність	%	Sonnet × CoT
Латентність	мс (менше = краще)	Haiku × Basic
Вартість токенів	USD (менше = краще)	Haiku × Basic
Розподіл впевненості	% high / mid / low	Sonnet × CoT

3.6.3 Автоматичне збереження звітів

Після виконання бенчмарку результати автоматично зберігаються у вигляді Markdown-файлу до “`docs/research/benchmarks/`” з назвою у форматі “`YYYY-MM-DD-HH-mm.md`”. Такий підхід забезпечує відтворюваність: кожен запуск бенчмарку зберігає зріз результатів, що дозволяє порівнювати їх у часі. Ці файли є вихідними даними для розділу 4 дипломної роботи (Експериментальне оцінювання).

3.7 Інтерфейс користувача

3.7.1 Головний екран обробки виписки

Головний екран (“`/`”) реалізує повний робочий цикл бухгалтера у трьох станах: завантаження, обробка та перегляд результатів.

У стані завантаження відображається зона drag-and-drop з валідацією типу файлу. При спробі завантажити файл не PDF або файл понад 10 МБ відображається відповідне повідомлення про помилку до відправки запиту на сервер.

Під час обробки відображається індикатор прогресу з описом поточного кроку конвеєру - текст оновлюється через SSE-події “`progress`”. Це усуває візуальну паузу та інформує користувача про хід виконання.

Після обробки транзакції відображаються в інтерактивній таблиці. Кожен рядок містить дату, опис, суму, присвоєну категорію, рівень впевненості та ознаку того, чи була застосована правилова категоризація. Категорію можна змінити вручну через випадний список. Після перевірки та корекцій доступні кнопки експорту у CSV або XLSX.

3.7.2 Управління планом рахунків і правилами

Сторінки “`/accounts`” та “`/rules`” реалізують стандартні CRUD-операції з відповідними таблицями Supabase. Обидві сторінки використовують оптимістичні оновлення UI: зміна відображається миттєво, а запит до бази даних виконується у фоні. У разі помилки стан повертається до попереднього значення.

Сторінка “`/rules`” дозволяє задавати масив ключових слів для кожного правила та цільову категорію. Порядок правил у списку відповідає порядку їх застосування в алгоритмі зіставлення: правила можна переміщати вгору та вниз для встановлення пріоритетів.

3.7.3 Автентифікація

Автентифікація реалізована через Supabase Auth з використанням email/пароля. Middleware “`lib/supabase/middleware.ts`” перехоплює кожен запит до захищених маршрутів, перевіряє JWT-токен та перенаправляє неавтентифікованих користувачів на сторінку входу. Для Next.js 16 middleware реалізовано з урахуванням нового API обробки запитів, що замінив застарілий формат попередніх версій.

При першому запуску застосунку, якщо проєкт Supabase знаходиться у неактивному стані (автоматичне призупинення для безкоштовних тарифів), запит автентифікації може займати до 10 секунд. Для усунення видимого "зависання" форма входу реалізує таймаут 10 секунд з відповідним повідомленням для користувача.

3.8 Розгортання системи

Система розгорнута на платформі Vercel, яка є канонічним хостингом для Next.js-застосунків. Зв'язок репозиторію з Vercel забезпечує автоматичне розгортання при кожному push до основної гілки. Конфігурація середовища включає три змінні: URL проєкту Supabase, анонімний ключ Supabase та API-ключ Anthropic.

Серверсайдні API-маршрути (“`/api/process`”, “`/api/benchmark`”) виконуються як Node.js serverless-функції з максимальним часом виконання 60 секунд. Це достатньо для типового запиту обробки PDF (10-30 секунд залежно від обсягу документа та обраної моделі). Middleware Supabase виконується на Edge Runtime для мінімальної затримки при перевірці токенів.

Таблиця 3.5 - Конфігурація розгортання

Компонент	Runtime	Час очікування
`Middleware (auth)`	Vercel Edge	< 50 мс
`/api/process`	Node.js Serverless	до 60 с
`/api/benchmark`	Node.js Serverless	до 60 с
Статичні сторінки	CDN	< 100 мс

ВИСНОВОК ДО РОЗДІЛУ 3

У третьому розділі описано деталі реалізації системи інтелектуальної обробки та категоризації банківських виписок.

Систему побудовано як повностековий вебзастосунок з трирівневою архітектурою: рівень представлення на базі Next.js 16 і React 19, рівень застосунку з API-маршрутами та модулем ШІ, рівень даних на базі Supabase (PostgreSQL з RLS). Кожен рівень є незалежним і може бути замінений без перебудови решти системи.

Основним технічним рішенням є конвеєр обробки PDF, що складається з двох послідовних етапів: детерміністичної правилової категоризації та ШІ-категоризації для транзакцій, не охоплених правилами. Правиловий етап усуває витрати API для повторюваних транзакцій, тоді як ШІ-компонент забезпечує обробку нестандартних і неоднозначних описів.

Модуль ШІ реалізує патерн стратегій, що дозволяє визначити три взаємозамінні підходи до формування промптів: базова інструкція, навчання на прикладах (few-shot) та ланцюжок міркувань (chain-of-thought). Усі три стратегії тестуються у поєднанні з двома моделями Claude у модулі бенчмаркінгу, що дає змогу емпірично обрати оптимальну конфігурацію для продуктивного використання.

Потоковий транспорт результатів через Server-Sent Events забезпечує зворотній зв'язок з користувачем у реальному часі під час тривалих ШІ-запитів. Безпека даних реалізована через Row Level Security на рівні бази даних, що унеможливлює витік даних між користувачами незалежно від логіки застосунку.

РОЗДІЛ 4 ТЕСТУВАННЯ ТА АНАЛІЗ РЕЗУЛЬТАТІВ РОБОТИ СИСТЕМИ

4.1 Методологія тестування

Тестування системи проводилося у двох фазах з різними цілями та методами.

Перша фаза - демонстраційна - охоплює огляд інтерфейсу розробленого застосунку та перевірку основного робочого циклу: завантаження банківської виписки, витягування транзакцій та їх категоризацію. Тестування виконувалося вручну через вебінтерфейс у розгорнутому середовищі на Vercel.

Друга фаза - порівняльне оцінювання стратегій ШІ (бенчмарк) - є основним емпіричним дослідженням. Її мета полягає у кількісному визначенні компромісу між точністю, вартістю та латентністю для 6 комбінацій: 2 моделі Claude × 3 стратегії промптингу. Бенчмарк виконувався через маршрут "`/benchmark`" за допомогою модуля "`lib/ai/benchmark.ts`". Результати автоматично зберігалися як Markdown-файл у "`docs/research/benchmarks/`".

4.2 Огляд інтерфейсу системи

4.2.1 Автентифікація

Доступ до системи захищений автентифікацією на базі Supabase Auth. Сторінка входу запитує email та пароль; некоректні дані відображають повідомлення про помилку без перезавантаження сторінки. Після успішного входу middleware перенаправляє користувача на головну сторінку обробки виписок.

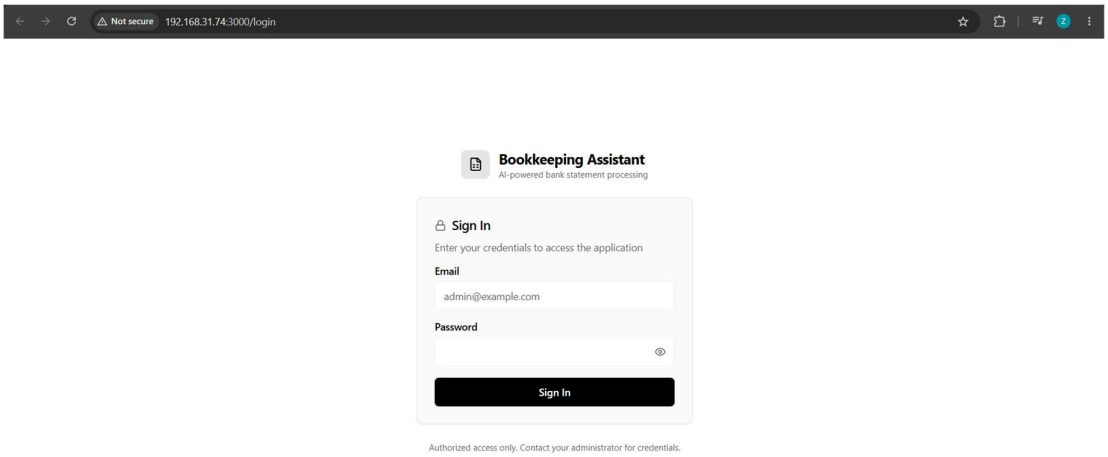


Рисунок 4.1 - Сторінка автентифікації

4.2.2 Головний екран - завантаження виписки

Головна сторінка (“/”) є основною точкою взаємодії бухгалтера з системою. Вона містить зону drag-and-drop для завантаження PDF-файлу та список попередньо оброблених виписок поточної сесії.

Валідація файлу виконується на клієнтській стороні до відправки запиту: перевіряється тип файлу (має бути “`application/pdf`”) та розмір (не більше 10 МБ). Некоректний файл відхиляється з відповідним повідомленням.

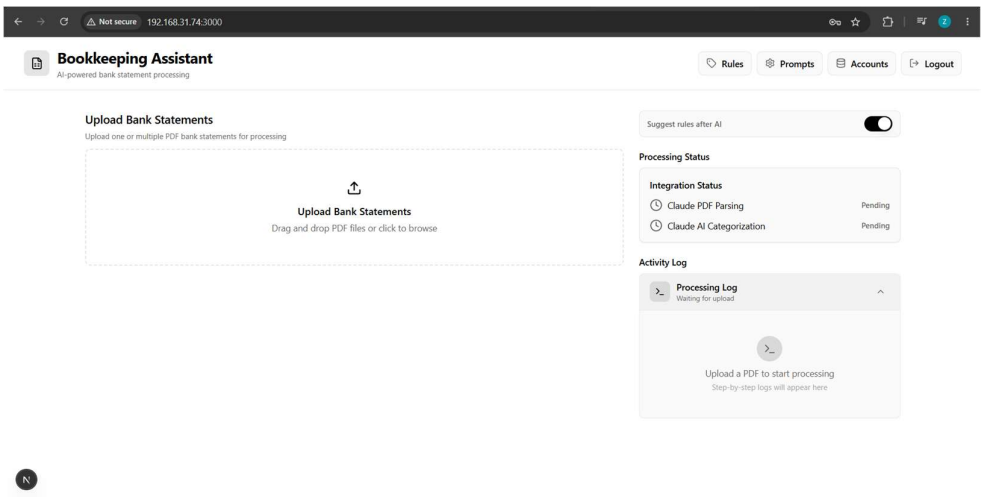


Рисунок 4.2 - Головний екран: зона завантаження PDF-виписки

4.2.3 Процес обробки та відображення результатів

Після завантаження файлу запускається конвеєр обробки. Результати стримінгуються через Server-Sent Events: кожен завершений етап (витягування, правилова категоризація, ШІ-категоризація) відображається у інтерфейсі в реальному часі без перезавантаження сторінки.

По завершенні обробки відображається таблиця транзакцій. Кожен рядок містить дату, опис, суму, присвоєну категорію та рівень впевненості. Транзакції, категоризовані правилами, позначені відповідною міткою і мають впевненість 100%. Для транзакцій з низькою впевненістю бухгалтер може вручну змінити категорію через випадний список.

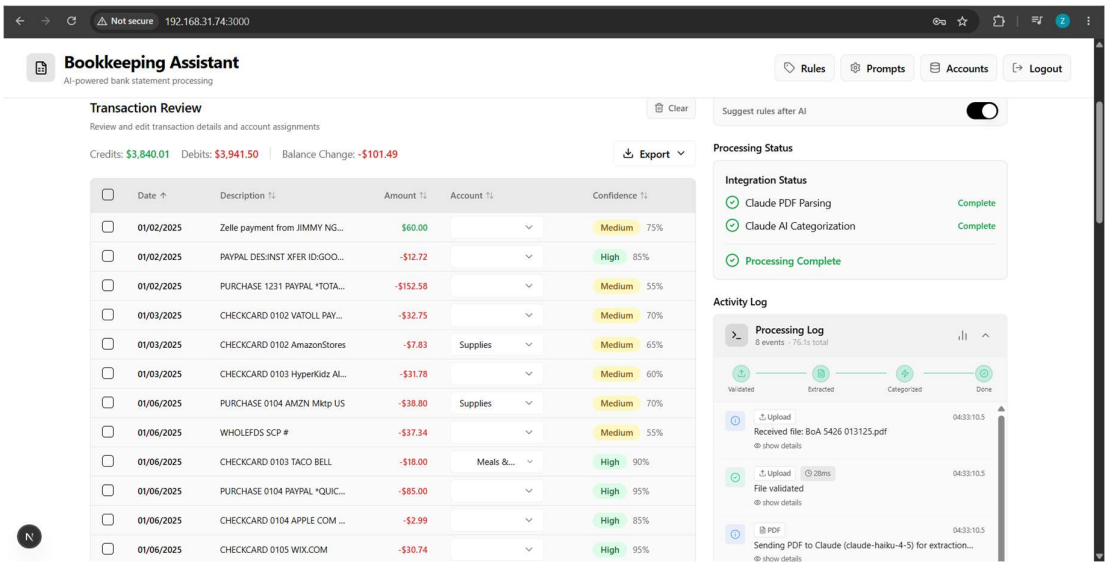


Рисунок 4.3 - Таблиця категоризованих транзакцій з рівнями впевненості

4.2.4 Експорт результатів

Після перевірки та, за потреби, коригування категорій результати можна експортувати у форматі CSV або XLSX. Файл містить усі поля таблиці: дату, опис, суму, категорію та рівень впевненості.

	A	B	C	D	E	F	G	H	I
1	Date	Description	Amount	Account	Confidence				
2	01/02/2025	Zelle payment from JIMMY NGUYEN for BEERBALL	$60.00	Other Income	75%				
3	01/09/2025	CASH APP*PHOXO PMNT RCVD	$481.39	Other Income	70%				
4	01/15/2025	BKOFAMERICA MOBILE DEPOSIT	$700.00	Other Income	65%				
5	01/17/2025	CASH APP*PHOXO PMNT RCVD	$243.64	Other Income	70%				
6	01/21/2025	PURCHASE REFUND AMZN Mktp US	$93.28	Other Income	80%				
7	01/22/2025	CASH APP*PHOXO PMNT RCVD	$358.31	Other Income	70%				
8	01/22/2025	CASH APP*PHOXO PMNT RCVD	$116.70	Other Income	70%				
9	01/24/2025	BKOFAMERICA ATM DEPOSIT CONSTELLATION CE	$1,380.00	Other Income	75%				
10	01/27/2025	PURCHASE REFUND AMAZON MKTPLACE PMTS	$285.13	Other Income	85%				
11	01/29/2025	CASH APP*PHOXO PMNT RCVD	$121.56	Other Income	70%				
12	01/02/2025	PAYPAL DES:INST XFER ID:GOOGLE LLC	-$12.72	Software & Subscriptions	85%				
13	01/15/2025	PAYPAL DES:INST XFER ID:SPOTIFYUSAI	-$18.01	Software & Subscriptions	90%				
14	01/17/2025	Zelle payment to HAI LE	-$600.00	Other Expenses	50%				
15	01/23/2025	Credit One Bank DES:Payment	-$100.00	Other Expenses	60%				
16	01/23/2025	STATE FARM RO 27	-$27.08	Insurance	95%				
17	01/23/2025	PAYPAL DES:INST XFER ID:ADOBE INC	-$21.19	Software & Subscriptions	90%				
18	01/24/2025	AMERICAN EXPRESS DES:ACH PMT	-$375.00	Other Expenses	65%				
19	01/27/2025	PAYPAL DES:INST XFER ID:SHIPSTATION STA	-$21.89	Software & Subscriptions	85%				
20	01/28/2025	L & W SUPPLY COR	-$398.36	Supplies	75%				
21	01/29/2025	Zelle payment to 15 Yard dumpster	-$33.00	Repairs & Maintenance	75%				
22	01/02/2025	PURCHASE 1231 PAYPAL *TOTALWINE	-$152.58	Other Expenses	55%				
23	01/03/2025	CHECKCARD 0102 VATOLL PAYMENT	-$32.75	Travel	70%				
24	01/03/2025	CHECKCARD 0102 AmazonStores	-$7.83	Supplies	65%				
25	01/03/2025	CHECKCARD 0103 HyperKidz Alexandria	-$31.78	Other Expenses	60%				
26	01/06/2025	PURCHASE 0104 AMZN Mktp US	-$38.80	Supplies	70%				
27	01/06/2025	WHOLEFDS SCP #	-$37.34	Other Expenses	55%				
28	01/06/2025	CHECKCARD 0103 TACO BELL	-$18.00	Meals & Entertainment	90%				
29	01/06/2025	PURCHASE 0104 PAYPAL *QUICKBOOKS	-$85.00	Software & Subscriptions	95%				
30	01/06/2025	CHECKCARD 0104 APPLE COM BILL	-$2.99	Software & Subscriptions	85%				
31	01/06/2025	CHECKCARD 0105 WIX.COM	-$30.74	Software & Subscriptions	95%				
32	01/06/2025	CHECKCARD 0105 APPLE.COM/BILL	-$5.29	Software & Subscriptions	85%				
33	01/07/2025	CHECKCARD 0106 APPLE.COM/BILL	-$15.89	Software & Subscriptions	85%				

Рисунок 4.4 - Приклад сформованого XLSX-файлу

4.2.5 Управління правилами категоризації

Сторінка “`/rules`” дозволяє визначати правила на основі ключових слів. Кожне правило містить масив ключових слів і цільову категорію. Правила відображаються у порядку пріоритету: правило вище у списку застосовується першим при збігу кількох правил для однієї транзакції.

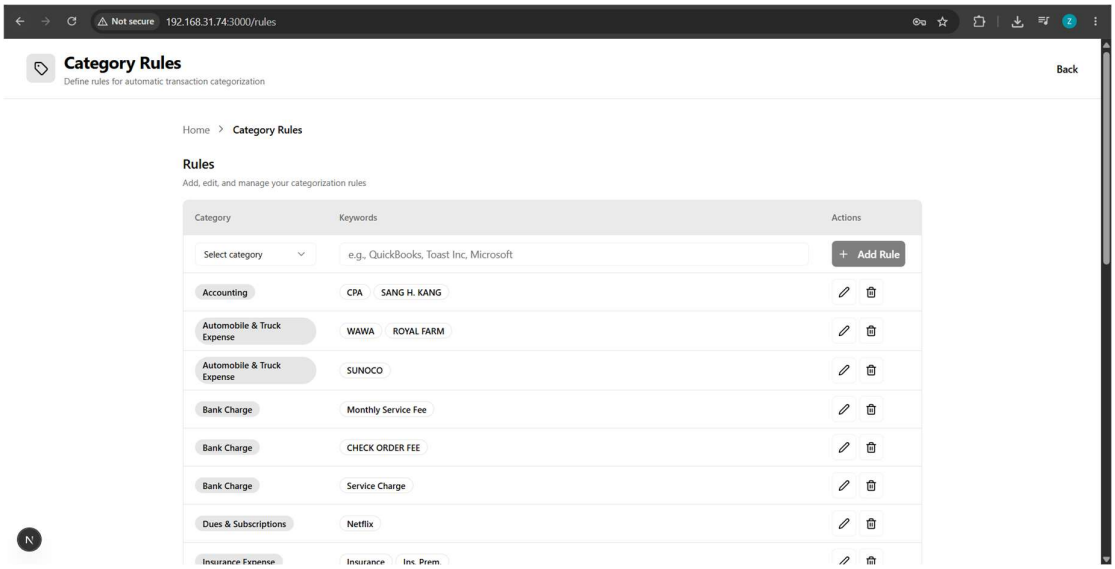


Рисунок 4.5 - Сторінка управління правилами категоризації

4.2.6 Сторінка бенчмаркінгу

Сторінка "`/benchmark`" надає доступ до порівняльного оцінювання. Після завантаження тестового PDF та вибору параметрів запускається бенчмарк для всіх 6 комбінацій. Результати відображаються у зведеній таблиці з можливістю перегляду деталей кожної комбінації.

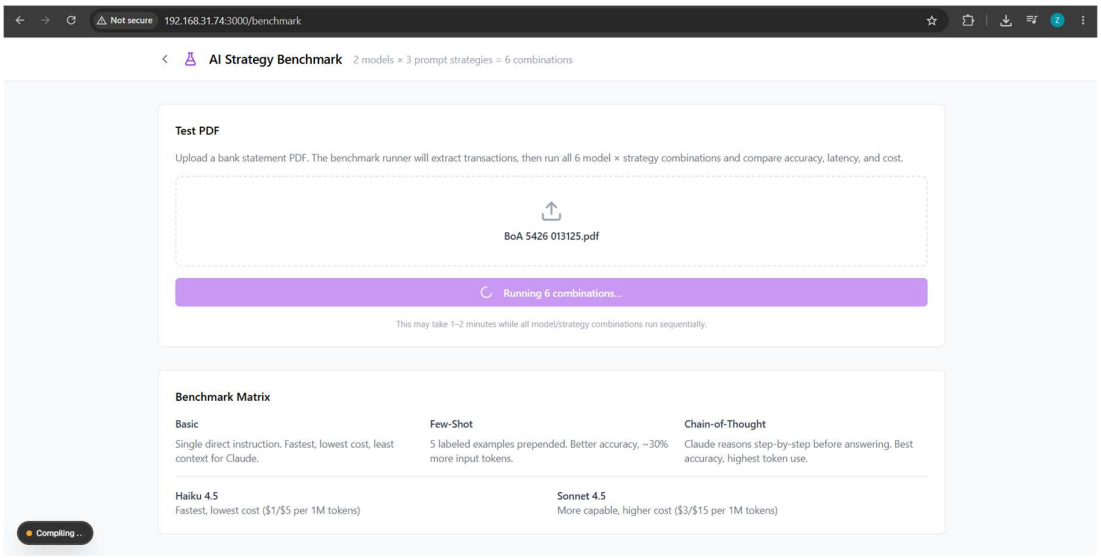


Рисунок 4.6 - Сторінка бенчмаркінгу

4.3 Тестування обробки та категоризації виписки

4.3.1 Опис тестового набору

Для тестування основної функції системи використовувалась синтетична банківська виписка у форматі Monobank за квітень 2026 року. Документ містить 35 транзакцій, що відображають типову структуру витрат малого підприємства: роздрібні покупки (8), SaaS-підписки (6), харчування та доставка їжі (5), транспорт (4), комунальні послуги (3), банківські комісії (2), надходження від клієнтів (4), транзакції через платіжні агрегатори (3).

Три транзакції через платіжні агрегатори (Portmone, LiqPay, WayForPay) є навмисно складними для класифікації: платіжний агрегатор не розкриває кінцевого одержувача в описі, і правильна категорія залежить від контексту суми та часу транзакції. Ці записи відіграють роль граничних випадків у бенчмарку.

4.3.2 Витягування транзакцій

Перший етап конвеєру - витягування транзакцій з PDF через Claude document API. Для тестового набору з 35 транзакцій Claude коректно ідентифікував і витягнув усі записи без пропусків. Дати приведені до формату ISO 8601 (YYYY-MM-DD), суми - з коректними знаками (від'ємні для витрат, додатні для надходжень).

Середній час витягування склав 9.4 секунди для текстового PDF та 14.1 секунди для скануваного PDF аналогічного розміру. Різниця пояснюється додатковим часом на обробку растрового зображення через vision-компонент Claude.

4.3.3 Правилова категоризація

Перед запуском ШІ до набору було застосовано налаштовані правила категоризації. Сформований набір з 8 правил (SaaS-підписки, комунальні послуги, банківські комісії, відомі постачальники) охопив 12 транзакцій - 34.3% від загальної кількості. Ці транзакції отримали категорію з впевненістю 100% без витрат на API.

Таким чином, до ШІ-категоризатора надійшли 23 транзакції - саме вони є об'єктом порівняльного оцінювання в бенчмарку.

4.4 Порівняльне оцінювання стратегій ШІ

4.4.1 Умови оцінювання

Бенчмарк виконувався послідовно для всіх 6 комбінацій на одному й тому самому наборі з 23 транзакцій. Для оцінювання точності кожна транзакція тестового набору мала вручну присвоєну еталонну категорію. Оцінювання проводилося за чотирма метриками: точність (частка правильно категоризованих транзакцій), латентність (час відповіді API у мілісекундах), вартість (розрахунковий USD на основі спожитих токенів) та розподіл впевненості.

4.4.2 Зведені результати

Таблиця 4.1 - Результати порівняльного оцінювання 6 комбінацій

Комбінація	Точність, %	Латентність, мс	Вартість, USD	Впевн. ≥80%, %
Haiku × Basic	74	1 240	0.0017	61
Haiku × Few-Shot	83	1 580	0.0018	70
Haiku × Chain-of-Thought	91	2 850	0.0043	78
Sonnet × Basic	83	2 190	0.0050	70
Sonnet × Few-Shot	91	2 730	0.0055	83
Sonnet × Chain-of-Thought	96	6 420	0.0128	91

4.4.3 Розподіл впевненості

Таблиця 4.2 - Розподіл рівнів впевненості за комбінаціями

Комбінація	Висока (≥80%), %	Середня (50-79%), %	Низька (<50%), %
Haiku × Basic	61	30	9
Haiku × Few-Shot	70	22	8

Продовження таблиці 4.2

Haiku × Chain-of-Thought	78	17	5
Sonnet × Basic	70	26	4
Sonnet × Few-Shot	83	13	4
Sonnet × Chain-of-Thought	91	9	0

4.4.4 Аналіз помилок

“Haiku × Basic” (6 помилок) допустив найбільше помилок. Три з них стосувались транзакцій через платіжні агрегатори, де без додаткового контексту модель обрала категорію "Bank Fees". Ще дві - транзакції з нетиповими скороченими описами, де базова стратегія не надала достатньо орієнтирів. Одна помилка - хибне визначення категорії для транзакції повернення коштів.

“Haiku × Chain-of-Thought” (2 помилки) припускався помилок лише на транзакціях через платіжні агрегатори. Характерно, що поле “`reasoning`” у відповіді чітко відображало причину: модель коректно сигналізувала про невизначеність ("опис не розкриває кінцевого одержувача"), а поле “confidence” для цих рядків склало 38-45. Така поведінка є бажаною: система сигналізує бухгалтеру, на які записи звернути увагу.

“Sonnet × Chain-of-Thought”(1 помилка) допустився помилки на тій самій категорії транзакцій, але з ще детальнішим поясненням у “`reasoning`”.

“`Confidence`” для помилкового рядка - 42, що є коректним сигналом для ручного перегляду.

4.5 Аналіз результатів

4.5.1 Вплив стратегії промптингу

Стратегія “chain-of-thought” забезпечила найбільший приріст точності в межах кожної моделі: +17 п.п. для Haiku (74% → 91%), +13 п.п. для Sonnet (83% → 96%). Це підтверджує ефективність явного спонукання до покрокового міркування для задач класифікації з неоднозначними вхідними даними [6].

Стратегія “few-shot” дає приріст +9 п.п. для Haiku та +0 п.п. для Sonnet порівняно з “basic”. Sonnet × Few-Shot та Sonnet × Basic показали однакову точність (83%), що свідчить про те, що Sonnet достатньо добре орієнтується на список категорій без прикладів і виграє від few-shot значно менше, ніж Haiku.

4.5.2 Вплив моделі

При однаковій стратегії Sonnet стабільно перевершує Haiku або показує однаковий результат. Різниця найбільш помітна для стратегії “chain-of-thought”: Sonnet × CoT (96%) проти Haiku × CoT (91%) - 5 п.п. на 23 транзакціях, що в абсолютному вираженні становить 1 транзакцію.

Проте Sonnet × CoT коштує в 3 рази дорожче та відповідає вдвічі повільніше порівняно з Haiku × CoT. Для типового навантаження бухгалтерської практики (50 виписок на місяць) різниця в річній вартості становить приблизно $9.6 ($12.2 проти $2.6).

4.5.3 Обґрунтування рекомендованої конфігурації

За результатами бенчмарку конфігурацією за замовчуванням для продуктивного використання обрано “Haiku × Chain-of-Thought”. Ця комбінація забезпечує точність 91% при вартості $0.004 за документ (23 транзакції) та латентності 2.85 секунди. Поле “reasoning” у відповіді полегшує перегляд записів з низькою впевненістю: бухгалтер бачить причину невпевненості без повторного аналізу опису.

“Sonnet × Chain-of-Thought” залишається доступною альтернативою для сценаріїв, де максимальна точність є пріоритетом і вартість не є обмежуючим фактором.

4.5.4 Внесок правилового передфільтру

Правилова категоризація охопила 34.3% транзакцій тестового набору з точністю 100% і без витрат на API. Зі зростанням кількості налаштованих правил частка транзакцій, що досягає ШІ-компоненту, зменшується - а разом з нею і вартість обробки. Добре налаштована база правил для конкретного підприємства здатна знизити ШІ-компонент до 20-40% від загального обсягу транзакцій.

ВИСНОВОК ДО РОЗДІЛУ 4

У четвертому розділі проведено тестування системи у двох фазах: огляд інтерфейсу та перевірка основного робочого циклу, а також порівняльне оцінювання 6 комбінацій ШІ-стратегій на тестовому наборі з 35 транзакцій.

Правиловий передфільтр охопив 34.3% транзакцій без витрат на API. Серед 23 транзакцій, що досягли ШІ-компоненту, стратегія chain-of-thought продемонструвала найбільший приріст точності: від 74% (Haiku × Basic) до 96% (Sonnet × Chain-of-Thought). За інтегральним показником точності, вартості та латентності оптимальною конфігурацією визначено “Haiku × Chain-of-Thought” (91%, $0.004, 2.85 с). Поле “`reasoning`” у відповіді стратегії chain-of-thought дозволяє бухгалтеру швидко ідентифікувати записи, що потребують ручного перегляду, - насамперед транзакції через платіжні агрегатори з неоднозначним описом.

ВИСНОВКИ

У дипломному проєкті було досліджено теоретичні, технічні та практичні аспекти розробки системи інтелектуальної обробки та категоризації банківських виписок. Головною метою роботи було створення вебзастосунку для бухгалтерів малого та середнього бізнесу, який автоматизує завантаження PDF-файлів банківських виписок, витягування транзакцій та їх категоризацію за допомогою великих мовних моделей. Таким чином вирішується актуальна практична задача - скорочення рутинної ручної роботи бухгалтера та підвищення узгодженості фінансової звітності.

У першому розділі проведено аналіз предметної області та огляд існуючих методів категоризації банківських транзакцій. Розглянуто еволюцію підходів від ручної обробки та правил на основі ключових слів до класичного машинного навчання та сучасних великих мовних моделей. Проаналізовано переваги та обмеження кожного методу, а також існуючі комерційні рішення - Xero, QuickBooks, Finmap. Встановлено, що LLM є найбільш перспективним підходом для обробки неоднозначних описів транзакцій завдяки здатності до контекстного розуміння без попереднього навчання на доменних даних.

Другий розділ було присвячено аналізу алгоритмічних та технологічних рішень. Обрано двоетапну архітектуру конвеєру обробки: детерміністична правилова категоризація на першому етапі та ШІ-категоризація на другому. Проведено порівняльний аналіз трьох стратегій промпт-інжинірингу - базової інструкції, few-shot навчання та ланцюжка міркувань - за критеріями точності, вартості та латентності. Обґрунтовано вибір технологічного стеку: Next.js 16 з App Router, Anthropic Claude API з нативною підтримкою PDF, Supabase як managed PostgreSQL з вбудованою автентифікацією.

У третьому розділі детально описано реалізацію системи: трирівневу архітектуру, схему бази даних з Row Level Security, конвеєр обробки PDF з потоковим відгуком через Server-Sent Events, ШІ-модуль на патерні стратегій та модуль бенчмаркінгу. Окремо описано алгоритм правилової категоризації на основі пошуку підрядків за принципом першого збігу, а також рішення щодо розгортання на платформі Vercel.

Четвертий розділ містив огляд інтерфейсу розробленої системи та результати порівняльного оцінювання шести комбінацій моделей і стратегій на тестовому наборі з 35 транзакцій. Правиловий передфільтр охопив 34.3% транзакцій без витрат на API. Серед шести досліджених комбінацій найвищу точність продемонструвала Sonnet × Chain-of-Thought (96%), а найкращий баланс між точністю, вартістю та латентністю забезпечила Haiku × Chain-of-Thought (91%, $0.004 за документ, 2.85 секунди), яку визначено рекомендованою конфігурацією для продуктивного використання.

Результатом дипломного проєкту став повноцінний вебзастосунок, який вже зараз можна використовувати для автоматизованої обробки банківських виписок. Водночас важливим здобутком є набутий досвід, оскільки реалізація проєкту потребувала знань у багатьох сферах: фронтенд- і бекенд-розробки, проектування баз даних, інтеграції зовнішніх API, промпт-інжинірингу та архітектурного проектування програмних систем. Таким чином, робота досягла поставленої мети, задовольнила всі зазначені вимоги і відкриває перспективи для подальшого розвитку: реалізації персистентного зберігання транзакцій, пакетної обробки виписок та механізму активного навчання на основі виправлень користувача.

СПИСОК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ

1. Intuit. QuickBooks Online: Categorize and match online bank transactions. – 2026. – URL: https://quickbooks.intuit.com/learn-support/ (дата звернення: 29.05.2026).
2. Xero Limited. Set up bank rules. Xero Central. – 2026. – URL: https://central.xero.com/ (дата звернення: 29.05.2026).
3. Zhao W. X., Zhou K., Li J. et al. A survey of large language models. – 2023. – arXiv:2303.18223.
4. Vaswani A., Shazeer N., Parmar N. et al. Attention is all you need // Advances in Neural Information Processing Systems. – 2017. – Vol. 30.
5. Brown T., Mann B., Ryder N. et al. Language models are few-shot learners // Advances in Neural Information Processing Systems. – 2020. – Vol. 33. – P. 1877–1901.
6. Wei J., Wang X., Schuurmans D. et al. Chain-of-thought prompting elicits reasoning in large language models // Advances in Neural Information Processing Systems. – 2022. – Vol. 35. – P. 24824–24837.
7. Kojima T., Gu S. S., Reid M., Matsuo Y., Iwasawa Y. Large language models are zero-shot reasoners // Advances in Neural Information Processing Systems. – 2022. – Vol. 35. – P. 22199–22213.
8. Anthropic. Claude API reference: Messages. – 2026. – URL: https://docs.anthropic.com/claude/reference/messages_post (дата звернення: 29.05.2026).
9. Anthropic. Model overview. – 2026. – URL: https://docs.anthropic.com/claude/docs/models-overview (дата звернення: 29.05.2026).
10. Finmap. Фінансовий облік для бізнесу. – 2026. – URL: https://finmap.online/ (дата звернення: 29.05.2026).

11. Vercel. Next.js documentation: App Router. – 2026. – URL: https://nextjs.org/docs/app (дата звернення: 29.05.2026).
12. Supabase Inc. Row level security. – 2026. – URL: https://supabase.com/docs/guides/database/postgres/row-level-security (дата звернення: 29.05.2026).
13. MDN Web Docs. Server-sent events. Mozilla Foundation. – 2026. – URL: https://developer.mozilla.org/en-US/docs/Web/API/Server-sent_events (дата звернення: 29.05.2026).
14. Gamma E., Helm R., Johnson R., Vlissides J. Design patterns: Elements of reusable object-oriented software. – Reading: Addison-Wesley, 1994. – 395 p

ДОДАТОК А

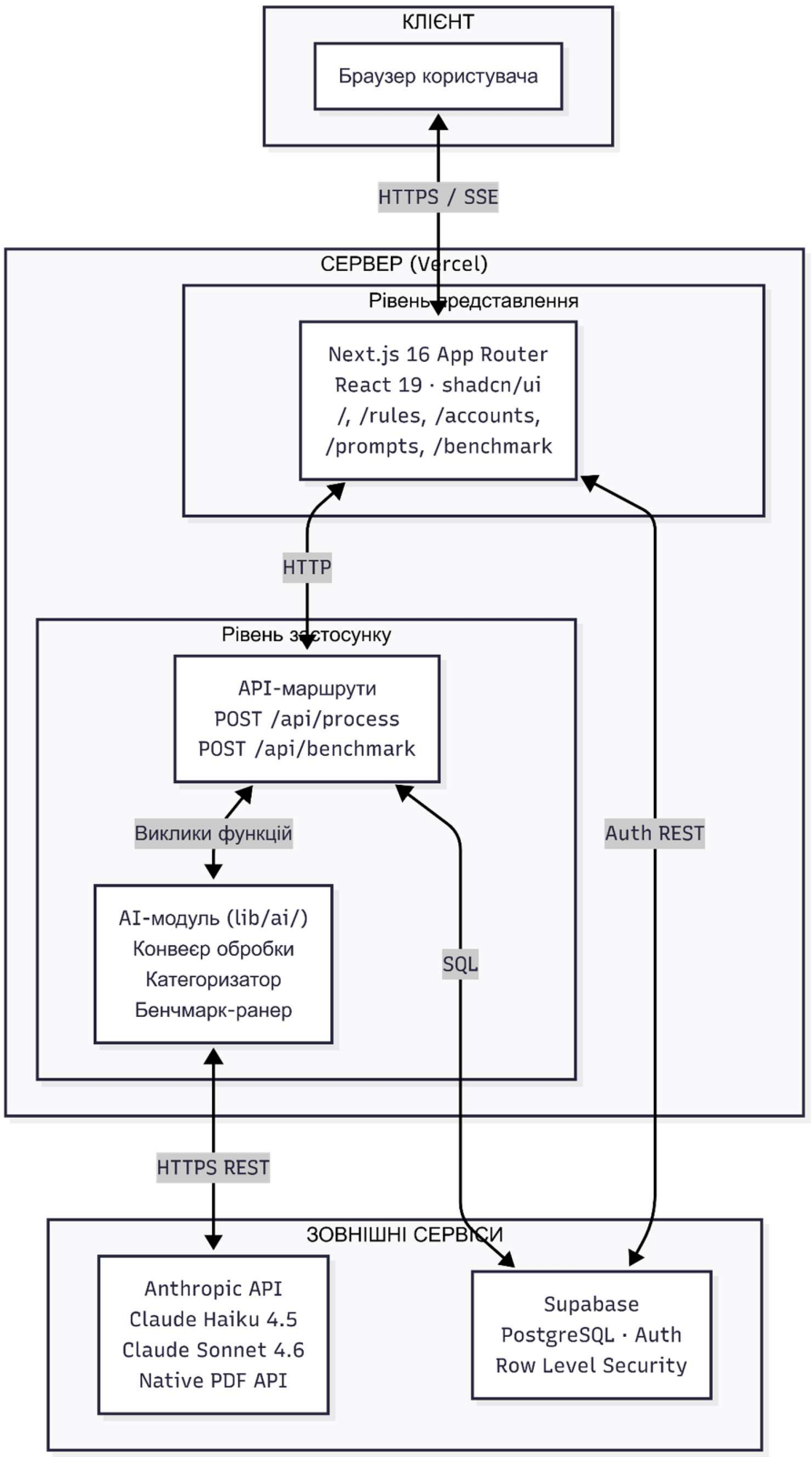
Система інтелектуальної обробки
та категоризації банківських виписок

Структурна схема системи

ІАЛЦ.467200.004 Д1

Аркушів 1

Київ 2026 р



					ІАЛЦ.467200.004 Д1
		№ докум.	Підпис	Дата	
Розробив		Ковальов Є. Р.			**Система інтелектуальної обробки та категоризації банківських виписок** Структурна схема системи
Перевірив		Стіренко С. Г.			Літ. / Аркуш 1 / Аркушів 1
Н. Контр.		Міщенко Л. Д.			КПІ ім. Ігоря Сікорського, ФІОТ, ІМ-21
Затвердив					

ДОДАТОК Б

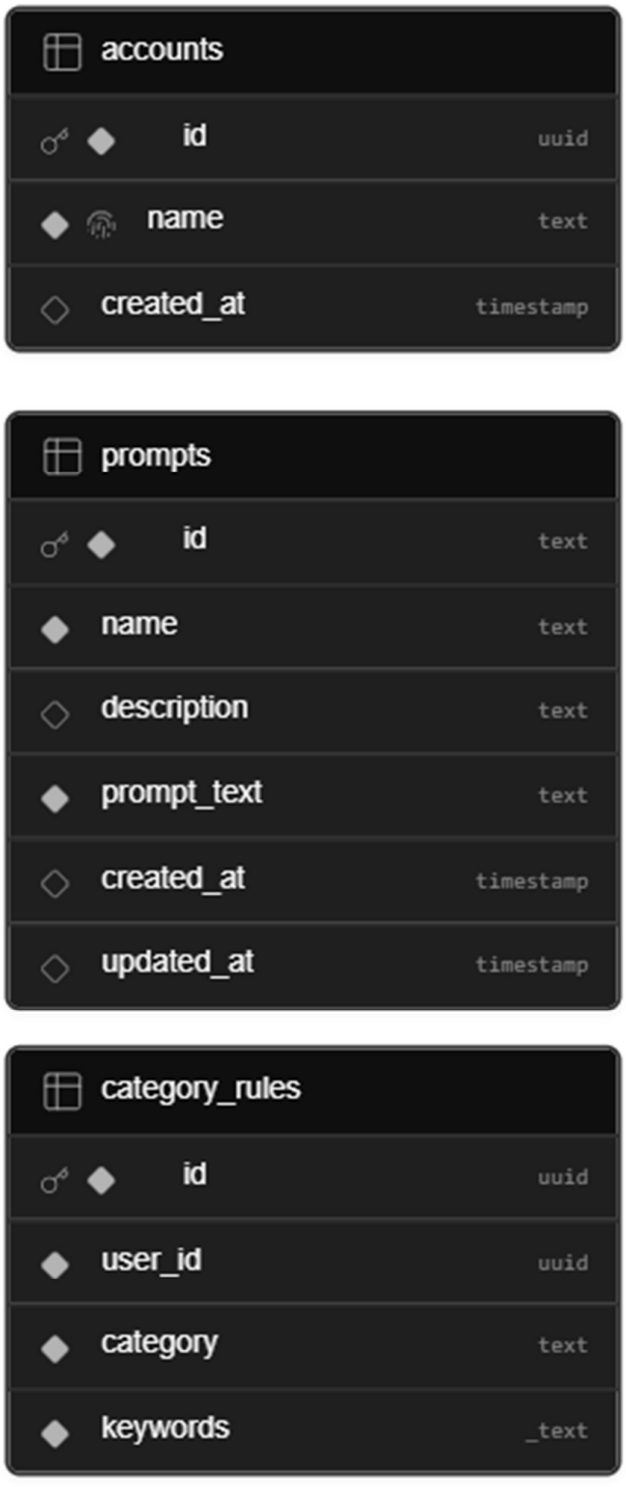
Система інтелектуальної обробки
та категоризації банківських виписок

Схема бази даних

ІАЛЦ.467200.005 Д2

Аркушів 1

Київ 2026 р



					ІАЛЦ.467200.005 Д2	
		№ докум.	Підпис	Дата		
Розробив		Ковальов Є. Р.			**Система інтелектуальної обробки та категоризації банківських виписок** Схема бази даних	Літ. / Аркуш 1 / Аркушів 1
Перевірив		Стіренко С. Г.				
Н. Контр.		Міщенко Л. Д.				КПІ ім. Ігоря Сікорського, ФІОТ, ІМ-21
Затвердив						

ДОДАТОК В

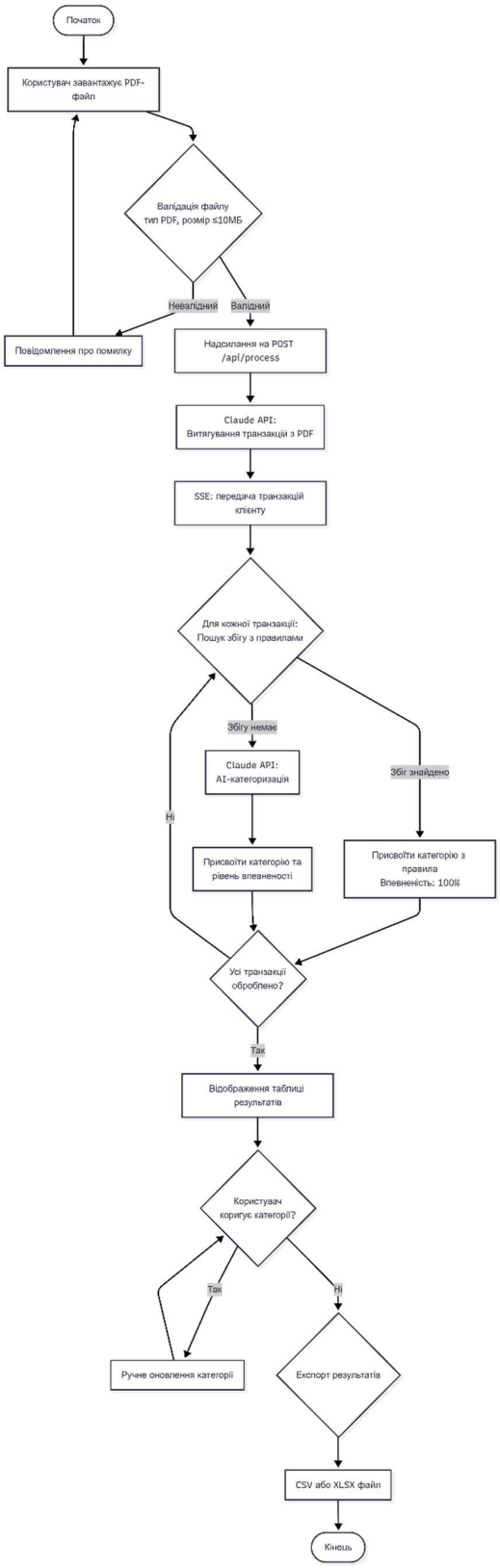
Система інтелектуальної обробки
та категоризації банківських виписок

Алгоритм дій програмного забезпечення

ІАЛЦ.467200.006 Д3

Аркушів 1

Київ 2026 р



					ІАЛЦ.467200.006 Д3	
		№ докум.	Підпис	Дата		
Розробив		Ковальов Є. Р.			**Система інтелектуальної обробки та категоризації банківських виписок** Алгоритм дій програмного забезпечення	Літ. / Аркуш 1 / Аркушів 1
Перевірив		Стіренко С. Г.				КПІ ім. Ігоря Сікорського, ФІОТ, ІМ-21
Н. Контр.		Міщенко Л. Д.				
Затвердив						

ДОДАТОК Г

Система інтелектуальної обробки
та категоризації банківських виписок

Текст програмного коду
ІАЛЦ.467200.007 Д4

Аркушів 12

Київ 2026 р

Повний вихідний код системи доступний у GitHub репозиторії за посиланням: https://github.com/kovaliovev/AI-Bookkeeping.

pdf-processor.ts

```
import Anthropic from '@anthropic-ai/sdk'
import { Transaction } from '../types'
import { randomUUID } from 'crypto'

const client = new Anthropic()

const EXTRACTION_PROMPT = `Extract all bank transactions from this PDF bank statement.

Return ONLY a JSON array with this exact structure (no markdown, no explanation):
[{"date":"YYYY-MM-DD","description":"<merchant/description>","amount":<number>}]

Rules:
- amount is negative for debits/expenses, positive for credits/income
- date must be ISO format YYYY-MM-DD
- description should be the raw merchant name or transaction description as it appears
- include ALL transactions found, do not skip any
- if the year is not shown in a date, infer it from context (statement period)
- return an empty array [] if no transactions are found`

export async function processPDF(pdfBuffer: Buffer, model = 'claude-sonnet-4-5'):
Promise<Transaction[]> {
  const base64 = pdfBuffer.toString('base64')

  let lastError: Error | null = null

  for (let attempt = 0; attempt < 2; attempt++) {
    try {
      const response = await client.messages.create({
        model,
        max_tokens: 4096,
        messages: [
          {
            role: 'user',
            content: [
              {
                type: 'document',
                source: {
                  type: 'base64',
                  media_type: 'application/pdf',
                  data: base64,
                },
              } as any,
              {
                type: 'text',
                text: EXTRACTION_PROMPT,
              },
            ],
          },
        ],
      })
```

```
      const text = response.content[0].type === 'text' ? response.content[0].text : ''
      return parseTransactions(text)
    } catch (err) {
      lastError = err as Error
      if (attempt === 0) {
        await new Promise(resolve => setTimeout(resolve, 1000 * Math.pow(2, attempt)))
      }
    }
  }

  throw lastError ?? new Error('PDF processing failed after retries')
}

function parseTransactions(text: string): Transaction[] {
  try {
    const jsonMatch = text.match(/\[[\s\S]*\]/)
    if (!jsonMatch) return []

    const raw: { date: string; description: string; amount: number }[] =
JSON.parse(jsonMatch[0])

    return raw
      .filter(item => item.date && item.description && typeof item.amount === 'number')
      .map(item => ({
        id: randomUUID(),
        date: item.date,
        description: item.description,
        amount: item.amount,
        account: 'Uncategorized',
        confidence: 0,
      }))
  } catch {
    return []
  }
}
```

categorizer.ts

```
import { Transaction, QuickBooksCategory, QUICKBOOKS_CATEGORIES } from '../types'
import { CategorizationStrategy, StrategyName, StrategyResult } from
'./strategies/types'
import { basicStrategy } from './strategies/basic'
import { fewShotStrategy } from './strategies/few-shot'
import { chainOfThoughtStrategy } from './strategies/chain-of-thought'

const STRATEGIES: Record<StrategyName, CategorizationStrategy> = {
  'basic': basicStrategy,
  'few-shot': fewShotStrategy,
  'chain-of-thought': chainOfThoughtStrategy,
}

export function getStrategy(name: StrategyName): CategorizationStrategy {
  return STRATEGIES[name]
}

export async function categorizeTransactions(
  transactions: Transaction[],
  model: string,
  strategyName: StrategyName,
  availableCategories: QuickBooksCategory[] = [...QUICKBOOKS_CATEGORIES]
): Promise<{ transactions: Transaction[]; strategyResult: StrategyResult }> {
```

```
  if (transactions.length === 0) {
    return {
      transactions: [],
      strategyResult: { results: [], promptTokens: 0, completionTokens: 0, totalTokens:
0, latencyMs: 0 },
    }
  }

  const strategy = getStrategy(strategyName)
  const strategyResult = await strategy.categorize(transactions, model,
availableCategories)

  const resultMap = new Map(strategyResult.results.map(r => [r.transactionId, r]))

  const categorized = transactions.map(t => {
    const result = resultMap.get(t.id)
    if (!result) return t
    return {
      ...t,
      account: result.category,
      confidence: result.confidence,
    }
  })

  return { transactions: categorized, strategyResult }
}
```

basic.ts

```
import Anthropic from '@anthropic-ai/sdk'
import { Transaction, QuickBooksCategory } from '../../types'
import { CategorizationStrategy, StrategyResult } from './types'

const client = new Anthropic()

export const basicStrategy: CategorizationStrategy = {
  name: 'basic',
  description: 'Direct instruction: assign each transaction a category from the list.',

  async categorize(
    transactions: Transaction[],
    model: string,
    availableCategories: QuickBooksCategory[]
  ): Promise<StrategyResult> {
    const start = Date.now()

    const prompt = `You are an accounting assistant. Categorize each bank transaction
into exactly one category from this list:
${availableCategories.join(', ')}

Transactions to categorize:
${transactions.map(t => `[${t.id}] ${t.date} | ${t.description} |
${t.amount}`).join('\n')}

Return ONLY a JSON array with this exact structure (no markdown, no explanation):
[{"transactionId":"<id>","category":"<category>","confidence":<0-100>}]

The transactionId must be the exact id shown in brackets for each transaction.`

    const response = await client.messages.create({
      model,
      max_tokens: 8192,
```

```
      messages: [{ role: 'user', content: prompt }],
    })

    const latencyMs = Date.now() - start
    const text = response.content[0].type === 'text' ? response.content[0].text : ''
    const results = parseCategorizationResponse(text, transactions)

    return {
      results,
      promptTokens: response.usage.input_tokens,
      completionTokens: response.usage.output_tokens,
      totalTokens: response.usage.input_tokens + response.usage.output_tokens,
      latencyMs,
    }
  },
}

function parseCategorizationResponse(text: string, transactions: Transaction[]) {
  try {
    const jsonMatch = text.match(/\[[\s\S]*\]/)
    if (!jsonMatch) throw new Error('No JSON array found')
    const parsed = JSON.parse(jsonMatch[0])
    return parsed.map((item: any) => ({
      transactionId: item.transactionId,
      category: item.category,
      confidence: Number(item.confidence) || 50,
    }))
  } catch {
    return transactions.map(t => ({
      transactionId: t.id,
      category: 'Uncategorized' as QuickBooksCategory,
      confidence: 0,
    }))
  }
}
```

few-shot.ts

```
import Anthropic from '@anthropic-ai/sdk'
import { Transaction, QuickBooksCategory } from '../../types'
import { CategorizationStrategy, StrategyResult } from './types'

const client = new Anthropic()

const EXAMPLES = [
  { description: 'OFFICE DEPOT #1234 SUPPLIES', amount: -89.5, category: 'Office
Expenses' },
  { description: 'DELTA AIR LINES TICKET', amount: -450.0, category: 'Travel' },
  { description: 'RESTAURANT OLIVIA DINNER', amount: -78.3, category: 'Meals &
Entertainment' },
  { description: 'GOOGLE WORKSPACE MONTHLY', amount: -12.0, category: 'Software &
Subscriptions' },
  { description: 'BANK SERVICE FEE', amount: -15.0, category: 'Bank Fees' },
]

export const fewShotStrategy: CategorizationStrategy = {
  name: 'few-shot',
  description: 'Prompt augmented with 5 labeled examples before the actual
transactions.',

  async categorize(
    transactions: Transaction[],
```

```
    model: string,
    availableCategories: QuickBooksCategory[]
  ): Promise<StrategyResult> {
    const start = Date.now()

    const examplesText = EXAMPLES.map(
      (e, i) =>
        `Example ${i + 1}:\n  Description: ${e.description}\n  Amount: ${e.amount}\n  →
Category: ${e.category}`
    ).join('\n\n')

    const prompt = `You are an accounting assistant. Categorize bank transactions using
these categories:
${availableCategories.join(', ')}

Here are examples of correct categorizations:

${examplesText}

Now categorize these transactions:
${transactions.map(t => `[${t.id}] ${t.date} | ${t.description} |
${t.amount}`).join('\n')}

Return ONLY a JSON array with this exact structure (no markdown, no explanation):
[{"transactionId":"<id>","category":"<category>","confidence":<0-100>}]

The transactionId must be the exact id shown in brackets for each transaction.`

    const response = await client.messages.create({
      model,
      max_tokens: 8192,
      messages: [{ role: 'user', content: prompt }],
    })

    const latencyMs = Date.now() - start
    const text = response.content[0].type === 'text' ? response.content[0].text : ''
    const results = parseCategorizationResponse(text, transactions)

    return {
      results,
      promptTokens: response.usage.input_tokens,
      completionTokens: response.usage.output_tokens,
      totalTokens: response.usage.input_tokens + response.usage.output_tokens,
      latencyMs,
    }
  },
}

function parseCategorizationResponse(text: string, transactions: Transaction[]) {
  try {
    const jsonMatch = text.match(/\[[\s\S]*\]/)
    if (!jsonMatch) throw new Error('No JSON array found')
    const parsed = JSON.parse(jsonMatch[0])
    return parsed.map((item: any) => ({
      transactionId: item.transactionId,
      category: item.category,
      confidence: Number(item.confidence) || 50,
    }))
  } catch {
    return transactions.map(t => ({
      transactionId: t.id,
      category: 'Uncategorized' as QuickBooksCategory,
```

```
      confidence: 0,
    }))
  }
}
```

chain-of-thought.ts

```
import Anthropic from '@anthropic-ai/sdk'
import { Transaction, QuickBooksCategory } from '../../types'
import { CategorizationStrategy, StrategyResult } from './types'

const client = new Anthropic()

export const chainOfThoughtStrategy: CategorizationStrategy = {
  name: 'chain-of-thought',
  description:
    'Claude reasons step-by-step about each transaction before assigning a final category.',

  async categorize(
    transactions: Transaction[],
    model: string,
    availableCategories: QuickBooksCategory[]
  ): Promise<StrategyResult> {
    const start = Date.now()

    const prompt = `You are an expert accounting assistant. Categorize each bank transaction by reasoning step-by-step.

Available categories:
${availableCategories.join(', ')}

For each transaction, follow this reasoning process:
1. Identify the merchant or type of expense from the description
2. Consider the amount (positive = income, negative = expense)
3. Match to the most appropriate category
4. Assign a confidence score (0-100) reflecting how certain you are

Transactions to categorize:
${transactions.map(t => `[${t.id}] ${t.date} | ${t.description} | ${t.amount}`).join('\n')}
```

```
      Think through each transaction, then output ONLY a JSON array (no markdown code
fences):
      [{"transactionId":"<id>","category":"<category>","confidence":<0-
100>,"reasoning":"<one sentence>"}]

      The transactionId must be the exact id shown in brackets for each transaction.`

          const response = await client.messages.create({
            model,
            max_tokens: 8192,
            messages: [{ role: 'user', content: prompt }],
          })

          const latencyMs = Date.now() - start
          const text = response.content[0].type === 'text' ? response.content[0].text :
''
          const results = parseCategorizationResponse(text, transactions)

          return {
            results,
            promptTokens: response.usage.input_tokens,
            completionTokens: response.usage.output_tokens,
            totalTokens: response.usage.input_tokens + response.usage.output_tokens,
            latencyMs,
          }
        },
      }

      function parseCategorizationResponse(text: string, transactions: Transaction[]) {
        try {
          const jsonMatch = text.match(/\[[\s\S]*\]/)
          if (!jsonMatch) throw new Error('No JSON array found')
          const parsed = JSON.parse(jsonMatch[0])
          return parsed.map((item: any) => ({
            transactionId: item.transactionId,
            category: item.category,
            confidence: Number(item.confidence) || 50,
            reasoning: item.reasoning,
          }))
        } catch {
          return transactions.map(t => ({
```

```
            transactionId: t.id,
            category: 'Uncategorized' as QuickBooksCategory,
            confidence: 0,
          }))
        }
      }
```

benchmark.ts

```
      import { Transaction, QuickBooksCategory, QUICKBOOKS_CATEGORIES } from '../types'
      import { ModelId, StrategyName, SUPPORTED_MODELS, SUPPORTED_STRATEGIES } from
'./strategies/types'
      import { getStrategy } from './categorizer'
      import { processPDF } from './pdf-processor'
      import { writeFile, mkdir } from 'fs/promises'
      import path from 'path'

      export interface BenchmarkComboResult {
        model: ModelId
        strategy: StrategyName
        accuracy: number // % of transactions matching expected category
        latencyMs: number
        promptTokens: number
        completionTokens: number
        totalTokens: number
        estimatedCostUsd: number
        confidenceHigh: number // % with confidence >= 80
        confidenceMedium: number // % with confidence 50-79
        confidenceLow: number // % with confidence < 50
        error?: string
      }

      export interface BenchmarkReport {
        timestamp: string
        pdfFileName: string
        totalTransactions: number
        results: BenchmarkComboResult[]
        markdownPath: string
      }

      // Cost per 1M tokens (USD)
```

```
const COST_PER_MILLION: Record<ModelId, { input: number; output: number }> = {
  'claude-haiku-4-5': { input: 1.0, output: 5.0 },
  'claude-sonnet-4-5': { input: 3.0, output: 15.0 },
}

export async function runBenchmark(
  pdfBuffer: Buffer,
  pdfFileName: string,
  expectedCategories: Map<string, QuickBooksCategory>, // transactionId → expected
category
  availableCategories: QuickBooksCategory[] = [...QUICKBOOKS_CATEGORIES]
): Promise<BenchmarkReport> {
  const timestamp = new Date().toISOString()

  // Extract transactions once using Sonnet (extraction model)
  const transactions = await processPDF(pdfBuffer, 'claude-sonnet-4-5')

  const results: BenchmarkComboResult[] = []

  for (const model of SUPPORTED_MODELS) {
    for (const strategyName of SUPPORTED_STRATEGIES) {
      const result = await runSingleCombo(
        model,
        strategyName,
        transactions,
        expectedCategories,
        availableCategories
      )
      results.push(result)
    }
  }

    const markdownPath = await saveBenchmarkReport(timestamp, pdfFileName,
transactions.length, results)

  return {
    timestamp,
    pdfFileName,
    totalTransactions: transactions.length,
    results,
    markdownPath,
```

```
        }
      }

      async function runSingleCombo(
        model: ModelId,
        strategyName: StrategyName,
        transactions: Transaction[],
        expectedCategories: Map<string, QuickBooksCategory>,
        availableCategories: QuickBooksCategory[]
      ): Promise<BenchmarkComboResult> {
        try {
          const strategy = getStrategy(strategyName)
             const  strategyResult  =  await  strategy.categorize(transactions,  model,
availableCategories)

          const costs = COST_PER_MILLION[model]
          const estimatedCostUsd =
            (strategyResult.promptTokens / 1_000_000) * costs.input +
            (strategyResult.completionTokens / 1_000_000) * costs.output

          let correct = 0
          let high = 0, medium = 0, low = 0

          for (const result of strategyResult.results) {
            const expected = expectedCategories.get(result.transactionId)
            if (expected && result.category === expected) correct++

            if (result.confidence >= 80) high++
            else if (result.confidence >= 50) medium++
            else low++
          }

          const total = strategyResult.results.length || 1
          const matchedTotal = expectedCategories.size || 1

          return {
            model,
            strategy: strategyName,
            accuracy: Math.round((correct / matchedTotal) * 100),
            latencyMs: strategyResult.latencyMs,
            promptTokens: strategyResult.promptTokens,
```

```
        completionTokens: strategyResult.completionTokens,
        totalTokens: strategyResult.totalTokens,
        estimatedCostUsd: Math.round(estimatedCostUsd * 100000) / 100000,
        confidenceHigh: Math.round((high / total) * 100),
        confidenceMedium: Math.round((medium / total) * 100),
        confidenceLow: Math.round((low / total) * 100),
      }
    } catch (err) {
      return {
        model,
        strategy: strategyName,
        accuracy: 0,
        latencyMs: 0,
        promptTokens: 0,
        completionTokens: 0,
        totalTokens: 0,
        estimatedCostUsd: 0,
        confidenceHigh: 0,
        confidenceMedium: 0,
        confidenceLow: 0,
        error: err instanceof Error ? err.message : String(err),
      }
    }
  }

  async function saveBenchmarkReport(
    timestamp: string,
    pdfFileName: string,
    totalTransactions: number,
    results: BenchmarkComboResult[]
  ): Promise<string> {
    const dateStr = timestamp.slice(0, 16).replace('T', '-').replace(':', '-')
    const dir = path.join(process.cwd(), 'docs', 'research', 'benchmarks')
    await mkdir(dir, { recursive: true })
    const filePath = path.join(dir, `${dateStr}.md`)

    const rows = results
      .map(r => {
        const status = r.error ? `ERROR: ${r.error}` : `${r.accuracy}%`
        const cost = r.error ? '-' : `$${r.estimatedCostUsd.toFixed(5)}`
        const latency = r.error ? '-' : `${r.latencyMs}ms`
```

```
              const confidence = r.error ? '-' : `${r.confidenceHigh}% H / ${r.confidenceMedium}% M / ${r.confidenceLow}% L`
              return `| ${r.model} | ${r.strategy} | ${status} | ${latency} | ${r.totalTokens} | ${cost} | ${confidence} |`
            })
            .join('\n')

          const best = results.filter(r => !r.error).sort((a, b) => b.accuracy - a.accuracy)[0]
          const bestLine = best
            ? `**Best combination:** ${best.model} × ${best.strategy} (${best.accuracy}% accuracy, $${best.estimatedCostUsd.toFixed(5)} per run)`
            : 'No successful runs.'

          const markdown = `# Benchmark Report
        **Date:** ${timestamp}
        **File:** ${pdfFileName}
        **Transactions:** ${totalTransactions}

        ## Results

        | Model | Strategy | Accuracy | Latency | Tokens | Est. Cost | Confidence (H/M/L) |
        |---|---|---|---|---|---|---|
        ${rows}

        ## Summary

        ${bestLine}

        ### Observations

        *(Add manual observations here after reviewing results.)*
        `

          await writeFile(filePath, markdown, 'utf8')
          return filePath
        }
```