

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ДЕРЖАВНИЙ УНІВЕРСИТЕТ «ЖИТОМИРСЬКА ПОЛІТЕХНІКА»
ФАКУЛЬТЕТ ІНФОРМАЦІЙНО-КОМП'ЮТЕРНИХ ТЕХНОЛОГІЙ
КАФЕДРА ІНЖЕНЕРІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

ПОЯСНЮВАЛЬНА ЗАПИСКА

до кваліфікаційної роботи освітнього ступеня «бакалавр»
за спеціальністю 121 «Інженерія програмного забезпечення»
(освітня програма «Інженерія програмного забезпечення»)

на тему:

**«Архітектура інтелектуального фінансового асистента
на основі ШІ-агента»**

Виконав студент групи ЗПЗ-22-1
МАЙЗЕНБЕРГ Петро Дмитрійович

Керівник роботи:
ГОДЛЕВСЬКИЙ Юрій Олександрович

Рецензент:
ДУНЄВ Сергій Сергійович

Житомир – 2026

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ДЕРЖАВНИЙ УНІВЕРСИТЕТ «ЖИТОМИРСЬКА ПОЛІТЕХНІКА»
ФАКУЛЬТЕТ ІНФОРМАЦІЙНО-КОМП'ЮТЕРНИХ ТЕХНОЛОГІЙ
КАФЕДРА ІНЖЕНЕРІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

«ЗАТВЕРДЖУЮ»

Зав. кафедри інженерії
програмного забезпечення
Тетяна Вакалюк
«28» лютого 2026 р.

ЗАВДАННЯ
на кваліфікаційну роботу

Здобувач вищої освіти: **МАЙЗЕНБЕРГ Петро Дмитрійович**

Керівник роботи: **ГОДЛЕВСЬКИЙ Юрій Олександрович**

Тема роботи: **«Архітектура інтелектуального фінансового асистента на основі ШІ-агента»,**

затверджена Наказом закладу вищої освіти від **«27» лютого 2026 р., №93/с**

Вихідні дані для роботи: **об'єктом дослідження є інформаційні технології для розробки веб-застосунку обліку особистих фінансів, а предметом дослідження є засоби та методи, що забезпечують облік доходів і витрат, аналіз фінансових даних та взаємодію користувача з ШІ-асистентом**

Консультанти з бакалаврської кваліфікаційної роботи із зазначенням розділів, що їх стосуються:

Розділ	Консультант	Завдання видав	Завдання прийняв
1	Годлевський Ю.О.	13.03.2026р.	18.03.2026р
2	Годлевський Ю.О.	24.05.2026р	27.05.2026р
3	Годлевський Ю.О.	07.06.2026р	10.06.2026р

РЕФЕРАТ

Кваліфікаційна робота бакалавра складається з вебзастосунку для управління особистими фінансами «FinAgent» та пояснювальної записки. Пояснювальна записка містить 77 сторінок, 20 рисунків, 21 таблицю, 3 додатки і 30 використаних джерел.

Метою роботи є розробка архітектури фінансового асистента на основі ШІ-агента для обліку та аналізу особистих фінансів користувача.

У розділах пояснювальної записки проаналізовано аналогічні програмні продукти, розглянуто методи та засоби реалізації проєкту й обрано технології для розробки системи. Обґрунтовано розподілену клієнт-серверну архітектуру, спроектовано базу даних та схему варіантів використання. Проведено тестування та продемонстровано функціонал застосунку.

КЛЮЧОВІ СЛОВА: ШІ-АСИСТЕНТ, ВЕБЗАСТОСУНОК, МІКРОСЕРВІСНА АРХІТЕКТУРА, УПРАВЛІННЯ ОСОБИСТИМИ ФІНАНСАМИ, ШТУЧНИЙ ІНТЕЛЕКТ.

					ІПЗ.КР.Б–121–26–ПЗ			
Змн.	Арк.	№ докум.	Підпис	Дата	Архітектура інтелектуального фінансового асистента на основі ШІ-агента Пояснювальна записка	Літ.	Арк.	Аркушів
Розроб.		Майзенберг П.Д.						
Керівник		Годлевський Ю.О.					4	77
Н. контр.		О.П. Тичина				Житомирська політехніка, група ЗІПЗ-22-1		
Зав. каф.		Т.А. Вакалюк						

ABSTRACT

The bachelor's thesis consists of a "FinAgent" web application for personal finance management and an explanatory note. The explanatory note contains 77 pages, 20 figures, 21 tables, 3 appendices, and 30 references.

The purpose of the work is to develop the architecture of an AI-agent-based financial assistant for tracking and analyzing user's personal finances.

In the sections of the explanatory note, similar software products were analyzed, methods and tools for the project implementation were considered, and technologies for the system development were selected. The distributed client-server architecture was justified, while the database and the use case diagram were designed. Testing was conducted, and the application's functionality was demonstrated.

KEYWORDS: AI ASSISTANT, ARTIFICIAL INTELLIGENCE,
MICROSERVICE ARCHITECTURE, PERSONAL FINANCE MANAGEMENT,
WEB APPLICATION

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ	6
ВСТУП.....	7
РОЗДІЛ 1. АНАЛІЗ НАПРЯМКІВ ВИКОРИСТАННЯ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ ДЛЯ УПРАВЛІННЯ ФІНАНСАМИ.....	10
1.1. Аналіз предметної області	10
1.2. Огляд та порівняльний аналіз існуючих програмних рішень.....	13
1.3. Вибір архітектури системи управління фінансами	17
1.4. Обґрунтування вибору інструментальних засобів та вимоги до апаратного забезпечення.....	21
Висновки до розділу 1	25
РОЗДІЛ 2. ПРОЄКТУВАННЯ ТА РЕАЛІЗАЦІЯ ФІНАНСОВОГО АСИСТЕНТА НА ОСНОВІ ШІ-АГЕНТА.....	26
2.1. Визначення вимог, ролей користувачів та варіантів використання	26
2.2. Розробка бази даних системи	29
2.3. Проєктування алгоритмів роботи системи та взаємодії модулів	33
2.4. Реалізація backend та бізнес-логіки.....	38
2.5. Реалізація ШІ-сервісу та клієнтської частини	41
Висновки до розділу 2	49
РОЗДІЛ 3. ІНТЕРФЕЙС ТА ТЕСТУВАННЯ СИСТЕМИ УПРАВЛІННЯ ФІНАНСАМИ.....	50
3.1. Порядок встановлення та запуску системи.....	50
3.2. Структура інтерфейсу та демонстрація роботи системи.....	53
3.3. Тестування основних сценаріїв роботи системи	60
Висновки до розділу 3	63
ВИСНОВКИ	64
СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ.....	66
ДОДАТКИ	69

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

API – прикладний програмний інтерфейс;

БД – база даних;

СУБД – система управління базами даних;

ПЗ – програмне забезпечення;

PFM – система управління особистими фінансами;

ШІ – штучний інтелект;

LLM – велика мовна модель;

UI – інтерфейс користувача;

UX – досвід користувача під час роботи із застосунком;

REST – архітектурний підхід до побудови веб API;

JWT – токен автентифікації користувача;

CRUD – операції створення, читання, оновлення та видалення даних;

CSV – текстовий формат для експорту табличних даних;

SQL – мова запитів до реляційних баз даних.

ВСТУП

Сучасний світовий банкінг та економіка майже повністю перейшли у цифрову площину, практично витіснивши готівку з повсякденного обігу. Однак доступність онлайн-платежів породила нову проблему хаотичності витрат та повної відсутності у більшості людей системного контролю над власними капіталами. Стандартний мобільний банкінг працює постфактум. Він просто констатує цифру списання, але не дає користувачеві глибокої аналітики, інструментів для стратегічного планування чи прогнозів на основі його реальних споживчих звичок [1; 2; 3; 4].

Саме тому виникла потреба в альтернативних системах управління особистими фінансами (PFM). Ведення таблиць Excel чи паперові записи банально забирають надто багато часу, а більшість наявних на ринку програм або занадто складні в налаштуванні, або взагалі не використовують автоматизацію на базі штучного інтелекту. Водночас повністю автоматичне виконання дій штучним інтелектом було б небезпечним для застосунку, що працює з особистими фінансами. У FinAgent дії, які можуть змінити дані користувача, спочатку формуються як очікувана дія. Після цього користувач має підтвердити її виконання. Лише після підтвердження backend виконує відповідну операцію. Це дозволяє поєднати контроль з боку користувача зі зручністю чатового інтерфейсу [1; 2; 3].

Функціональна структура додатка спирається на кілька ключових учасників взаємодії. Першим є неавторизований користувач. Його можливості обмежені переглядом початкової сторінки, реєстрацією та входом, оскільки фінансові дані мають бути доступні тільки після автентифікації.

Головною дійовою особою в системі є авторизований користувач. Одразу після успішного входу він потрапляє у персональний кабінет, де зосереджені всі інструменти для керування особистими фінансами. Система дає йому доступ до основних дій: можна вносити доходи й витрати, видаляти або редагувати застарілі транзакції, створювати унікальні категорії, гортати історію операцій, застосовувати фільтри та вивантажувати звіти. Окремий фокус зроблено на

					ІПЗ.КР.Б–121–26-ІЗ	7

фінансових цілях. Людина може запланувати велику покупку чи відпустку, а додаток наочно показуватиме прогрес накопичень. Паралельно у системі працює ізольований ШІ-модуль. Це закритий внутрішній сервіс. Серверне ядро звертається до нього лише тоді, коли користувач пише повідомлення у чат. Важливо, що ШІ-модуль не має прямого доступу до зміни записів у базі. Його роль є консультативною та аналітичною. Він розбирає текстові запити, підраховує витрати за обрані проміжки часу, оцінює вплив великих покупок і дає персоналізовані підказки щодо економії.

Об’єктом цього дослідження виступають сучасні вебплатформи, призначені для автоматизації щоденного обліку, аналітики та планування грошових потоків. Предметом дослідження став безпосередній життєвий цикл розробки PFM-застосунку. Він охоплює всі етапи: від проєктування таблиць реляційної бази даних та написання серверної бізнес-логіки на Go до верстки інтерфейсу на React та інтеграції окремого ШІ-компонента для розпізнавання природної мови. Головна мета роботи полягає у створенні та комплексному тестуванні реального, повністю працездатного прототипу системи «FinAgent». Проєкт покликаний показати на практиці, що традиційні фінансові трекери стають зручнішими, якщо поєднати їх із розумним асистентом, який допомагає з аналізом даних у форматі звичайного діалогу.

Для досягнення цієї мети необхідно послідовно вирішити низку інженерних завдань. Спочатку потрібно провести прикладний аналіз ринку готових PFM-додатків та популярних мобільних банківських сервісів. Це дасть змогу чітко виявити їхні технічні обмеження та інтерфейсні прорахунки. Спираючись на результати аналізу, необхідно сформулювати вичерпний перелік вимог до системи «FinAgent». До нього мають увійти авторизація профілів, облік транзакцій, логіка категорій, відстеження накопичень, інтерактивні графіки аналітики та робота інтелектуального чату. Наступним кроком має стати модульне розділення архітектури на чотири ізольовані блоки: клієнтську частину, фінансове серверне ядро, ШІ-сервіс та сховище даних. Таке розмежування обов’язків має зробити систему гнучкою. Для надійного

збереження всієї інформації потрібно спроектувати схему бази даних PostgreSQL. Вона має містити оптимізовані таблиці для профілів, категорій, грошових операцій, фінансових цілей та повної історії листування з асистентом.

Технологічний стек необхідно підібрати під конкретні навантаження. Серверне ядро доцільно реалізувати мовою Go із використанням швидкого фреймворку Gin. Цей модуль має відповідати за автентифікацію, перевірку прав доступу, CRUD-операції з фінансами та зв'язок із ШІ. Сам ШІ-модуль має функціонувати як незалежний мікросервіс мовою Python під керуванням FastAPI. Він повинен приймати текстові речення, аналізувати контекст і повертати назад або готову відповідь, або строго структурований намір дії для бекенду. Клієнтську частину необхідно створити на React із залученням суворої типізації TypeScript. У ній мають бути реалізовані динамічні форми введення даних, віджети дашборда та інтерфейс чату. Весь цей комплекс потрібно запакувати у контейнери та оркеструвати через Docker Compose, завдяки чому система зможе підніматися всього однією командою. Працездатність платформи необхідно підтвердити наскрізним тестуванням ключових сценаріїв, включаючи створення кабінету, ведення обліку, аудит цілей та валідацію команд штучного інтелекту. Практична цінність очікуваних результатів полягає у створенні готового прототипу PFM-системи, що демонструє міжсервісну взаємодію через REST API і може слугувати базою для подальшого розвитку [5; 6; 7; 8; 9; 10; 11; 12; 13].

РОЗДІЛ 1. АНАЛІЗ НАПРЯМКІВ ВИКОРИСТАННЯ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ ДЛЯ УПРАВЛІННЯ ФІНАНСАМИ

1.1. Аналіз предметної області

Справжня проблема сучасних систем управління особистими фінансами криється зовсім не у дефіциті мобільних додатків чи вебсервісів. Головна складність для користувача полягає у великому обсязі нудної ручної роботи. Сьогодні люди щодня користуються різними платіжними інструментами: банківськими картками кількох установ, паперовою готівкою, швидкими онлайн-переказами та автоматичними підписками на цифрові сервіси. Усе це розриває фінансову інформацію на окремі частини. Частина транзакцій автоматично фіксується банкінгом, дещо людина тримає в голові, а дрібні повсякденні витрати взагалі розчиняються безслідно. Через такий хаос зібрати докупи реальну картину власного бюджету практично неможливо [1; 2; 3; 4].

Звичайне сповіщення від банку про списання коштів та назву магазину не дає глибокого розуміння ситуації. Користувачеві необхідно бачити семантичний контекст: до якої категорії належить покупка, які статті витрат забирають найбільше ресурсів, як динамічно змінюється баланс протягом місяця та чи не блокують поточні витрати досягнення глобальних фінансових цілей. Головна місія PFM-системи полягає саме в тому, щоб перетворити сирий потік цифр на структуровані, зрозумілі метрики.

Для цього недостатньо просто зберігати список операцій у вигляді таблиці. Важливо, щоб кожна транзакція мала зрозумілу структуру: суму, дату, тип операції, опис і категорію. Саме категоризація дає змогу перейти від окремих списань до аналізу поведінки користувача. Без цього система може показати лише факт витрати, але не пояснить, які саме напрямки споживання найбільше впливають на бюджет.

Ще одним критичним бар'єром є стрімка втрата мотивації до ведення обліку. Зазвичай усе починається на ентузіазмі. Користувач перші кілька днів сумлінно заповнює форми, обирає категорії та аналізує графіки. Але незабаром

щоденне рутинне клацання по екрану заради фіксації кожної чашки кави, поїздки в транспорті чи купівлі продуктів починає відверто дратувати. Процес перетворюється на бюрократію. Достатньо пропустити всього кілька днів, і аналітика стає неповною. Користувач швидко закидає цю справу. Коли дані втрачають точність, додаток перетворюється на інструмент, який майже не використовується.

Проект покликаний розв'язати ці проблеми через автоматизацію чотирьох ключових архітектурних блоків.

Перший блок охоплює базовий фундамент обліку. Сюди входять реєстрація користувачів, безпечна автентифікація, керування профілями, створення персоналізованих категорій та строга фіксація транзакцій. Кожен грошовий рух обов'язково прив'язується до конкретного користувача і маркується сумою, типом операції, точною датою, описом та категоріальним тегом. Без цього неможливо побудувати коректну історію чи запустити фільтрацію даних. Окремо важливо, що прив'язка транзакцій до користувача є не лише функціональною вимогою, а й елементом захисту персональної фінансової інформації.

Другий вектор завдань фокусується на наочній аналітиці. Замість безликих таблиць система має пропонувати користувачеві агреговані результати у вигляді інформаційного дашборда. На екрані миттєво виводяться сумарні доходи, витрати, актуальний баланс, відсотковий розподіл грошей за категоріями та стрічка останніх операцій. Графічні віджети допомагають оцінити фінансовий стан за лічені секунди. Таке подання даних скорочує час на самостійний аналіз і допомагає користувачеві швидше помічати зміни у власній фінансовій поведінці.

Третій напрям роботи системи відповідає за стратегічне планування. Для цього у проєкті інтегровано інструмент управління фінансовими цілями. Користувач отримує можливість не просто констатувати минулі збитки, а моделювати майбутнє. Можна створювати цілі для накопичень на великі покупки чи подорожі, задавати бажані суми та в реальному часі спостерігати за

шкалою прогресу. Такий підхід дозволяє розглядати фінансовий облік не лише як історію минулих операцій, а і як засіб планування майбутніх рішень.

Четвертим, найбільш інноваційним компонентом системи є інтелектуальний ШІ-чат. Замість блукання складними меню людина може спілкуватися з платформою звичайною розмовною мовою. ШІ-помічник здатен миттєво підрахувати витрати за тиждень, оцінити, як покупка нового гаджета вплине на місячний бюджет, або дати практичну пораду щодо оптимізації витрат. У такому форматі чат виступає додатковим інтерфейсом доступу до фінансової аналітики, а не заміною основних таблиць, форм і графіків.

При цьому архітектура взаємодії зі штучним інтелектом підпорядкована суворому правилу безпеки. Пряма модифікація бази даних нейромережею не допускається. ШІ-модуль не має можливості самостійно створювати чи видаляти записи, що зменшує ризик помилкових дій через неточну відповідь моделі. Агент здатний лише згенерувати стандартизований намір дії. Backend-сервер ретельно валідує цей пакет даних, після чого виводить користувачеві інтерактивну картку підтвердження. Лише після явної команди людини операція потрапляє до реляційного сховища.

Таким чином, предметна область системи охоплює не тільки класичний облік доходів і витрат, а й аналітику, планування та контрольовану взаємодію з ШІ-асистентом. Саме поєднання цих блоків дає змогу розглядати проекти як RFM-систему, у якій автоматизація не усуває користувача з процесу прийняття рішень, а допомагає йому швидше аналізувати власні фінансові дані.

Додатково у межах предметної області важливо враховувати зручність роботи з історією операцій. З часом кількість транзакцій користувача зростає, тому система має підтримувати не лише їх збереження, а й швидкий пошук, фільтрацію за категоріями, датами чи типом операції, а також сортування й поступове завантаження записів. Це дає змогу не перевантажувати інтерфейс зайвими даними та залишати роботу з фінансовою історією зрозумілою навіть при великій кількості записів.

1.2. Огляд та порівняльний аналіз існуючих програмних рішень

Перш ніж формувати технічні вимоги до системи «FinAgent», необхідно детально дослідити роботу наявних на ринку продуктів. Для аналізу було обрано три автономні міжнародні сервіси (Spendee, PocketGuard, YNAB) та один інтегрований інструмент українського ринку (внутрішня аналітика Monobank). Основну увагу приділено архітектурі інтерфейсів, методам збору фінансових даних та логіці класифікації транзакцій.

Сервіс **Spendee** будує взаємодію з користувачем навколо хронологічної стрічки операцій. Основний екран програми відображає кожну витрату у вигляді окремої картки з іконкою категорії, сумою та коментарем. Для автоматизації обліку Spendee використовує сторонні банківські агрегатори (наприклад, Salt Edge), які підключаються до рахунків через відкриті API. Головний екран системи та приклад візуалізації категорій наведено на рисунку 1.1 [1; 14].

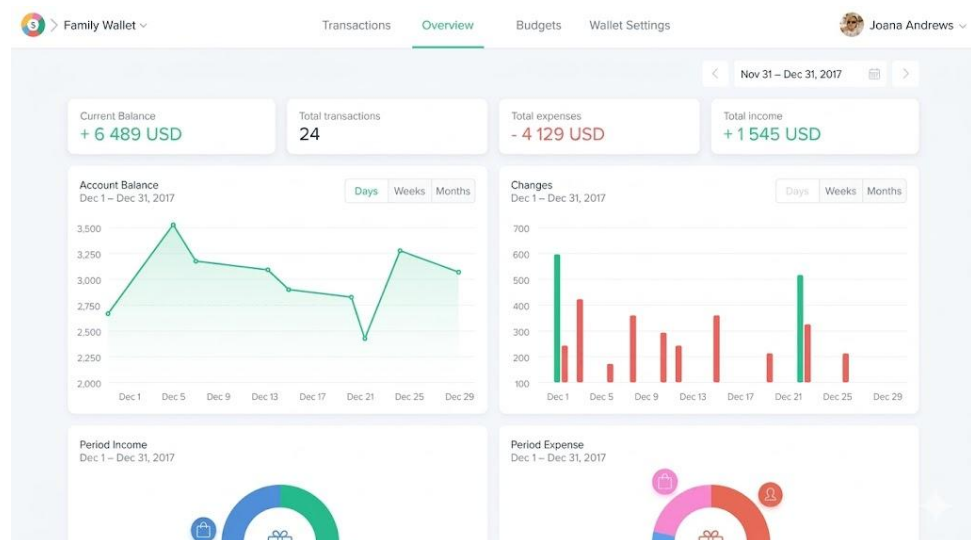


Рис. 1.1. Інтерфейс аналітичної панелі та стрічки транзакцій у додатку Spendee

Головна проблема Spendee – це відсутність будь-якої інтелектуальної обробки. Якщо дані вносяться вручну, доводиться проходити через трирівневе меню: вибір типу операції, пошук потрібної категорії серед десятка іконок і лише потім введення суми. Це створює серйозне рутинне навантаження.

Застосунок **PocketGuard** використовує зовсім іншу логіку інтерфейсу. Його головна особливість – великий віджет на робочому столі із назвою «In My

Pocket» (В кишені). Математична модель додатка автоматично вираховує суму, яку можна вільно витратити прямо зараз. Для цього програма бере загальний дохід користувача, віднімає від нього зафіксовані обов'язкові платежі та кошти, які вирішено відкласти на фінансові цілі, а отриману різницю розподіляє на поточний місяць. Система вміє самостійно знаходити у виписках регулярні підписки (Netflix, Spotify, комунальні платежі) та бронювати під них гроші наперед. Приклад роботи цього віджета та загального інтерфейсу показано на рисунку 1.2 [2].

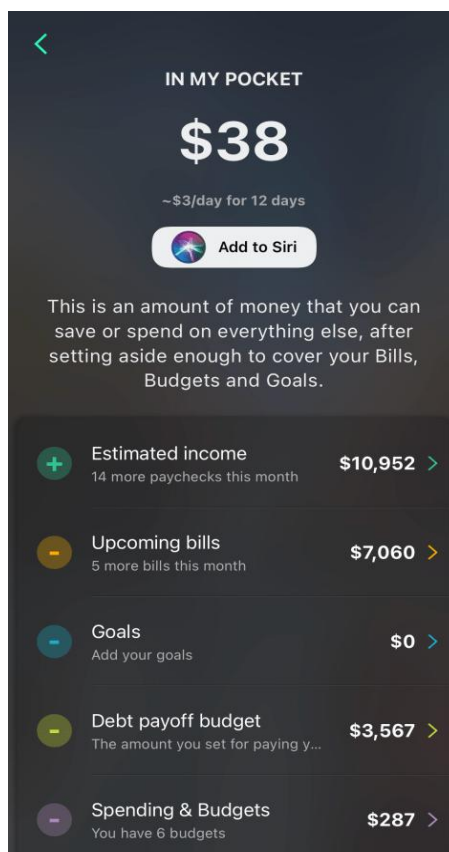


Рис. 1.2. Відображення розрахунку вільного балансу в інтерфейсі PocketGuard

Технічний мінус PocketGuard полягає в алгоритмі розпізнавання транзакцій. Програма намагається вгадати категорію витрати суто за текстовим рядком від банку. Якщо алгоритм бачить назву типу «FOP PETROV M.V.», він не може зрозуміти, що це – покупка одягу, оплата кави чи ремонт авто. Система просто закидає такий запис у категорію «Інше», через що виникає необхідність ручного перепризначення операцій.

Американська платформа **YNAB** (You Need A Budget) вважається найбільш професійним, але водночас найскладнішим інструментом. Інтерфейс програми виконаний у вигляді великої інтерактивної таблиці, схожої на Excel. Логіка базується на чотирьох правилах нульового бюджету, де кожна копійка має бути розписана на конкретне завдання (наприклад, «оренда», «продукти», «ліки») одразу після її надходження. У YNAB є функція «віку грошей» (Age of Money), яка показує, скільки днів у середньому кошти лежать на рахунку перед тим, як бути витраченими. Структуру робочої таблиці планування YNAB продемонстровано на рисунку 1.3 [3].

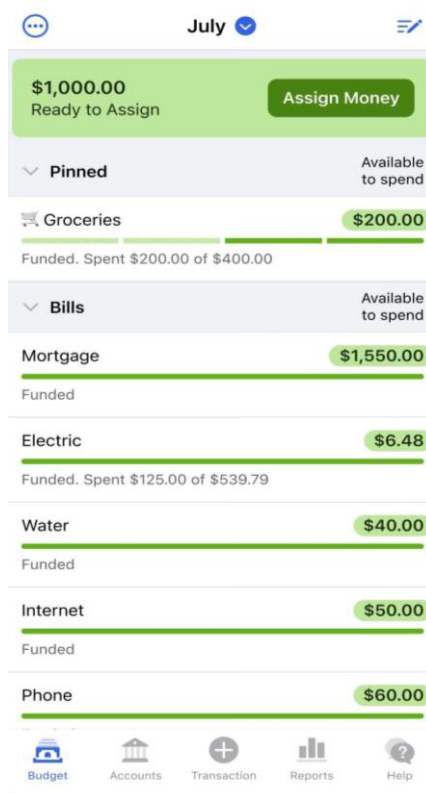


Рис. 1.3. Модуль розподілу бюджету та планування витрат у системі YNAB

Через таку складну табличну структуру додаток має високий поріг входження. Доводиться витратити години на планування бюджетних стовпчиків на місяць наперед. Жодних інструментів автоматизації розпізнавання тексту чи швидкого введення через діалог тут немає. Для звичайного використання, коли

потрібно просто фіксувати щоденні витрати, YNAB є занадто перевантаженим рішенням.

Внутрішній фінансовий помічник у додатку **Monobank** є прикладом автоматизації в межах одного банку. Інтерфейс аналітики інтегрований у мобільний клієнт. Користувач бачить кругову діаграму витрат за поточний місяць, де кожен сектор відповідає за певну категорію. Запис про витрату створюється миттєво в момент прикладання картки до термінала. Екран фінансової статистики банку наведено на рисунку 1.4 [4].

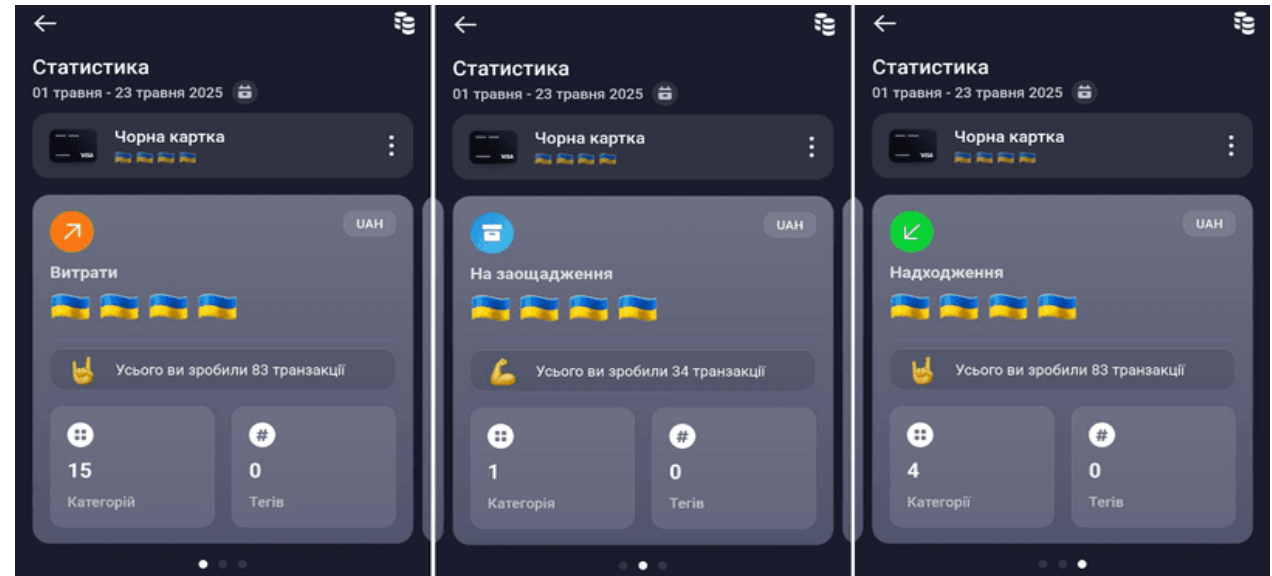


Рис. 1.4. Екран аналітики та автоматичного розподілу транзакцій у додатку Monobank

Головне архітектурне обмеження Monobank – це закритість екосистеми. У разі зняття готівки додаток відображає це як одну велику витрату «Зняття готівки». Що саме було куплено на ці паперові гроші – система відстежити не може. Також розподіл за категоріями прив'язаний виключно до чотиризначних кодів MCC (Merchant Category Code), які зашиті в банківському терміналі магазину. Якщо техніка купується в супермаркеті, де термінал зареєстрований під кодом «Продукти», вся сума піде в категорію їжі, і змінити це налаштування неможливо.

Для детального порівняння всіх технічних характеристик та функцій досліджених систем було складено матрицю параметрів, куди внесено також показники розроблюваного рішення «FinAgent» (табл. 1.1).

Порівняння аналогів із системою «FinAgent»

Функціональний критерій	Spendee	YNAB	PocketGuard	Monobank	FinAgent
Наявність безкоштовної версії	Частково	Ні	Частково	Так	Так
Інтелектуальна категоризація (ШІ)	Ні	Ні	Частково	Так	Так (LLM)
Наявність ШІ-асистента (чат)	Ні	Ні	Ні	Ні	Так
Мультиплатформеність (Web + Mobile)	Так	Так	Так	Ні	Так
Можливість ведення готівки	Так	Так	Так	Ні	Так
Гнучкість налаштування категорій	Частково	Так	Частково	Частково	Так

Висновки з аналізу показують, що поточний ринок змушує обирати між двома крайнощами. З одного боку існує жорстка банківська автоматизація без можливості кастомізації (Monobank), з іншого – гнучкі додатки, які вимагають щодня монотонно вводити кожну транзакцію вручну, якщо користувач хоче детально слідкувати за фінансами (Spendee, YNAB).

Розроблювана система покликана закрити цю нішу. Створюється архітектура системи, яка дає повну свободу у налаштуванні гаманців та категорій, але прибирає рутину ручного введення завдяки ШІ-агенту. Замість натискання кнопок інтерфейсу з'являється можливість у простій формі написати опис операції звичайними словами у вікно чату, спростивши монотонний процес.

1.3. Вибір архітектури системи управління фінансами

Проектні рішення при створенні архітектури «FinAgent» диктувалися необхідністю виконати функціональні вимоги і одночасно зберегти прозорість кодової бази для подальшого супроводу та демонстрації роботи системи. Оскільки додаток оперує різномірними технологіями, вибір припав на розподілений клієнт-серверний підхід. Це оптимальний варіант для такого

прототипу. Він позбавлений надмірної інфраструктурної складності, але розмежовує зони відповідальності між компонентами платформи. Браузерна клієнтська частина повністю фокусується на взаємодії з користувачем. Вона відповідає за швидкий рендеринг інтерактивних форм, таблиць, динамічних графіків аналітики, відстеження фінансових цілей та діалогового інтерфейсу чату. Серверний блок виконує роль головного регулятора бізнес-правил. Саме тут відбувається автентифікація, перевірка токенів доступу та безпосередній запис операцій. Backend виступає єдиним інструментом для зміни фінансових станів платформи. При цьому роботу зі штучним інтелектом свідомо виділено в ізольовану структуру. Робота з мовною моделлю, промптами та викликом інструментів вимагає специфічного середовища, яке простіше розгортати всередині Python-екосистеми [15; 16].

Такий підхід також дозволяє уникнути змішування різних типів логіки в одному модулі. Фінансові операції, перевірка прав доступу та робота з базою даних залишаються у відповідальності backend-рівня, тоді як ШІ-сервіс працює лише з обробкою текстових запитів, формуванням відповідей і підготовкою структурованих намірів дій. Завдяки цьому кожна частина системи має чітке призначення. Якщо у майбутньому виникне потреба змінити ШІ-модель або спосіб взаємодії з LLM API, це можна буде зробити без повного переписування основного фінансового ядра.

Всередині самого серверного ядра реалізовано шарову структуру за принципами Clean Architecture. Головне правило тут полягає у розділенні відповідальності між частинами коду. Веб-обробники не повинні самостійно формувати SQL-запити чи прораховувати фінансову логіку. Замість цього вхідний потік даних послідовно проходить крізь три ізольовані рівні: handlers, usecases та repositories. Рівень handler приймає HTTP-запит, розпаковує його та валідує параметри. Далі управління переходить до usecase, де реалізовано правила предметної області. Завершує цей ланцюжок шар repository, який безпосередньо взаємодіє з базою даних PostgreSQL для витягування чи збереження інформації [16; 17].

Поділ на такі рівні спрощує супровід системи. Якщо змінюється формат HTTP-запиту, достатньо відредагувати handler. Якщо змінюється бізнес-правило, наприклад порядок перевірки транзакції або логіка виконання очікуваної дії, зміни вносяться на рівні usecase. Якщо ж потрібно змінити спосіб збереження даних або оптимізувати SQL-запити, це стосується repository-рівня. Така організація коду робить backend менш залежним від конкретної технології доступу до бази даних і дозволяє точніше контролювати місця, де відбуваються критичні фінансові операції.

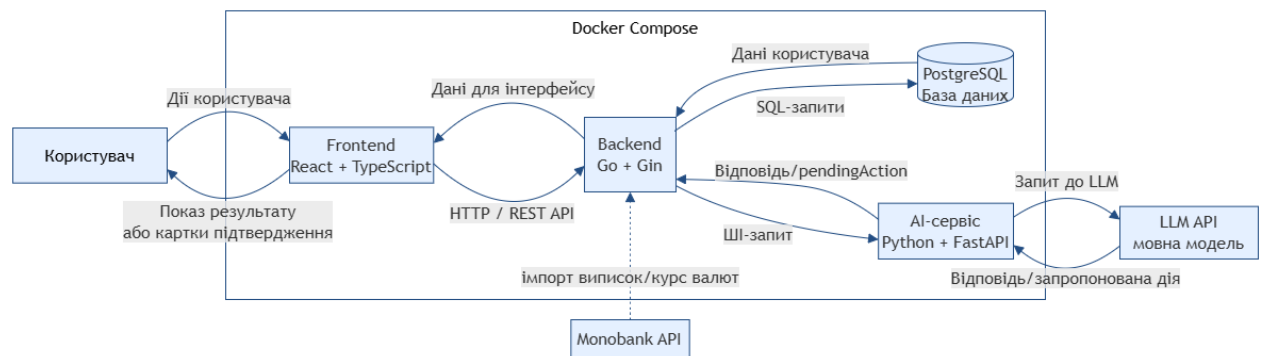


Рис. 1.5. Основні функціональні блоки системи «FinAgent»

На рисунку 1.5 наочно зафіксовано логічну схему взаємодії цих ключових вузлів системи. Користувач надсилає команди через React-інтерфейс. Якщо людина починає текстовий діалог із ШІ-асистентом, backend приймає запит, формує безпечний контекст і перенаправляє його до FastAPI-сервісу через внутрішній REST API [9; 16; 18; 19].

У стандартних сценаріях, наприклад під час перегляду транзакцій, редагування категорій або роботи з фінансовими цілями, запит також проходить через backend. Він перевіряє автентифікацію користувача, викликає відповідний usecase і лише після цього звертається до PostgreSQL. Таким чином, frontend не має прямого доступу до бази даних, а всі фінансові операції проходять через контрольований серверний рівень.

Важливою основою безпеки є усунення ШІ-сервісу від прямого керування базою даних. Мовна модель не може самостійно внести запис про нову витрату чи модифікувати параметри фінансової цілі. Вона може лише розпізнати намір людини та упакувати його у структурований об'єкт. Остаточне рішення про

виконання операції, яка змінює дані, завжди залишається за фінансовим ядром backend і підтвердженням від самого користувача. Такий підхід зменшує ризик хаотичних або помилкових змін даних через можливі неточності відповіді мовної моделі, що особливо важливо для системи персональних фінансів [18; 19; 20; 21].

Таблиця 1.2

Порівняння можливих архітектурних підходів

Підхід	Переваги	Недоліки	Доцільність для проекту
Монолітний застосунок	Проста структура, швидкий старт розробки	Складніше окремо масштабувати ШІ-логіку	Можливий, але менш зручний для ШІ-сервісу
Клієнт-серверна архітектура	Зрозумілий поділ UI, API та БД	Потребує налаштування взаємодії компонентів	Основний вибір для системи
Мікросервісна архітектура	Гнучке масштабування компонентів	Може бути надмірною для бакалаврського прототипу	Використано частково через окремий ШІ-сервіс
Serverless-підхід	Мінімальна підтримка серверів	Складніше локальне тестування і контроль стану	Не обрано для основної реалізації

Окрім того, архітектурний каркас проектувався з розрахунком на подальше розширення. Платформа містить модулі для підключення зовнішніх систем, зокрема API Monobank. Він може використовуватися для автоматичного імпорту валютних курсів та банківських виписок, проте ядро системи залишається автономним і не залежить від сторонніх сервісів. Це демонструє можливість розвитку проекту, але не перевантажує бакалаврську роботу зайвими деталями. Розгортання повноцінної мікросервісної мережі було б надлишковим кроком для академічного прототипу (табл. 1.2). З іншого боку, чистий моноліт обмежив би можливість поєднати фінансові розрахунки на Go та обробку LLM на Python. Обраний комбінований архітектурний патерн з чотирьох базових елементів, а саме frontend, backend, ШІ-сервісу та реляційного сховища, став практичним інженерним компромісом для реалізації системи [4].

1.4. Обґрунтування вибору інструментальних засобів та вимоги до апаратного забезпечення

Серверне ядро «FinAgent» побудоване на базі мови Go та вебфреймворку Gin. Таке рішення має чітке інженерне обґрунтування. Go демонструє високу швидкість компіляції та добру продуктивність. Строга статична типізація допомагає зменшити кількість помилок ще на етапі написання коду, а модель конкурентності дозволяє обробляти паралельні запити. Важливою перевагою є можливість розгортання системи у вигляді єдиного самодостатнього бінарного файлу, що зменшує потребу у великій кількості зовнішніх залежностей на сервері. Фреймворк Gin у цій зв'язці бере на себе рутинні завдання: маршрутизацію HTTP-запитів, налаштування проміжних шарів middleware та побудову REST API. Для бакалаврського проєкту це практичний вибір, оскільки він дозволяє реалізувати ефективний backend-стек без зайвого ускладнення архітектури [17; 22].

Для надійного збереження системної інформації залучено СУБД PostgreSQL. Вибір реляційного сховища тут є закономірним, оскільки фінансові дані за своєю природою вимагають структуризації та контролю зв'язків. Користувач має пов'язані з ним транзакції, власні категорії, фінансові цілі, логи інтелектуального чату та історію системних дій. У фінансовому обліку важливо забезпечити цілісність даних, типізацію полів та коректну роботу зовнішніх ключів на рівні таблиць. Разом із тим PostgreSQL пропонує розробнику корисний інструмент у вигляді типу даних JSONB. Це дозволяє зберігати динамічні об'єкти, наприклад структуровані відповіді від зовнішньої мовної моделі або персональні конфігурації розташування віджетів на користувацькому дашборді в особистому кабінеті [5; 6; 23; 24].

Інтелектуальний ШІ-сервіс функціонує окремо. Його написано мовою Python з використанням сучасного асинхронного фреймворку FastAPI. Логіка такого розділення є зрозумілою: саме Python має розвинену екосистему для роботи з великими мовними моделями, промптами та інтеграцією зовнішніх ШІ-сервісів. У свою чергу, FastAPI дає можливість швидко розгорнути HTTP-

ендпоїнти з автоматичною валідацією вхідних даних. У загальному контурі «FinAgent» цей сервіс відповідає за підтримку діалогу в чаті, обробку текстових запитів користувача та формування структурованих відповідей для backend. Проте він не має власного постійного сховища даних і не має права вносити прямі зміни в транзакційну історію [7; 18].

Візуальний клієнтський інтерфейс розроблено на базі бібліотеки React, мови TypeScript та інструменту збірки Vite. Компонентний підхід React дозволив створити гнучку архітектуру інтерфейсу, де кожен елемент, наприклад вікно чату чи форма транзакції, є окремою частиною коду. TypeScript допоміг зменшити кількість помилок, пов'язаних із типами даних на frontend. Використання Vite прискорило локальний процес розробки завдяки швидкому оновленню модулів. Для побудови фінансової інфографіки впроваджено бібліотеку Recharts, яка підходить для рендерингу адаптивних графіків. Глобальний стан інтерфейсу збалансовано: дані розділені між локальними станами компонентів та контекстами застосунку. Цей стек закрив потребу у створенні швидкого односторінкового вебзастосунку, який оновлює інформацію на панелі керування без перезавантаження сторінок [8; 9; 10; 11].

Щоб зв'язати всі ці різні технологічні шари в єдину систему, застосовано контейнеризацію через Docker Compose. Frontend, backend, ШІ-сервіс та реляційна база даних запускаються в ізольованих, але пов'язаних між собою контейнерах, які спілкуються через внутрішню віртуальну мережу. Це зменшує проблему працездатності коду на різних операційних системах. Проєкт можна розгорнути на іншій машині однією командою, незалежно від локально встановлених версій Go, Python чи Node.js [12; 13; 25].

Питанням безпеки в архітектурі «FinAgent» приділено окрему увагу. Процес автентифікації користувачів реалізовано за допомогою токенів JWT, тоді як захист паролів покладено на алгоритм хешування bcrypt. Механізм JWT дозволяє клієнтській частині зберігати сесію та передавати токен доступу в заголовках HTTP-запитів до захищених ендпоїнтів фінансового ядра. Алгоритм bcrypt, у свою чергу, не дозволяє зберігати паролі у відкритому вигляді всередині

бази даних, що зменшує ризики у випадку компрометації таблиць. Окремо передбачено захист чутливих параметрів інтеграцій із зовнішніми сервісами, зокрема Monobank, оскільки такі ключі можуть відкривати доступ до конфіденційної фінансової інформації користувача [19; 20; 21; 27].

Для узагальнення вибору технологій доцільно подати їх у вигляді таблиці. Це дозволяє коротко показати, який компонент системи за що відповідає, яка технологія використовується для його реалізації та чому саме вона була обрана. Такий підхід також допомагає не перевантажувати основний текст повторенням технічних деталей, оскільки основні аргументи зведено в табл. 1.3.

Таблиця 1.3

Обрані інструментальні засоби реалізації

Компонент	Технологія	Призначення	Причина вибору
Backend	Go, Gin	REST API і бізнес-логіка	Швидкодія, статична типізація, просте розгортання
База даних	PostgreSQL	Зберігання фінансових і службових даних	Реляційні зв'язки, цілісність даних, підтримка JSONB
ШІ-сервіс	Python, FastAPI	Обробка природної мови та ШІ-відповіді	Зручна LLM-екосистема, типізовані HTTP API
Frontend	React, TypeScript, Vite	Інтерфейс користувача	Компонентність, типізація, швидка збірка
Розгортання	Docker Compose	Локальне багатоконтейнерне середовище	Повторюваний запуск та ізоляція сервісів
Безпека	JWT, bcrypt, шифрування токенів	Автентифікація, захист паролів і ключів	Поширені підходи до захисту даних

Як видно з табл. 1.3, обраний стек не прив'язаний до однієї технологічної платформи. Кожен інструмент виконує окрему роль у системі. Go і Gin використовуються для швидкої та типізованої серверної логіки, PostgreSQL відповідає за структуроване зберігання фінансових даних, Python і FastAPI винесені в окремий ШІ-сервіс, а React із TypeScript забезпечують зручний клієнтський інтерфейс. У результаті система залишається модульною, але не

перетворюється на надмірно складну мікросервісну інфраструктуру [5; 6; 7; 8; 9; 10; 11; 12; 13].

Окремо потрібно врахувати вимоги до середовища запуску. Оскільки «FinAgent» складається з кількох сервісів, для його роботи потрібні ресурси не тільки для backend і frontend, а й для ШІ-сервісу та бази даних PostgreSQL. Мінімальні вимоги до локального запуску для демонстрації повного циклу розробки вебзастосунку від аналізу та проектування до реалізації, тестування й опису роботи системи, наведено в табл. 1.4.

Таблиця 1.4

Мінімальні вимоги до середовища запуску

Компонент	Мінімальна вимога	Пояснення
Операційна система	Windows, Linux або macOS	Docker Compose забезпечує однакове середовище запуску
Оперативна пам'ять	8 ГБ	Потрібна для одночасного запуску кількох контейнерів
Процесор	2 ядра	Достатньо для локальної демонстрації прототипу
Дисковий простір	5–10 ГБ	Потрібний для залежностей, образів контейнерів і бази даних
Браузер	Сучасний Chrome, Edge або Firefox	Потрібний для доступу до React-інтерфейсу
Мережа	Доступ до API мовної моделі за потреби	Потрібний для виконання реальних ШІ-викликів

Апаратні вимоги в табл. 1.4 сформульовано насамперед для локальної демонстрації та перевірки роботи прототипу. У реальному середовищі ці параметри можуть змінюватися залежно від кількості користувачів, обсягу історії транзакцій, частоти звернень до ШІ-сервісу та способу розгортання. Для бакалаврської роботи локального запуску через Docker Compose достатньо, оскільки він дозволяє перевірити працездатність усіх основних компонентів системи в одному відтворюваному середовищі. Також локальне розгортання дає змогу оцінити не лише окремі модулі, а й їхню взаємодію: frontend, backend, базу даних та ШІ-сервіс. Це особливо важливо для демонстрації архітектури, оскільки всі компоненти працюють як єдина система, але залишаються логічно розділеними [13].

Висновки до розділу 1

У першому розділі було проаналізовано предметну область персонального фінансового обліку та визначено задачі, які доцільно автоматизувати у вебзастосунку. Встановлено, що користувачу потрібна не лише історія транзакцій, а й засоби категоризації, аналітики, роботи з цілями та пояснення фінансових даних у простій формі.

Проведений аналіз аналогів показав, що існуючі рішення або зосереджені на ручному обліку, або прив'язані до банківської екосистеми, або мають високий поріг входження. Розроблювана система відрізняється поєднанням класичного фінансового обліку з контрольованим ШІ-чатом, у якому дії зі зміною даних виконуються лише після підтвердження користувача.

Обґрунтовано архітектуру системи, у якій frontend, backend, ШІ-сервіс і PostgreSQL мають окремі полі. Вибраний стек Go/Gin, PostgreSQL, Python/FastAPI, React/TypeScript/Vite і Docker Compose відповідає задачам бакалаврської роботи та забезпечує достатню модульність, відтворюваність і можливість подальшого розвитку програмного продукту.

РОЗДІЛ 2. ПРОЄКТУВАННЯ ТА РЕАЛІЗАЦІЯ ФІНАНСОВОГО АСИСТЕНТА НА ОСНОВІ ШІ-АГЕНТА

2.1. Визначення вимог, ролей користувачів та варіантів використання

Під час проєктування архітектури «FinAgent» було враховано три основні принципи: інтерфейс має бути зрозумілим для користувача, фінансові записи повинні бути захищені від неконтрольованих змін, а ШІ-модуль має залишатися окремим компонентом, який можна розвивати або замінювати незалежно від основної бізнес-логіки. Такий підхід дозволяє розвивати або повністю змінювати нейромережевий модуль, не втручаючись у стабільне фінансове ядро системи. На цьому аналітичному етапі було зафіксовано фінальний набір вимог, визначено ролі дійових осіб та описано логіку сценаріїв, за якими працює система [28].

У системі виділено дві основні зовнішні ролі: неавторизований користувач і авторизований користувач. Можливості неавторизованого користувача обмежені створенням нового акаунта або входом до вже існуючого профілю. Після проходження автентифікації він отримує доступ до основних інструментів керування особистими фінансами. Система відкриває йому доступ до управління транзакціями, створення категорій, трекера фінансових цілей, інтерактивного дашборда з аналітикою, інструментів вивантаження даних та діалогового вікна чату. Окремим внутрішнім компонентом є ШІ-сервіс. Це не жива людина і не користувач застосунку, а ізольований програмний компонент. Він взаємодіє з серверним ядром через обмежені API-виклики, допомагаючи обробляти запити, але не маючи прямого доступу до бази даних.

Функціональні вимоги сформовано як перелік основних дій: від входу та редагування профілю до повного циклу CRUD-операцій із фінансовими записами, фільтрації історії та експорту даних у форматі CSV. Сюди також входять ведення фінансових цілей, відображення агрегованої аналітики, підтримка контекстного діалогу з ШІ-модулем, розрахунок гіпотетичних сценаріїв «що, якщо» та виявлення нетипових витрат користувача.

Нефункціональні вимоги зосереджені на ізоляції даних між різними обліковими записами, відтворюваності інфраструктурного запуску в Docker-середовищі, зручності інтерфейсу та стабільності ключових сценаріїв системи [29].

Особливістю вимог до ШІ-модуля є наявність обмеження: мовна модель не повинна самостійно змінювати фінансові дані. Якщо в діалозі користувач просить створити транзакцію або змінити ціль, ШІ-сервіс формує структурований опис очікуваної дії. Backend зіставляє її з даними користувача і повертає картку підтвердження. Лише після натискання користувачем відповідної кнопки дія виконується і записується в журнал дій [18; 19].

Варіанти використання системи були визначені так, щоб кожен основний сценарій мав відповідне відображення у коді. Наприклад, робота з транзакціями реалізується через backend-обробники, usecase-рівень, repository-рівень і сторінку frontend. Робота з ШІ-чатом охоплює frontend-компонент, backend ChatHandler, клієнт ШІ-сервісу та Python-модуль, який формує відповідь або очікувану дію.

Для перевірки повноти вимог було використано принцип трасованості. Кожна заявлена функція має бути пов'язана з конкретними файлами або модулями проєкту. Якщо функція реалізована частково, вона не повинна описуватися як основна завершена можливість. Саме тому експериментальна ШІ-категоризація у цій роботі подається як додатковий модуль, тоді як ШІ-чат, транзакції, цілі, дашборд і нетипові витрати належать до основного функціоналу.

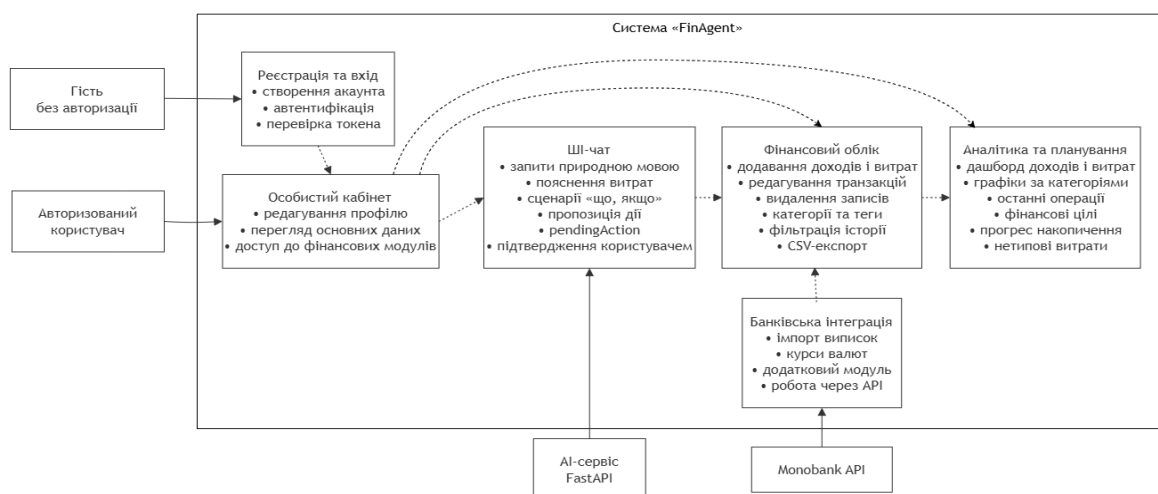


Рис. 2.1. Схема варіантів використання системи

На рисунку 2.1 подано контекстну схему основних варіантів використання системи «FinAgent». У ній функціонал згруповано за ключовими сценаріями роботи: реєстрація та вхід, особистий кабінет, фінансовий облік, аналітика, ШІ-чат і банківська інтеграція.

Неавторизований користувач має доступ лише до реєстрації та входу. Авторизований користувач працює з особистим кабінетом, транзакціями, категоріями, аналітикою, фінансовими цілями та ШІ-чатом. ШІ-сервіс показано як окремий внутрішній компонент, який обробляє запити природною мовою, але не має прямого доступу до бази даних. Monobank API відображено як зовнішній сервіс для імпорту банківських даних і курсів валют [4; 18].

Таблиця 2.1

Функціональні вимоги до системи

Група вимог	Опис	Реалізація у проєкті
Автентифікація	Реєстрація, вхід, JWT-доступ, профіль	auth routes, user usecase, ProfilePage
Транзакції	Додавання, редагування, видалення, фільтрація, пагінація	transaction handler/usecase/repository, TransactionsPage
Категорії	Системні та користувацькі категорії доходів і витрат	category usecase, CategoriesPage
Аналітика	Підсумки доходів і витрат, розподіл за категоріями, останні операції	stats API, dashboard widgets
Фінансові цілі	Створення цілей, внесення та зняття коштів, прогрес	goal handler/usecase, GoalsPage
ШІ-чат	Питання про фінанси, що-якби, очікувані дії	chat handler, AI service, ChatPanel
Нетипові витрати	Пошук витрат, що відрізняються від звичного рівня	audit endpoint, AnomaliesPage

Табл. 2.1 узагальнює функціональні вимоги та показує, у яких частинах проєкту вони реалізовані. Така таблиця важлива для кваліфікаційної роботи, оскільки пов'язує вимоги з фактичними модулями програмного продукту. У подальших підрозділах ці групи вимог деталізуються через структуру бази даних, алгоритми та компоненти реалізації.

Нефункціональні вимоги до системи

Вимога	Зміст вимоги	Спосіб забезпечення
Безпека доступу	Користувач бачить тільки власні дані	JWT, userID у контексті, перевірка власника сутностей
Захист паролів	Паролі не зберігаються у відкритому вигляді	bcrypt-хешування
Контроль ШІ-дій	ШІ не змінює дані без підтвердження	pendingAction і окремий endpoint виконання
Відтворюваність запуску	Система має запускатися в однаковому середовищі	Docker Compose та .env.example
Підтримуваність	Код має бути розділений за відповідальністю	domain, usecase, repository, delivery/http
Межі ШІ-відповідей	Поради мають загальний характер	обмежений контекст і формулювання у чаті

Нефункціональні вимоги з табл. 2.2 формують інженерні обмеження системи та визначають умови, за яких програмний продукт має залишатися безпечним, стабільним і зручним для використання. Найважливішими для цього проєкту є ізоляція даних користувачів і контроль дій ШІ-асистента. Саме ці вимоги вплинули на архітектурне рішення, за яким ШІ-сервіс повертає тільки пропозицію дії, а backend виконує її після перевірки та підтвердження. Такий підхід дозволяє зменшити ризик випадкових змін у фінансових записах і зберегти контроль над критичними операціями на стороні серверної логіки.

2.2. Розробка бази даних системи

Для забезпечення зберігання фінансових операцій користувача, категоризації витрат, роботи аналітики та підтримки ШІ-чату в системі «FinAgent» спроектовано реляційну базу даних. Для реалізації обрано PostgreSQL, оскільки ця СКБД підтримує транзакційність, зовнішні ключі, числові типи для точного зберігання сум і JSONB-поля для службових структурованих даних [7; 15; 27].

Схему бази даних спроектовано так, щоб таблиці не змішували різні типи даних в одному місці. Таблиця users відповідає за облікові дані користувача. Фінансова частина винесена в окремі сутності: categories групує доходи й витрати, transactions фіксує фактичні фінансові операції, а goals відповідає за майбутні накопичення. Зв'язки через зовнішні ключі дозволяють backend перевіряти, чи належить конкретний запис поточному користувачу. Завдяки цьому дані різних користувачів не змішуються між собою, а запити до транзакцій, категорій, цілей та інших сутностей виконуються тільки в межах відповідного облікового запису.

Для сум грошових операцій використовується числовий тип, який дозволяє уникати помилок округлення, характерних для чисел з плаваючою комою. Це важливо для фінансових даних, оскільки суми доходів, витрат і цілей повинні зберігатися точно.

Логічна схема бази даних складається з основних фінансових таблиць і додаткових службових таблиць. Взаємодію між ними та ключові зв'язки відображено на ER-діаграмі (рис. 2.2).

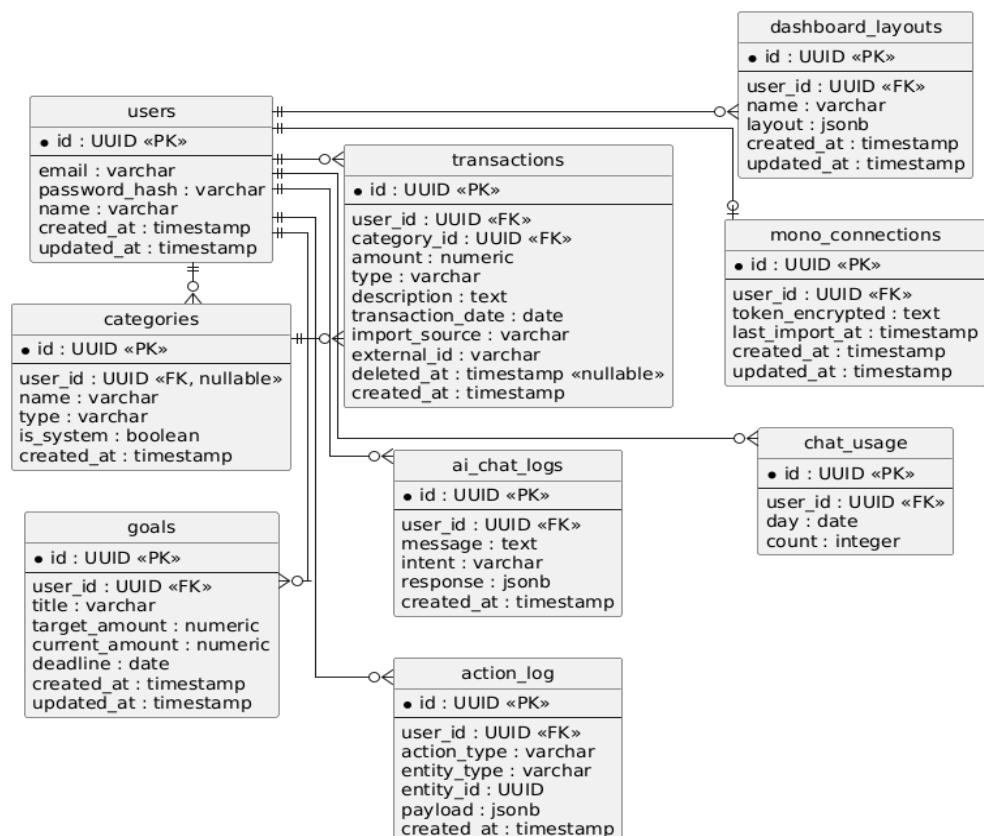


Рис. 2.2. ER-діаграма бази даних системи «FinAgent»

Рисунок 2.2 демонструє повну схему бази даних системи «FinAgent». Центральною таблицею є users, оскільки більшість записів прив'язана до конкретного користувача через поле user_id. Основу фінансового обліку формують таблиці transactions, categories і goals. Додаткові таблиці ai_chat_logs, action_log, dashboard_layouts, mono_connections і chat_usage підтримують роботу ІІІ-чату, журналу дій, дашборда, банківської інтеграції та обмеження кількості ІІІ-запитів.

Таблиця 2.3

Призначення основних таблиць бази даних

Таблиця	Призначення	Ключові поля
users	Облікові записи користувачів	id, email, password_hash, name
categories	Системні та користувацькі категорії	id, user_id, name, type
transactions	Доходи й витрати користувача	id, user_id, category_id, amount, type, transaction_date
goals	Фінансові цілі	id, user_id, title, target_amount, current_amount
ai_chat_logs	Історія звернень до ІІІ-сервісу	user_id, message, intent, response
action_log	Журнал дій для можливого скасування	action_type, entity_type, entity_id, payload
dashboard_layouts	Налаштування розташування віджетів	user_id, name, layout
mono_connections	Додаткове підключення Monobank	user_id, token_encrypted, last_import_at
chat_usage	Лічильник ІІІ-звернень	user_id, day, count

У табл. 2.3 наведено призначення основних таблиць бази даних. Найважливішими для користувацького сценарію є users, transactions, categories і goals, оскільки саме вони забезпечують облік фінансових даних. Таблиці ai_chat_logs і action_log підтримують роботу ІІІ-чату та можливість контролю виконаних дій. Інші таблиці використовуються для додаткових сценаріїв: налаштування дашборда, підключення Monobank і обмеження кількості звернень до ІІІ-сервісу.

Під час проєктування бази даних окремо враховано питання цілісності фінансових даних. Для цього основні записи прив'язуються до користувача через поле `user_id`, а зв'язок транзакції з категорією контролюється через `category_id`. Такий підхід дозволяє backend перевіряти не тільки сам факт існування запису, а й його належність поточному користувачу.

Для прискорення типових запитів доцільно використовувати індекси за полями, які найчастіше застосовуються під час фільтрації. Насамперед це `user_id`, дата транзакції, тип операції та категорія. Завдяки цьому сторінка історії операцій, дашборд і аналітичні запити можуть працювати з вибіркою конкретного користувача без повного перегляду таблиці транзакцій [7].

Механізм `soft-delete` у системі використовується для зменшення ризику випадкової втрати даних. Замість повного фізичного видалення запис може бути позначений як видалений через поле `deleted_at`. Для користувача така операція виглядає як звичайне видалення, однак на рівні бази даних залишається можливість відновлення логіки дії або перевірки історії змін.

Окрему увагу під час розробки бази даних приділено запитам, які виконуються найчастіше. Для користувача найбільш типовими є перегляд операцій за періодом, фільтрація за категоріями, формування показників дашборда та отримання історії повідомлень у ШІ-чаті. Якщо такі запити виконувати без додаткової оптимізації, зі збільшенням кількості транзакцій час відповіді backend може зростати.

Для зменшення цього ризику в логічній моделі передбачено використання індексів за полями, що часто беруть участь у фільтрації. Насамперед це `user_id`, дата фінансової операції, тип категорії та час створення повідомлення. Такі індекси не змінюють логіку роботи системи, але допомагають швидше знаходити потрібні записи у таблицях PostgreSQL.

Приклади індексів, які доцільно використовувати для основних сценаріїв роботи системи, наведено в табл. 2.4.

Приклади індексів бази даних системи «FinAgent»

Індекс або група полів	Таблиця	Призначення
user_id, transaction_date	transactions	Швидкий пошук операцій конкретного користувача за вибраний період
user_id, type	categories	Фільтрація категорій за власником і типом операції
user_id, created_at	ai_chat_logs	Отримання історії діалогу користувача з ШІ-сервісом
user_id, day	chat_usage	Контроль кількості звернень до ШІ-сервісу за день

Наведені індекси пов'язані не з окремими екранами інтерфейсу, а з типовими серверними запитам. Наприклад, дашборд використовує вибірку транзакцій за користувачем і періодом, тому індекс за user_id та датою операції допомагає швидше сформувати агреговані показники. Для ШІ-чату важливим є отримання останніх повідомлень і контроль кількості звернень, тому окремо враховано таблиці ai_chat_logs та chat_usage.

2.3. Проєктування алгоритмів роботи системи та взаємодії модулів

Алгоритми роботи системи ґрунтуються на розділенні відповідальності між frontend, backend, ШІ-сервісом і базою даних. Для звичайних фінансових операцій користувач взаємодіє з формою або таблицею, frontend надсилає HTTP-запит із JWT, backend перевіряє права доступу, usecase виконує бізнес-правила, repository змінює PostgreSQL, після чого відповідь повертається клієнту. Такий сценарій застосовується для транзакцій, категорій, профілю, цілей та експорту. [16; 20].

Для ШІ-чату послідовність роботи складніша. Користувач вводить повідомлення природною мовою. Backend отримує його, перевіряє ліміт звернень і формує стислий контекст із доступних фінансових даних. Далі повідомлення разом із контекстом передається в ШІ-сервіс. Якщо модель відповідає на інформаційне питання, результат повертається користувачу. Якщо

ж модель визначає, що потрібно змінити дані, вона не виконує дію, а повертає структурований опис очікуваної дії [18; 19].

Очікувана дія зберігається на рівні відповіді як ``pendingAction``. Вона містить тип операції, основні параметри та текст підтвердження. Користувач бачить картку з описом і може прийняти або відхилити дію. У разі підтвердження frontend надсилає запит на виконання до backend. Backend повторно перевіряє дані, виконує дію через відповідний usecase та записує її в ``action_log``. Завдяки цьому користувач може скасувати останню дію, якщо вона була помилковою [18; 19].

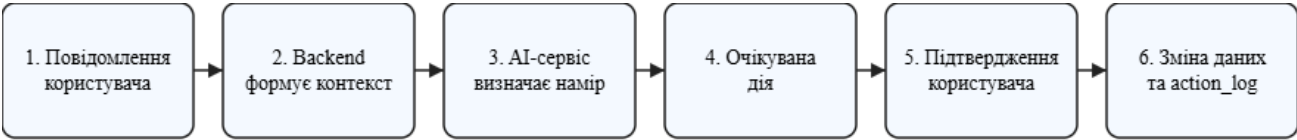
Алгоритм створення транзакції має кілька контрольних точок. Спочатку перевіряється автентифікація користувача, потім валідність суми, типу операції, дати та категорії. Якщо категорія є користувацькою, backend перевіряє її належність поточному користувачу. Після успішної перевірки repository створює запис у базі даних, а frontend оновлює список транзакцій і дашборд.

Алгоритм експорту CSV використовує ті самі фільтри, що й історія транзакцій. Це важливо, оскільки користувач очікує отримати у файлі саме ті дані, які бачить після застосування фільтрів. Крім того, експорт виключає м'яко видалені записи, щоб CSV не містив операцій, які користувач уже прибрав з основної історії [29].

Алгоритм виявлення нетипових витрат реалізований як запитна операція, а не як фоновий процес. Це означає, що система аналізує витрати тоді, коли користувач відкриває відповідну сторінку або віджет. Такий підхід простіший і відповідає обсягу бакалаврської роботи. Він не потребує окремого планувальника, push-повідомлень або email-інфраструктури, але дозволяє продемонструвати базову ідею пошуку витрат, які відрізняються від звичайного рівня.

Сценарій «що, якби» також має обмежений характер. Він не є фінансовим прогнозуванням, а виконує просту оцінку впливу запланованої покупки на місячний баланс користувача. Наприклад, якщо користувач питає, чи може він витратити певну суму, III-сервіс використовує доступний контекст і формує

пояснення. Це корисна функція для демонстрації природномовного інтерфейсу, але її не слід описувати як довгостроковий прогноз фінансової поведінки.



Дія не виконується без підтвердження, тому модель не має прямого доступу до зміни фінансових даних.

Рис. 2.3. Послідовність роботи ШІ-чату з підтвердженням дії

На рис. 2.3 наведено алгоритм роботи ШІ-чату. Основна ідея полягає в тому, що ШІ-сервіс виконує роль інтерпретатора намірів і генератора пояснень, але не має права самостійно змінювати фінансові записи. Такий підхід відповідає вимогам безпеки та підвищує довіру до системи, оскільки остаточне рішення завжди залишається за користувачем.

Таблиця 2.5

Основні алгоритми роботи системи

Алгоритм	Вхідні дані	Результат	Особливість реалізації
Реєстрація	email, пароль, ім'я	Новий користувач	Пароль хешується bcrypt
Створення транзакції	тип, сума, категорія, дата	Запис у transactions	Перевіряється власник категорії
Фільтрація історії	дати, тип, категорія, сторінка	Список транзакцій	limit/offset і total
Експорт CSV	фільтри транзакцій	CSV-файл	Виключаються soft-deleted записи
ШІ-чат	повідомлення користувача	відповідь або pendingAction	Дані не змінюються без підтвердження
Undo	останній запис action_log	відкат дії	create -> delete, delete -> restore
Аномалії	історія витрат	список нетипових витрат	Статистична оцінка за запитом

Табл. 2.5 показує, що більшість алгоритмів має чіткий вхід, результат і обмеження. Для системи важливо не лише виконати дію, але й забезпечити перевірку власника даних, коректне поводження з помилками та можливість

відновлення після помилкової ШІ-дії. Саме тому `action\_log` і soft-delete є частиною основної логіки, а не лише службовим доповненням.

Окремо варто виділити алгоритм ручного створення транзакції. У цьому сценарії frontend спочатку приймає дані з форми та виконує первинну перевірку обов'язкових полів. Після цього backend перевіряє JWT-токен, визначає користувача, перевіряє суму, тип операції, дату та категорію. Лише після проходження цих перевірок запис передається до repository-шару і зберігається у PostgreSQL.

Алгоритм побудови дашборда працює інакше, оскільки він не створює нові записи, а формує агреговану відповідь. Backend вибирає транзакції поточного користувача за визначений період, підраховує загальну суму доходів і витрат, визначає баланс, групує витрати за категоріями та повертає на frontend готовий набір даних для віджетів і графіків. Це дозволяє не переносити фінансові обчислення у клієнтську частину.

Під час обробки запиту до ШІ-чату система навмисно не передає мовній моделі повний вміст бази даних. Backend формує обмежений контекст, який містить лише ті дані, що потрібні для відповіді на конкретне питання. Такий підхід зменшує обсяг переданої інформації та робить взаємодію з ШІ-сервісом більш контрольованою.

Для кращого розуміння реалізації алгоритми доцільно розглядати не лише як загальні сценарії, а й як послідовність контрольних точок. Кожна така точка відповідає місцю, де система перевіряє коректність даних або обмежує доступ користувача до чужих записів.

Під час створення транзакції першою контрольною точкою є автентифікація. Backend отримує user_id з JWT-токена і надалі використовує його в усіх запитах до бази даних. Другою точкою є перевірка категорії: система не повинна дозволяти створити операцію з категорією іншого користувача. Третьою точкою є валідація суми, типу операції та дати.

У сценарії формування дашборда основна увага приділяється правильній агрегації. Backend не передає frontend повний набір фінансових записів для

самостійного підрахунку, а формує готові показники: суму доходів, суму витрат, поточний баланс, структуру витрат за категоріями та останні операції. Це зменшує дублювання бізнес-логіки на клієнтській стороні.

Для ШІ-сценаріїв контрольні точки мають ще більше значення. Система не повинна передавати мовній моделі зайві персональні дані та не повинна виконувати дію лише на підставі текстової відповіді. Тому backend формує обмежений контекст, ШІ-сервіс повертає відповідь або `pendingAction`, а основні контрольні точки цих сценаріїв наведено в табл. 2.6.

Таблиця 2.6

Контрольні точки основних алгоритмів системи

Сценарій	Контрольна точка	Мета перевірки
Створення транзакції	JWT, <code>user_id</code> , <code>category_id</code> , <code>amount</code>	Не допустити створення некоректної або чужої операції
Формування дашборда	Період, власник даних, типи операцій	Повернути тільки агреговані показники поточного користувача
ШІ-чат	Ліміт звернень, фінансовий контекст, тип запиту	Передати моделі лише потрібний мінімум даних
Підтвердження ШІ-дії	<code>pendingAction</code> , <code>user_id</code> , валідація <code>payload</code>	Виконати зміну лише після підтвердження користувача
Скасування дії	Останній запис <code>action_log</code>	Повернути систему до попереднього стану, якщо це можливо

У табл. 2.6 узагальнюються місця, де система перевіряє дані перед виконанням основних дій. Для фінансового застосунку це важливо, оскільки помилка в користувачі, сумі або категорії може вплинути на подальшу аналітику.

Такий підхід дозволяє описати алгоритми не тільки з погляду послідовності дій, а й з погляду безпеки. Для фінансового застосунку важливо, щоб кожен сценарій мав чітке місце перевірки даних, оскільки помилка в сумі, категорії або власнику запису може вплинути на коректність усієї подальшої аналітики.

2.4. Реалізація backend та бізнес-логіки

Backend реалізовано на Go з використанням Gin. Його основне призначення полягає в наданні REST API для frontend, перевірці автентифікації, виконанні бізнес-правил і роботі з PostgreSQL. Структура backend поділена на кілька шарів. Пакет `domain` містить сутності та інтерфейси. Пакет `repository` реалізує доступ до бази даних. Пакет `usecase` містить бізнес-логіку. Шар `delivery/http` відповідає за HTTP-обробники, маршрути й middleware [5; 6; 7; 16].

Такий поділ дозволяє уникнути ситуації, коли SQL-запити, перевірки прав і форматування HTTP-відповідей змішані в одному файлі. Наприклад, handler для транзакцій приймає запит і перетворює його на структуру вхідних даних, але не вирішує самостійно, чи має користувач право змінювати запис. Це виконує usecase, який перевіряє власника сутності та передає команду до repository. Repository відповідає тільки за SQL і мапінг результатів у структури Go.

Для автентифікації використовується JWT. Після успішного входу користувач отримує токен, який frontend додає до подальших запитів. Middleware перевіряє токен, витягує ідентифікатор користувача та додає його до контексту запиту. Завдяки цьому кожен usecase може працювати з конкретним `userID` і не покладатися на дані, які користувач передає вручну [20].

Паролі користувачів не зберігаються у відкритому вигляді. Перед записом у базу вони хешуються за допомогою bcrypt. Під час входу система порівнює введений пароль із хешем. Такий підхід відповідає загальним вимогам безпечного зберігання паролів і зменшує ризики у разі компрометації бази даних [26; 30].

Особливу роль у backend відіграє модуль скасування дій. Коли користувач підтверджує III-дію, backend не лише виконує її, а й створює запис у `action\_log`. Для створення транзакції в журналі зберігається інформація, яка дозволяє потім видалити або відновити запис. Для видалення використовується soft-delete, тому система може відновити дані без втрати контексту.

Інтеграція з Monobank реалізована як додатковий модуль. Вона не є обов'язковою для основного сценарію роботи системи, однак демонструє можливість розширення застосунку зовнішніми джерелами фінансових даних. Токен Monobank зберігається у зашифрованому вигляді, оскільки він надає доступ до банківської інформації користувача [4; 21].

Таблиця 2.7

Розподіл відповідальності backend-шарів

Шар	Призначення	Приклади
domain	Сутності та інтерфейси предметної області	User, Transaction, Goal, Repository interfaces
repository	SQL-запити та мапінг даних	TransactionRepository, StatsRepository
usecase	Бізнес-правила і перевірки	Create transaction, Undo, Contribute goal
delivery/http	HTTP-маршрути, handlers, middleware	Auth routes, ChatHandler, StatsHandler
pkg	Допоміжні функції	bcrypt, JWT, AES-GCM
ai/mono clients	Звернення до зовнішніх сервісів	AI client, Monobank client

Табл. 2.7 описує основні шари backend. Такий поділ має практичне значення: зміна способу зберігання даних не повинна вимагати переписування HTTP-обробників, а зміна інтерфейсу користувача не повинна впливати на бізнес-правила. Крім того, usecase легше покривати тестами, оскільки він не залежить безпосередньо від HTTP.

З практичної точки зору шарова структура backend також спрощує пошук помилок. Якщо проблема виникає на етапі приймання HTTP-запиту, її можна локалізувати в handler. Якщо помилка пов'язана з предметною логікою, наприклад із заборонаю доступу до чужої транзакції, вона належить до usecase-рівня. Якщо ж проблема стосується SQL-запиту або збереження даних, її область відповідальності обмежується repository-шаром.

Таке розділення важливе для подальшого розвитку системи. Наприклад, спосіб збереження даних або структура таблиць може змінюватися, але це не повинно впливати на формат HTTP-запитів, з якими працює frontend. Так само

зміна зовнішнього вигляду сторінки не повинна змінювати бізнес-правила backend.

Практична реалізація backend відображається у наборі REST endpoint-ів. Кожен endpoint відповідає окремому користувацькому сценарію або службовій дії. Для захищених маршрутів обов'язковою є перевірка JWT-токена, після чого у бізнес-логіку передається ідентифікатор поточного користувача.

Поділ endpoint-ів на групи дає змогу підтримувати зрозумілу структуру API. Наприклад, маршрути автентифікації не змішуються з фінансовими операціями, а маршрути ШІ-чату відокремлені від звичайних CRUD-запитів. Це спрощує тестування backend і дозволяє швидко знаходити місце помилки під час розробки. Основні endpoint-и backend API системи наведено в табл. 2.8.

Таблиця 2.8

Основні endpoint-и backend API системи «FinAgent»

Метод	Endpoint	Призначення
POST	/api/auth/register	Реєстрація нового користувача
POST	/api/auth/login	Авторизація та отримання JWT-токена
GET	/api/transactions	Отримання списку транзакцій користувача
POST	/api/transactions	Створення нової фінансової операції
PUT	/api/transactions/{id}	Редагування існуючої транзакції
DELETE	/api/transactions/{id}	М'яке видалення транзакції
GET	/api/dashboard	Отримання агрегованих показників для дашборда
POST	/api/chat	Надсилання повідомлення до ШІ-чату
POST	/api/chat/confirm	Підтвердження дії, запропонованої ШІ-сервісом

Endpoint-и в наведеній таблиці відображають основний робочий контур застосунку. Frontend не звертається напряму до PostgreSQL або ШІ-сервісу, а працює через backend API. Завдяки цьому всі перевірки доступу, валідація даних і виконання змін залишаються в одному контрольованому місці.

2.5. Реалізація III-сервісу та клієнтської частини

III-сервіс реалізовано як окремий FastAPI-застосунок. Він має кілька основних endpoint-ів: чат, категоризація та аудит нетипових витрат. Центральним сценарієм є чат, у якому користувач ставить питання про свої фінансові дані або просить підготувати дію. Сервіс працює з обмеженим контекстом, який передає backend, тому модель не має прямого доступу до всієї бази даних [9; 18; 19].

У чаті використовується підхід виклику інструментів моделлю. Це означає, що модель не просто генерує довільний текст, а може вибрати один із наперед описаних інструментів. Частина інструментів є інформаційними: вони допомагають отримати баланс, найбільші категорії витрат, прогрес цілей або результат простого сценарію «що, якби». Інша частина інструментів описує потенційні дії, але не виконує їх напряду [18].

Категоризація за описом транзакції у поточному стані проєкту реалізована як експериментальний endpoint. Backend має клієнт для звернення до цього endpoint, але повний користувацький сценарій не є центральним для інтерфейсу. Тому в документації та пояснювальній записці цю функцію доцільно подавати як додатковий модуль, а не як основну можливість системи.

Клієнтська частина на React забезпечує взаємодію з усіма основними функціями: сторінками входу й реєстрації, профілем, транзакціями, категоріями, фінансовими цілями, інформаційною панеллю, сторінкою нетипових витрат та III-чатом. TypeScript використовується для опису типів даних, що передаються між frontend і backend. Це зменшує ризик помилок під час роботи з відповідями API [10; 11].

У дашборді користувач бачить зведені показники, графіки та останні операції. Візуалізація допомагає швидко оцінити фінансовий стан без переходу до окремих таблиць. Налаштовуване розташування віджетів є додатковою можливістю, тому в основному описі воно не розглядається як ключова вимога. Головним є те, що дашборд показує користувачу агреговану інформацію в зрозумілому вигляді [23].

Інтерфейс чату побудований так, щоб користувач бачив результат роботи системи у звичайній формі. Якщо ШІ повертає текстову відповідь, вона відображається як повідомлення. Якщо ШІ пропонує дію, наприклад додати витрату, інтерфейс показує картку підтвердження. Це зменшує ризик помилкового виконання і дає користувачу зрозуміти, які саме дані будуть змінені.

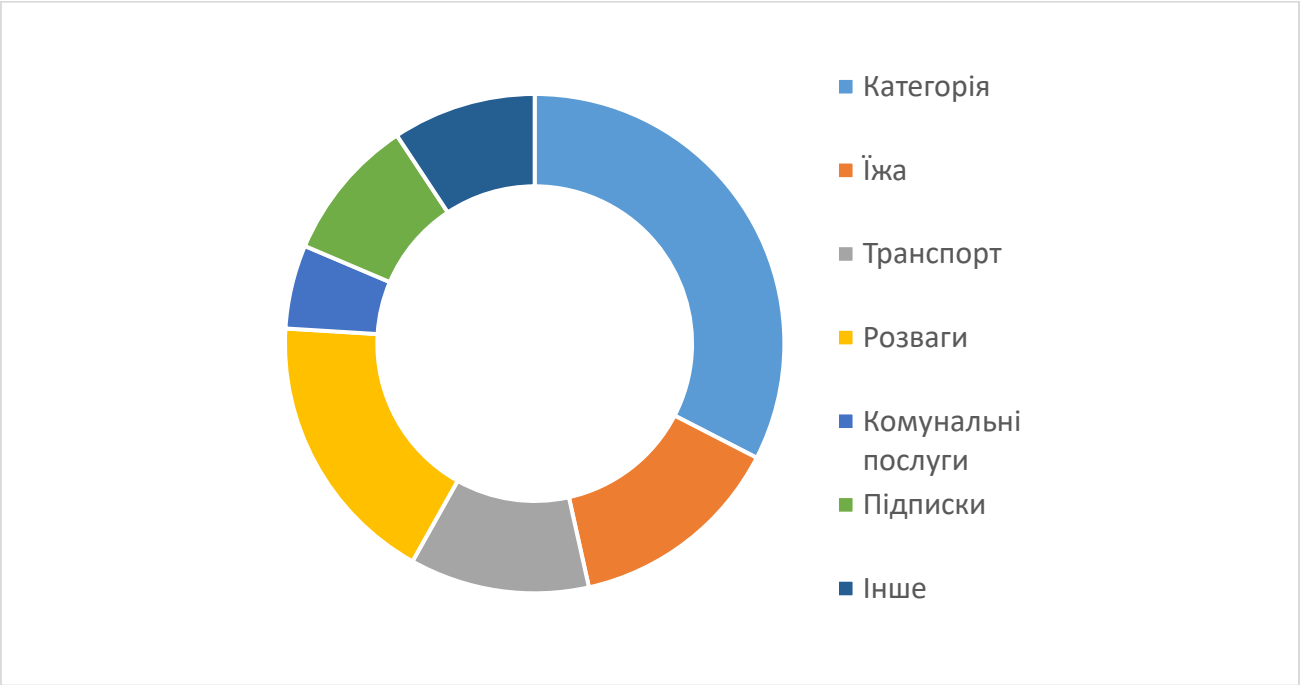


Рис. 2.4. Приклад візуалізації розподілу витрат за категоріями

На рис. 2.4 наведено приклад графічного подання розподілу витрат. У реалізованій системі такі дані формуються backend на основі транзакцій користувача, після чого frontend відображає їх у вигляді графіка. Це демонструє взаємодію трьох рівнів: база даних зберігає операції, backend обчислює агрегати, а frontend перетворює результат у зрозумілу візуалізацію, яка легко сприймається користувачем.

Таблиця 2.9

Основні frontend-сторінки системи

Сторінка	Призначення	Основні дії користувача
Auth pages	Реєстрація та вхід	Створити обліковий запис, увійти
ProfilePage	Профіль користувача	Переглянути та змінити персональні дані

TransactionsPage	Історія транзакцій	Фільтрувати, створювати, редагувати, експортувати
CategoriesPage	Категорії	Створити або змінити власні категорії
DashboardPage	Зведена статистика	Переглянути графіки й останні операції
GoalsPage	Фінансові цілі	Створити ціль, поповнити або зняти кошти
ChatPanel	ШІ-чат	Поставити питання, підтвердити або відхилити дію
AnomaliesPage	Нетипові витрати	Переглянути попередження про незвичні витрати

Табл. 2.9 показує, що frontend охоплює основні сценарії користувача. Інтерфейс не повинен вимагати від користувача знання внутрішньої архітектури. Наприклад, у чаті користувач бачить звичайну відповідь або картку підтвердження, тоді як реальна взаємодія між frontend, backend і ШІ-сервісом залишається прихованою.

Додатково в межах реалізації було сформовано набір REST endpoint-ів, які відображають основні дії користувача. Такий набір не випадковий: він відповідає функціональним вимогам, визначеним у п. 2.1, і повторює логіку сторінок frontend. Наприклад, сторінка транзакцій використовує endpoint-и для створення, отримання, оновлення, видалення та експорту операцій, тоді як дашборд звертається до статистичних endpoint-ів.

REST API спроектовано так, щоб frontend не мав прямого доступу до бази даних. Усі запити проходять через backend, де виконується автентифікація, перевірка прав доступу, валідація вхідних даних і перетворення доменних помилок у HTTP-відповіді. Це дозволяє зберегти контроль над бізнес-правилами та не дублювати логіку в клієнтській частині [16; 20; 21].

Основні групи REST endpoint-ів системи

Група API	Приклади endpoint-ів	Призначення
Auth	POST /api/auth/register, POST /api/auth/login	Реєстрація, вхід, отримання JWT
Users	GET /api/users/me, PUT /api/users/me	Перегляд і редагування профілю
Transactions	GET/POST /api/transactions, PUT/DELETE /:id	Робота з доходами й витратами
Categories	GET/POST /api/categories, PUT/DELETE /:id	Керування системними та власними категоріями
Stats	GET /api/stats/summary, /dashboard	Зведення, графіки, дашборд
Goals	GET/POST /api/goals, POST /:id/contribute	Фінансові цілі та зміна прогресу
Chat	POST /api/chat, POST /api/actions/execute	ШІ-чат і підтвердження дій
Anomalies	GET /api/anomalies	Показ нетипових витрат

У табл. 2.10 наведено основні групи API. Така структура полегшує розуміння взаємодії між frontend і backend. Кожна група endpoint-ів відповідає окремій предметній сутності або сценарію роботи. Це також допомагає під час тестування, оскільки для кожної групи можна підготувати контрольні запити та очікувані результати.

Обробка помилок у backend має єдиний принцип. Якщо usecase повертає доменну помилку, HTTP-шар перетворює її у відповідний статус: помилка валідації повертає відповідь клієнту з поясненням, відсутність сутності відповідає статусу «не знайдено», а спроба доступу до чужих даних розглядається як заборонена операція. Завдяки цьому frontend отримує передбачуваний формат помилок [16].

Валідація даних виконується на кількох рівнях. Frontend може не дозволити користувачу надіслати очевидно некоректну форму, але остаточна перевірка завжди залишається на backend. Це важливо, оскільки користувач або

сторонній клієнт може звернутися до API напяму. Тому backend не повинен довіряти лише перевіркам інтерфейсу [21].

Для ШІ-сервісу було розділено інструменти читання та інструменти дій. Інструменти читання не змінюють дані й використовуються для формування відповідей про баланс, категорії або цілі. Інструменти дій, навпаки, описують потенційну зміну даних, але повертають її до backend як очікувану дію. Це розділення робить ШІ-модуль зрозумілішим і безпечнішим [18; 19].

Таблиця 2.11

Групи інструментів ШІ-сервісу

Група інструментів	Приклади дій	Роль у системі
Інформаційні інструменти	Показ балансу, топ категорій, прогрес цілей	Формують відповідь без зміни даних
Сценарні інструменти	Оцінка впливу покупки на баланс	Дають просте пояснення «що, якби»
Інструменти дій	Створення транзакції, зміна цілі	Повертають pendingAction для підтвердження
Аудит витрат	Пошук нетипових операцій	Допомагають показати попередження користувачу
Категоризація	Пропозиція категорії за описом	Додатковий експериментальний модуль

Табл. 2.11 показує, що ШІ-сервіс не є одним великим універсальним алгоритмом. Його роботу поділено на групи інструментів, кожна з яких має чітке призначення.

Об'єкт pendingAction у цій архітектурі не розглядається як команда, що виконується автоматично. Це проміжний опис дії, який містить тип операції, основні параметри та текст пояснення для користувача. Backend додатково перевіряє такий об'єкт перед виконанням, оскільки остаточне рішення про зміну фінансових даних не може залежати лише від відповіді мовної моделі.

Щоб зробити цей механізм прозорим, відповідь ШІ-сервісу для дій, які можуть змінити дані, подається у структурованому форматі. У такому об'єкті зберігається тип дії, сутність, набір параметрів і ознака необхідності

підтвердження. Backend не виконує цю структуру автоматично, а спочатку перевіряє її зміст.

Наявність окремого поля `requiresConfirmation` є принциповою для предметної області персональних фінансів. Воно показує, що дія не є звичайною текстовою відповіддю, а потребує явної реакції користувача. Якщо користувач відхиляє пропозицію, зміни до бази даних не вносяться. Структуру об'єкта `pendingAction`, який використовується для контрольованих ШІ-дій, наведено в табл. 2.12.

Таблиця 2.12

Структура об'єкта `pendingAction`

Поле	Призначення
<code>type</code>	Тип запропонованої дії, наприклад створення транзакції або зміна цілі
<code>entity</code>	Сутність, до якої належить дія
<code>payload</code>	Дані, які потрібно перевірити та передати в backend-логіку
<code>message</code>	Текст пояснення, який бачить користувач перед підтвердженням
<code>requiresConfirmation</code>	Ознака того, що дія може бути виконана лише після підтвердження

Наприклад, якщо користувач у чаті просить додати витрату, ШІ-сервіс може сформувати `pendingAction` зі значенням `type = create_transaction`. У `payload` передаються сума, тип операції, опис і категорія. Після цього frontend відображає картку підтвердження, а backend виконує запис лише після позитивної відповіді користувача.

Таким чином, `pendingAction` є проміжною структурою між природномовним запитом і реальною зміною даних. Саме ця структура дозволяє поєднати зручність ШІ-чату з контрольованістю традиційного backend API.

У клієнтській частині окрему роль відіграє керування станом. Інтерфейс повинен знати, чи користувач авторизований, які транзакції завантажено, які фільтри застосовано, чи є активна очікувана ШІ-дія та які дані потрібно оновити після виконання операції. Тому стан розділяється за логічними зонами: автентифікація, фінансові дані, тема інтерфейсу, чат і дашборд.

Під час реалізації frontend важливо не дублювати бізнес-правила backend. Наприклад, сторінка може приховати кнопку редагування системної категорії, але справжня заборона має бути реалізована на сервері. Такий підхід захищає систему від прямого звернення до API та забезпечує однакову поведінку незалежно від того, через який клієнт зроблено запит.

Підсумовуючи реалізацію, можна зазначити, що програмний комплекс складається з кількох взаємопов'язаних, але незалежних за відповідальністю частин. Це дозволяє розглядати систему не як набір випадкових сторінок, а як цілісний програмний продукт із чіткою архітектурою, визначеними потоками даних і контрольованими місцями зміни фінансової інформації. Зв'язок основного функціоналу з програмними модулями системи подано в табл. 2.13.

Таблиця 2.13

Зв'язок функціоналу з програмними модулями

Функціонал	Backend	Frontend	III-сервіс
Автентифікація	user handler, auth package	Auth pages	Не використовується
Транзакції	transaction usecase/repository	TransactionsPage	Може пропонувати дію
Категорії	category usecase/repository	CategoriesPage	Додаткова категоризація
Дашборд	stats/dashboard handlers	DashboardPage, widgets	Пояснення в чаті
Фінансові цілі	goal usecase/repository	GoalsPage	Може аналізувати прогрес
III-чат	chat handler, ai client	ChatPanel	chat.py, tools.py
Undo	action usecase, action_log	кнопка скасування	Не виконує напряму
Аномалії	audit handler	AnomaliesPage	auditor.py

Під час проектування було також враховано питання журналювання і повторюваності дій користувача. Для звичайного CRUD-застосунку достатньо створити або видалити запис, однак у системі зі III-асистентом важливо мати можливість пояснити, чому певна дія була виконана і як її скасувати. Саме тому підтверджені III-дії записуються окремо. Це не перетворює систему на повноцінну систему аудиту, але дозволяє реалізувати практичний механізм undo.

Під час роботи з користувацькими даними важливо не змішувати дані різних облікових записів. У кожному запиті, який стосується транзакцій, категорій або цілей, backend використовує ідентифікатор поточного користувача з JWT-контексту. Це означає, що навіть якщо користувач спробує передати ідентифікатор чужої транзакції, usecase повинен перевірити належність запису і заборонити зміну.

Для frontend важливо підтримувати узгодженість стану після кожної зміни. Після створення транзакції потрібно оновити не лише таблицю операцій, а й дашборд, графіки, останні операції та потенційно сторінку нетипових витрат. У невеликому прототипі частина цих оновлень може виконуватися повторним запитом до API, що є простішим і надійнішим рішенням, ніж складне ручне оновлення всіх локальних структур.

Під час реалізації ШІ-відповідей було прийнято рішення не подавати модель як джерело фінансових гарантій. Відповіді мають пояснювальний характер і базуються на даних, які backend передав у контексті. Це обмежує можливості асистента, але робить поведінку системи чеснішою та простішою для захисту. Якщо в майбутньому потрібно буде розширити точність рекомендацій, це можна зробити окремим алгоритмічним модулем.

Таблиця 2.14

Інженерні рішення, що зменшують ризики системи

Рішення	Який ризик зменшує	Пояснення
Перевірка userID	Доступ до чужих даних	Запити виконуються в контексті поточного користувача
pendingAction	Неконтрольована ШІ-дія	ШІ лише пропонує дію, а не виконує її напяму
action_log	Неможливість відкотити помилку	Підтверджена дія може бути скасована
soft-delete	Втрата даних після видалення	Запис позначається видаленням, але не стирається одразу
JSONB для службових даних	Зайве ускладнення схеми	Гнучкі відповіді ШІ і layout дашборда не потребують окремих таблиць
Docker Compose	Проблеми запуску на іншому ПК	Середовище запуску описано у конфігурації

У табл. 2.14 узагальнено рішення, які мають не декоративний, а практичний характер. Вони безпосередньо пов'язані з ризиками, що виникають у системі персональних фінансів: несанкціонований доступ, помилкова дія, втрата даних, складність запуску та складність пояснення ІІІ-логіки.

Таким чином, реалізація «FinAgent» демонструє не лише набір функцій, а й послідовне застосування інженерних рішень. Для бакалаврської роботи це важливо, оскільки оцінюється не тільки факт наявності сторінок у браузері, але й здатність пояснити, чому обрані саме такі компоненти, як вони взаємодіють і які обмеження має розроблений прототип.

Висновки до розділу 2

У другому розділі було виконано проєктування та опис реалізації системи «FinAgent». Визначено ролі користувачів, основні функціональні та нефункціональні вимоги, а також сценарії використання, що відповідають задачам персонального фінансового обліку зі ІІІ-асистентом.

Розроблено структуру бази даних, у якій центральною сутністю є користувач, а транзакції, категорії, фінансові цілі, історія чату й журнал дій зв'язані з ним через зовнішні ключі. Описано використання PostgreSQL, JSONB, soft-delete та action_log для підтримки безпечної роботи з даними й скасування підтверджених ІІІ-дій.

Описано алгоритми роботи системи, зокрема звичайні CRUD-сценарії, ІІІ-чат, сценарій підтвердження дії, undo та виявлення нетипових витрат. Реалізація backend, ІІІ-сервісу та frontend побудована модульно, що дозволяє підтримувати систему, тестувати окремі частини та чітко пояснювати архітектуру під час захисту.

РОЗДІЛ 3. ІНТЕРФЕЙС ТА ТЕСТУВАННЯ СИСТЕМИ УПРАВЛІННЯ ФІНАНСАМИ

3.1. Порядок встановлення та запуску системи

Розгортання програмного комплексу «FinAgent» передбачено у двох варіантах: локально в середовищі розробника та на віддаленому сервері. Для локального запуску використовується Docker Compose. Це дозволяє не встановлювати окремо середовища виконання, зовнішні бібліотеки та СУБД, а запустити клієнтську частину, серверне ядро, ШІ-сервіс і базу даних PostgreSQL однією командою. Після запуску контейнери працюють у межах спільної Docker-мережі, тому компоненти можуть звертатися один до одного за системними іменами сервісів [12; 13].

Для швидкого тестування та демонстрації функціоналу робочу версію системи також розгорнуто на віддаленому сервері з публічним доменом <https://finagent-mpd-zipz-221.xyz>. Завдяки цьому вебзастосунок можна відкрити безпосередньо через браузер без локального збирання коду. Такий підхід зручний для демонстрації основних користувацьких сценаріїв: створення облікового запису, проходження автентифікації, ведення обліку доходів і витрат, створення фінансових цілей, перегляду аналітики на дашборді та роботи з ШІ-чатом.

Конфігурація системи здійснюється через змінні оточення. В окремому локальному файлі зберігаються параметри підключення до PostgreSQL, секретний ключ для підпису JWT-токенів, адреса ШІ-сервісу, службові токени для міжсервісних запитів та API-ключ для взаємодії із зовнішньою мовною моделлю. Щоб не допустити випадкового потрапляння цих даних до репозиторію, конфігураційний файл додано до правил ігнорування Git. Тому реальні паролі, ключі доступу та токени залишаються в середовищі розгортання і не публікуються разом із вихідним кодом проєкту [28].

Загальна структура середовища запуску наведена в табл. 3.1.

					ІПЗ.КР.Б–121–26-ІЗ	50

Основні компоненти середовища запуску системи «FinAgent»

Компонент	Технологія	Призначення	Приклад адреси або ролі
Frontend	React, TypeScript, Vite	Користувацький інтерфейс у браузері	http://localhost
Backend	Go, Gin	REST API, автентифікація, бізнес-логіка	http://localhost:8080
ШІ-сервіс	Python, FastAPI	Обробка повідомлень чату та формування ШІ-відповідей	внутрішній сервіс Docker
База даних	PostgreSQL	Зберігання користувачів, транзакцій, категорій, цілей і журналів	внутрішня мережа Docker
Docker Compose	docker-compose.yml	Одночасний запуск усіх компонентів	docker compose up -d --build

Після налаштування змінних середовища система запускається командою.

Лістинг команди наведено нижче:

```
docker compose up -d --build
```

Лістинг команди для перевірки стану контейнерів:

```
docker compose ps
```

На рис. 3.1 показано результат перевірки запущених контейнерів. У списку мають відображатися основні сервіси системи: frontend, backend, ШІ-сервіс та база даних PostgreSQL. Наявність активного стану контейнерів підтверджує, що середовище розгортання сформовано коректно, а компоненти системи готові до взаємодії між собою у межах спільної Docker-мережі.

Окремо важливо перевірити доступність клієнтської частини. Для цього після запуску контейнерів у браузері відкривається адреса frontend-сервісу або публічний домен системи. Якщо сторінка завантажується без помилок, це свідчить про успішний запуск вебінтерфейсу та можливість подальшої роботи з основними функціями застосунку.

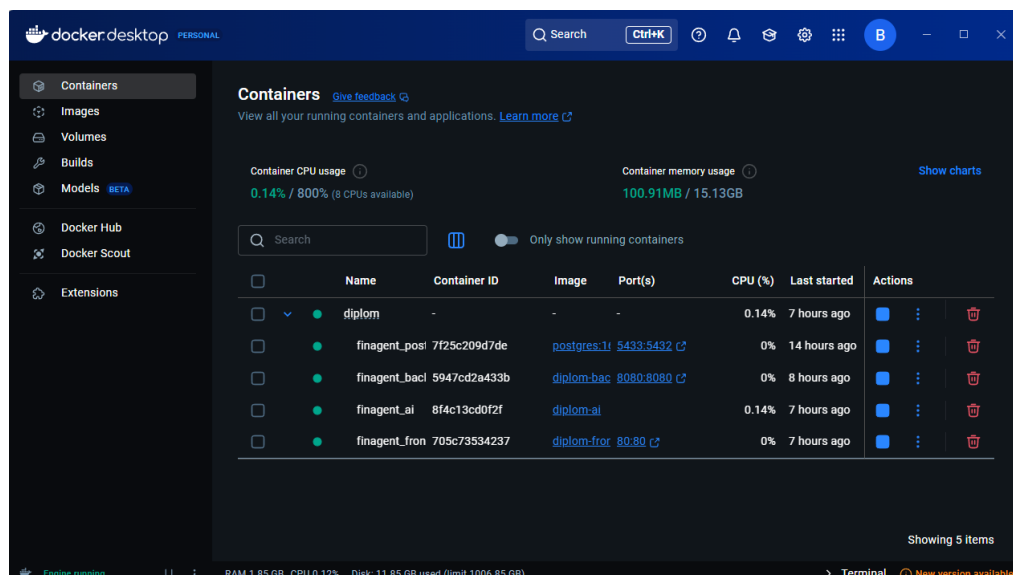


Рис. 3.1. Результат запуску контейнерів системи «FinAgent»

Після успішного запуску frontend відкривається у браузері. Початкова сторінка або сторінка входу підтверджує, що клієнтська частина коректно підключена до backend і готова до роботи користувача. Приклад початкової сторінки системи після запуску наведено на рис. 3.2.

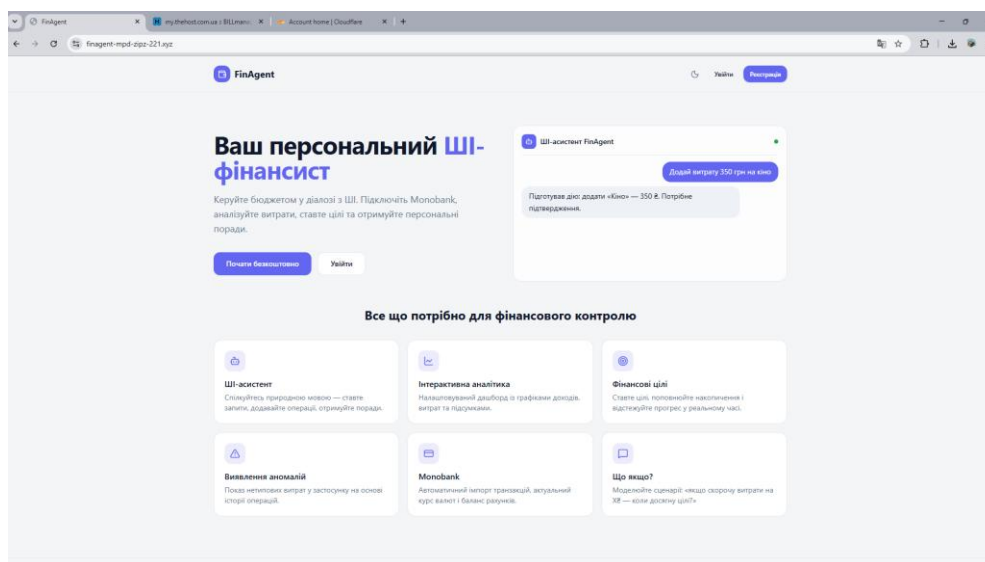


Рис. 3.2. Початкова сторінка системи після запуску

Таким чином, порядок запуску системи складається з трьох основних кроків: налаштування змінних середовища, запуску контейнерів через Docker Compose та перевірки доступності frontend у браузері. Такий спосіб зручний для демонстрації на захисті, оскільки дозволяє швидко відтворити робоче середовище без ручного встановлення кожного сервісу окремо.

3.2. Структура інтерфейсу та демонстрація роботи системи

Інтерфейс системи «FinAgent» побудовано як односторінковий вебзастосунок. Користувач взаємодіє з системою через браузер, а всі запити до даних виконуються через backend API. Основні сторінки інтерфейсу відповідають ключовим сценаріям роботи: реєстрація, вхід, перегляд dashboard, керування транзакціями, категоріями, фінансовими цілями та взаємодія з ШІ-чатом.

Першим сценарієм роботи є створення облікового запису. На сторінці реєстрації користувач вводить ім'я, адресу електронної пошти та пароль. Після відправлення форми backend перевіряє коректність даних, хешує пароль і створює запис користувача у базі даних.

Після реєстрації користувач може перейти до входу в систему. Форма входу приймає email і пароль. У разі успішної автентифікації backend повертає токен доступу, який надалі використовується frontend для звернення до захищених endpoint-ів.

Після входу користувач потрапляє на головну інформаційну панель. Dashboard відображає основні фінансові показники: поточний баланс, суму доходів, суму витрат, останні операції та графічні віджети. Такий екран потрібний для швидкої оцінки фінансового стану без ручного перегляду повної історії транзакцій. Приклад головної панелі користувача наведено на рис. 3.3.

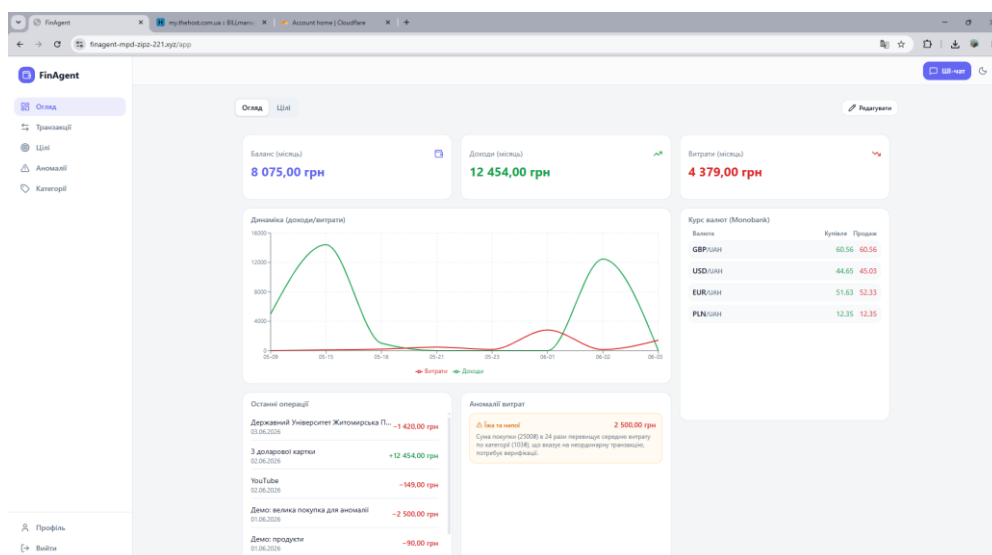


Рис. 3.3. Головна панель користувача

На рис. 3.3 показано центральний екран особистого кабінету користувача. Він об'єднує основні фінансові показники та дає змогу швидко оцінити загальний стан бюджету. Саме з цього екрана користувач може перейти до перегляду транзакцій, фінансових цілей, категорій або взаємодії з ШІ-асистентом.

Одним із важливих елементів dashboard є візуалізація витрат за категоріями. Вона дозволяє побачити, які напрями витрат займають найбільшу частку у бюджеті за обраний період. Наприклад, користувач може швидко помітити, що найбільше коштів витрачається на їжу, транспорт або розваги. Приклад такої візуалізації наведено на рис. 3.4.

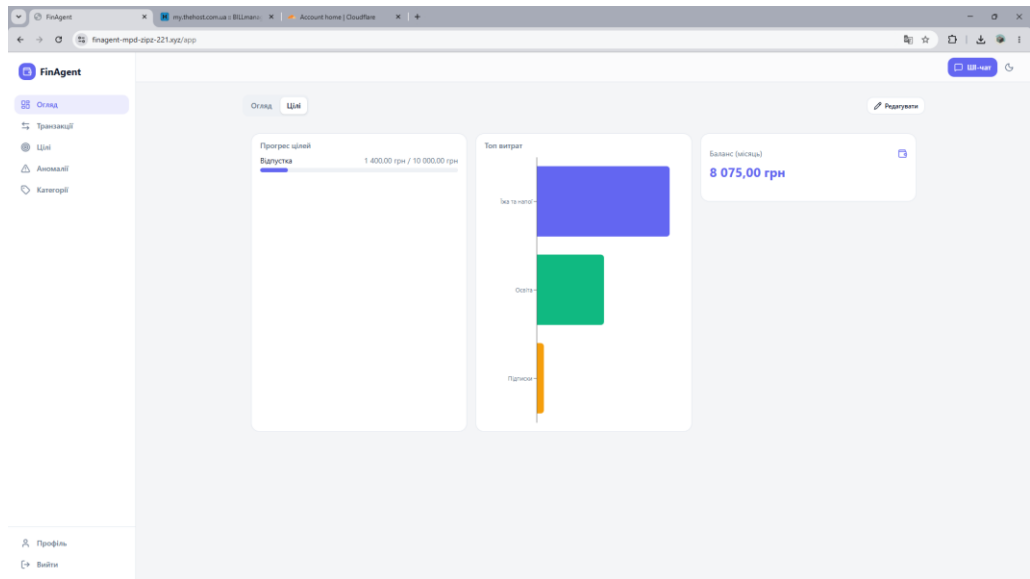


Рис. 3.4. Приклад візуалізації розподілу витрат за категоріями

Рис. 3.4 демонструє графічне подання фінансової аналітики. Завдяки такому відображенню користувачеві не потрібно самотійно аналізувати всі записи у таблиці транзакцій, оскільки система автоматично групує витрати за категоріями та подає їх у наочному вигляді.

Основним інструментом фінансового обліку є сторінка транзакцій. На ній користувач бачить список доходів і витрат із датою, сумою, категорією, типом операції та описом. Для зручності роботи можуть використовуватися фільтри за датою, типом або категорією, а також пагінація для великих списків. Зовнішній вигляд сторінки перегляду фінансових операцій показано на рис. 3.5.

Дата	Опис	Категорія	Тип	Сума
Червень 2026 р.				+8 075,00 грн
03.06.2026	Державний Університет Житомирська Політехніка	Освіта	Витрати	-1 420,00 грн
02.06.2026	3 доларової картки	Переклади	Дохід	+12 454,00 грн
02.06.2026	YouTube	Підписки	Витрати	-149,00 грн
01.06.2026	Демо: велика покупка для аномалії	Інка та напої	Витрати	-2 500,00 грн
01.06.2026	Демо: продукти	Інка та напої	Витрати	-90,00 грн
01.06.2026	Демо: обід	Інка та напої	Витрати	-120,00 грн
01.06.2026	Демо: кави та перекус	Інка та напої	Витрати	-100,00 грн
Травень 2026 р.				+19 466,42 грн
23.05.2026	Google	Підписки	Витрати	-6,00 грн
23.05.2026	WayForPay	Інше	Витрати	-162,00 грн
21.05.2026	Steam	Розваги	Витрати	-180,00 грн
21.05.2026	Base Kulturyi Didko	Переклади	Витрати	-300,00 грн
16.05.2026	Тест	Зарплата	Дохід	+12,00 грн
16.05.2026	Тест	Підробіток	Дохід	+1 000,00 грн

Рис. 3.5. Сторінка перегляду фінансових операцій

На рис. 3.5 відображено список фінансових операцій користувача. Такий інтерфейс дозволяє контролювати історію доходів і витрат, швидко знаходити потрібні записи та перевіряти правильність внесених даних.

Додавання нової операції виконується через окрему форму. Користувач обирає тип операції, вказує суму, дату, категорію та короткий опис. Після натискання кнопки збереження frontend передає дані на backend, де вони проходять перевірку і зберігаються у таблиці транзакцій.

Редагувати транзакцію

Тип: Дохід Сума (грн):

Категорія: Зарплата

Опис:

Дата:

Скасувати Зберегти

Рис. 3.6. Форма додавання фінансової операції

Рис. 3.6 показує форму, через яку користувач вручну додає новий дохід або витрату. Наявність окремих полів для суми, дати, категорії та опису дозволяє зберігати транзакції у структурованому вигляді, що надалі використовується для фільтрації та побудови аналітики.

Категорії використовуються для групування доходів і витрат. У системі можуть бути системні категорії, доступні всім користувачам, і власні категорії, створені конкретним користувачем. Такий підхід дозволяє одразу мати базовий набір категорій і водночас налаштувати облік під особисті потреби.

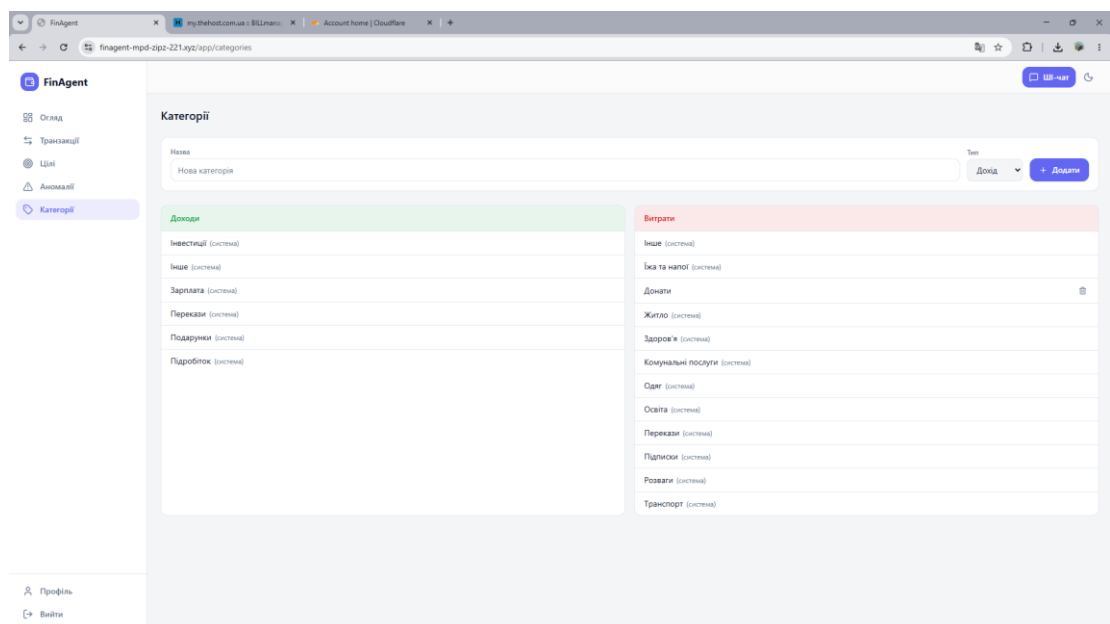


Рис. 3.7. Сторінка керування категоріями

На рис. 3.7 продемонстровано інтерфейс роботи з категоріями. Цей екран потрібний для налаштування логіки групування фінансових операцій, оскільки саме категорії визначають, як система буде розподіляти витрати та доходи в аналітичних віджетах.

Окремим функціональним блоком є фінансові цілі. Користувач може створити ціль, вказати бажану суму, поточний прогрес і термін досягнення. Система відображає стан накопичення та дозволяє оцінити, наскільки користувач наблизився до запланованого результату. Приклад сторінки фінансових цілей наведено на рис. 3.8.

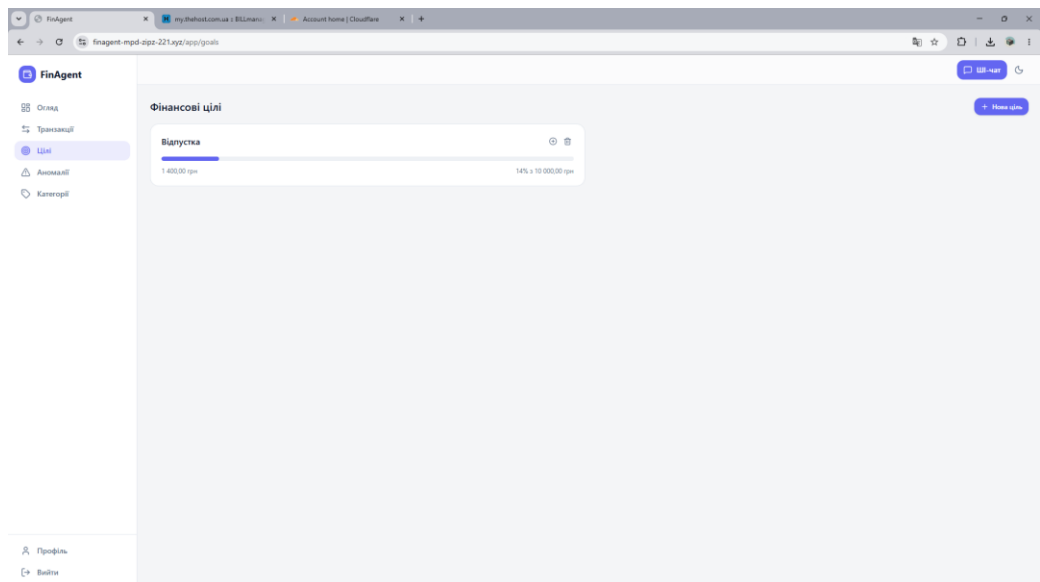


Рис. 3.8. Сторінка фінансових цілей користувача

Рис. 3.8 показує, як у системі реалізовано планування накопичень. Користувач бачить перелік власних фінансових цілей і може оцінити прогрес їх виконання. Це перетворює застосунок не лише на інструмент обліку минулих витрат, а й на засіб планування майбутніх фінансових рішень.

Найбільш характерною частиною «FinAgent» є ШІ-чат. Через нього користувач може ставити питання природною мовою. Наприклад, він може запитати, скільки було витрачено на певну категорію за місяць, які витрати були нетиповими або як запланована покупка вплине на поточний баланс.

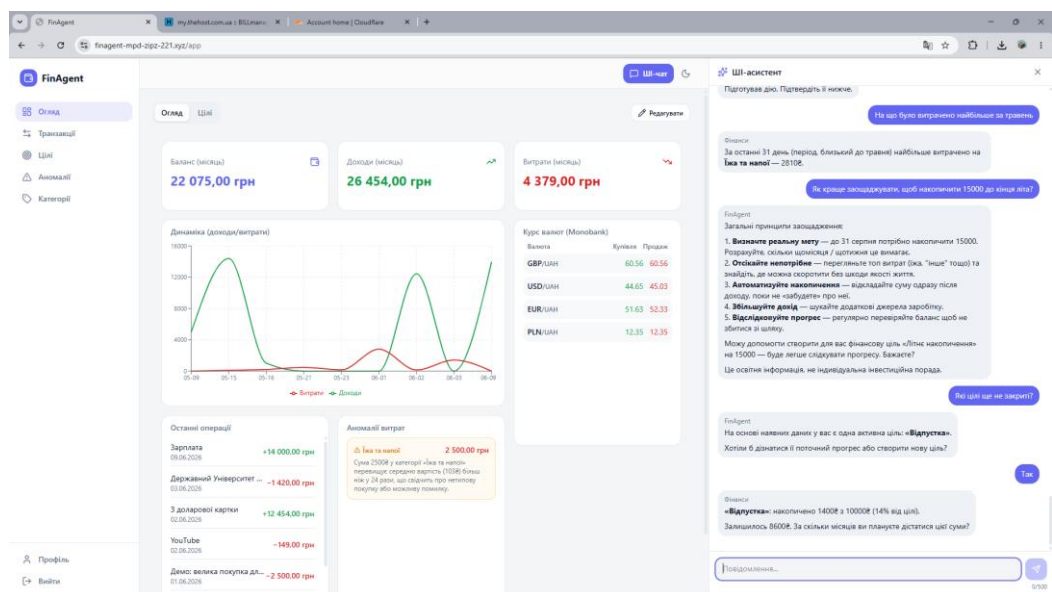


Рис. 3.9. Інтерфейс ШІ-чату системи «FinAgent»

На рис. 3.9 наведено діалогове вікно взаємодії з ШІ-асистентом. Цей екран відрізняє систему від звичайних фінансових трекерів, оскільки користувач може отримувати відповіді та аналітичні пояснення у форматі звичайного текстового спілкування.

Важливо, що ШІ-сервіс не змінює фінансові дані напряму. Якщо повідомлення користувача містить намір створити або змінити запис, система формує очікувану дію. Після цього frontend показує користувачу картку підтвердження з деталями майбутньої операції. Лише після натискання кнопки підтвердження backend виконує зміну в базі даних.

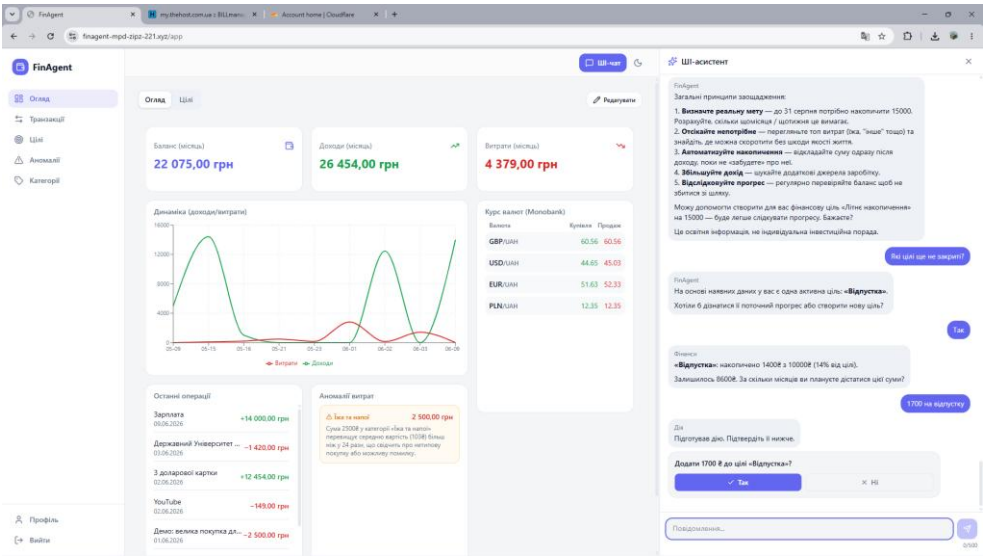


Рис. 3.10. Підтвердження дії, запропонованої ШІ-сервісом

Рис. 3.10 демонструє ключовий механізм безпеки системи – підтвердження дії перед її виконанням. Завдяки цьому ШІ-сервіс не має можливості самостійно змінювати фінансові записи. Користувач бачить зміст запропонованої операції та самостійно вирішує, чи потрібно її виконувати.

Після підтвердження запропонована дія виконується на backend і результат відображається в інтерфейсі. Наприклад, якщо користувач підтвердив створення витрати, новий запис з'являється у списку транзакцій. Такий сценарій демонструє контрольовану взаємодію між ШІ-сервісом і фінансовим ядром системи.

Додатково система підтримує експорт фінансових даних у CSV. Цей сценарій корисний для користувачів, які хочуть зберегти історію операцій окремим файлом або проаналізувати її в табличному редакторі [29].

	A	B	C	D	E	F	G	H	I	J
1	Дата	Опис	Категорія	Тип	Сума, грн					
2	09.06.2026	Зарплата	Зарплата	Дохід	14000					
3	03.06.2026	Державний Університет Житомирська Політехніка	Освіта	Витрата	1420					
4	02.06.2026	3 доларової картки	Перекази	Дохід	12454					
5	02.06.2026	YouTube	Підписки	Витрата	149					
6	01.06.2026	Демо: велика покупка для аномалії	Їжа та напої	Витрата	2500					
7	01.06.2026	Демо: продукти	Їжа та напої	Витрата	90					
8	01.06.2026	Демо: обід	Їжа та напої	Витрата	120					
9	01.06.2026	Демо: кава та перекус	Їжа та напої	Витрата	100					
10	23.05.2026	Google	Підписки	Витрата	6					
11	23.05.2026	WayForPay	Інше	Витрата	162					
12	21.05.2026	Steam	Розваги	Витрата	180					
13	21.05.2026	Base Kulturnyi Didko	Перекази	Витрата	300					
14	16.05.2026	Тест	Зарплата	Дохід	12					
15	16.05.2026	Тест	Підробіток	Дохід	1000					
16	16.05.2026	Тест	Комунальні послуги	Витрата	100					
17	16.05.2026	Spotify	Підписки	Витрата	110,58					
18	15.05.2026	NIC.UA	Підписки	Витрата	1					
19	15.05.2026	Скасування, NIC.UA	Інше	Дохід	1					
20	15.05.2026	TheHost.ua6	Підписки	Витрата	99					
21	15.05.2026	Тест	Інвестиції	Дохід	14412					
22	09.05.2026	Від: Анастасія Д. — День Народження т.Лєни	Подарунки	Дохід	5000					
23										
24										
25										
26										
27										
28										
29										

Рис. 3.11. Експорт фінансових даних у CSV

На рис. 3.11 показано результат експорту фінансової інформації у CSV-файл. Ця функція підтверджує, що система дозволяє не лише переглядати дані всередині вебінтерфейсу, а й вивантажувати їх для подальшого зберігання або зовнішнього аналізу.

Таблиця 3.2

Основні екрани інтерфейсу системи «FinAgent»

Екран	Призначення	Основні дії користувача
Реєстрація	Створення нового облікового запису	Введення імені, email і пароля
Вхід	Автентифікація користувача	Введення email і пароля
Dashboard	Швидкий перегляд фінансового стану	Перегляд балансу, доходів, витрат і графіків
Транзакції	Ручний облік доходів і витрат	Створення, редагування, видалення та фільтрація операцій
Категорії	Налаштування групування фінансових операцій	Створення та перегляд категорій
Фінансові цілі	Планування накопичень	Створення цілей і перегляд прогресу
ІІІ-чат	Запити природною мовою до власних фінансових даних	Отримання пояснень, аналітики та пропозицій дій

У табл. 3.2 узагальнено основні екрани інтерфейсу. Вони відповідають ключовим сценаріям роботи користувача і показують, що система має не лише серверну реалізацію, а й повноцінний користувацький інтерфейс для щоденного фінансового обліку.

3.3. Тестування основних сценаріїв роботи системи

Після реалізації інтерфейсу та серверної частини було виконано перевірку основних сценаріїв роботи системи. Тестування проводилося з позиції кінцевого користувача і охоплювало реєстрацію, вхід, роботу з транзакціями, категоріями, фінансовими цілями, ІІІ-чатом, підтвердженням дій та експортом даних.

Метою тестування було переконатися, що frontend коректно взаємодіє з backend, дані зберігаються у PostgreSQL, а ІІІ-сервіс не виконує змін без підтвердження користувача. Основні сценарії перевірки роботи додатку наведено в табл. 3.3.

Таблиця 3.3

Перевірка основних сценаріїв роботи системи «FinAgent»

№	Сценарій	Дії користувача	Очікуваний результат	Статус
1	Реєстрація	Заповнити форму реєстрації	Створюється новий обліковий запис	Виконано
2	Вхід до системи	Ввести email і пароль	Користувач переходить до особистого кабінету	Виконано
3	Додавання транзакції	Заповнити форму витрати або доходу	Новий запис з'являється у списку транзакцій	Виконано
4	Редагування транзакції	Змінити суму, опис або категорію	Дані операції оновлюються	Виконано
5	Видалення транзакції	Натиснути кнопку видалення	Запис приховується зі списку активних операцій	Виконано
6	Створення категорії	Додати нову категорію	Категорія доступна для вибору у формах	Виконано
7	Створення фінансової цілі	Ввести назву, суму та термін	Ціль з'являється на сторінці цілей	Виконано

8	Запит до ШІ-чату	Поставити питання природною мовою	Система повертає відповідь на основі фінансових даних	Виконано
9	ШІ-дія з підтвердженням	Підтвердити запропоновану дію	Дані змінюються тільки після підтвердження	Виконано
10	Скасування останньої дії	Виконати undo	Остання підтверджена дія відкочується	Виконано
11	CSV-експорт	Натиснути кнопку експорту	Завантажується CSV-файл із фінансовими даними	Виконано
12	Перевірка авторизації	Надіслати запит без JWT або з неправильним токеном	Backend відхиляє запит і не повертає приватні дані	Виконано
13	Перевірка валідації	Створити транзакцію з некоректною сумою або категорією	Backend повертає помилку валідації	Виконано

Окрему увагу під час тестування приділено сценарію ШІ-дії з підтвердженням. Він є важливим для всієї архітектури системи, оскільки доводить, що мовна модель не змінює фінансові записи самостійно. Спочатку ШІ-сервіс формує очікувану дію, потім backend перевіряє її структуру, а остаточне виконання відбувається тільки після підтвердження користувачем.

Для додаткової перевірки можна використати не лише інтерфейс браузера, а й інструменти розробника, журнали контейнерів або Postman. Це дозволяє переконатися, що запити мають правильні HTTP-статуси, backend повертає очікувані відповіді, а помилки валідації обробляються без аварійного завершення сервісів.

Окрім ручної перевірки через інтерфейс, у проєкті передбачено автоматизовані тести для окремих backend-модулів. Зокрема, перевіряється логіка реєстрації та входу користувача, хешування паролів, генерація і перевірка JWT, шифрування даних, обмеження частоти запитів, а також бізнес-правила для

транзакцій, категорій і фінансових цілей. Такі тести дозволяють перевірити не лише роботу інтерфейсу, а й коректність критичної серверної логіки.

Для ІІІ-сервісу також передбачено окремі перевірки. Вони охоплюють роботу read-only інструментів, формування структурованих дій, аналіз аномальних транзакцій і виправлення ситуацій, коли модель помилково формулює відповідь так, ніби дія вже виконана. Це дозволяє окремо контролювати не лише технічну працездатність ІІІ-модуля, а й правильність його поведінки у фінансовому контексті.

Результати тестування показали, що основні сценарії роботи системи виконуються коректно. Користувач може створити обліковий запис, увійти до системи, вести облік транзакцій, переглядати аналітику, працювати з фінансовими цілями та отримувати відповіді через ІІІ-чат. При цьому зміни фінансових даних, запропоновані ІІІ-сервісом, не виконуються без підтвердження користувача.

Додатково перевірялася стабільність роботи системи після повторного запуску контейнерів через Docker Compose. Після перезапуску сервісів перевірялося, чи зберігаються створені користувачі, транзакції, категорії та фінансові цілі у PostgreSQL, а також чи коректно відновлюється взаємодія між frontend, backend і ІІІ-сервісом. Це підтверджує, що основна логіка застосунку не залежить від тимчасового стану окремого контейнера.

Під час перевірки також враховувалися негативні сценарії, тобто ситуації, у яких користувач вводить некоректні дані або намагається виконати дію без належної авторизації. Наприклад, перевірялася реакція системи на порожні обов'язкові поля, некоректну суму транзакції, неправильний токен доступу або запит до захищеного ресурсу без авторизації. У таких випадках backend не повинен виконувати операцію, а frontend має показати користувачу зрозуміле повідомлення про помилку. Такий підхід дозволяє перевірити не лише успішні сценарії, а й стійкість системи до помилкових або неповних запитів в ключових сервісах проєкту. Це особливо важливо для фінансового застосунку, де

некоректні дані не повинні потрапляти до бази навіть у випадку помилки користувача.

Також перевірялася узгодженість даних після виконання операцій. Після створення, редагування або видалення транзакції перевірялося, чи оновлюється список операцій, аналітичні показники на дашборді та пов'язані фінансові дані. Це важливо, оскільки система працює не лише як журнал записів, а як комплексний фінансовий інструмент, де одна зміна може впливати на баланс, категоріальну аналітику, цілі та відповіді ШІ-асистента.

Висновки до розділу 3

У третьому розділі було показано порядок запуску та практичну роботу системи «FinAgent». Описано запуск застосунку через Docker Compose, призначення основних контейнерів і порядок перевірки доступності системи після налаштування змінних середовища.

Продемонстровано основні екрани користувацького інтерфейсу: реєстрацію, вхід, головну панель, сторінку транзакцій, форму додавання операції, керування категоріями, фінансові цілі, ШІ-чат, підтвердження ШІ-дії та експорт даних у CSV. Це підтверджує, що розроблена система має завершений користувацький контур і дозволяє виконувати базові операції з персональними фінансами.

Також проведено тестування основних сценаріїв роботи. Перевірено створення користувача, автентифікацію, CRUD-операції з транзакціями, роботу з категоріями та цілями, запити до ШІ-чату, підтвердження ШІ-дій, скасування останньої дії та експорт фінансових даних. Результати перевірки показали, що реалізований прототип відповідає поставленим функціональним вимогам і може бути використаний для демонстрації роботи фінансового асистента з контрольованою ШІ-взаємодією.

ВИСНОВКИ

У результаті виконання випускної кваліфікаційної роботи бакалавра було розроблено прототип вебзастосунку «FinAgent» для обліку особистих фінансів зі ШІ-асистентом. У роботі досягнуто поставленої мети: спроектовано та реалізовано програмний комплекс, який дозволяє вести доходи й витрати, працювати з категоріями, переглядати аналітику, створювати фінансові цілі та взаємодіяти з фінансовими даними через діалоговий інтерфейс.

Під час аналізу предметної області було встановлено, що користувачам потрібні не лише інструменти ручного введення транзакцій, а й засоби швидкого розуміння власного бюджету. Огляд аналогів показав, що існуючі рішення мають різні сильні сторони, однак не завжди поєднують облік, аналітику, фінансові цілі та контрольовану ШІ-взаємодію. Це обґрунтувало доцільність розробки системи «FinAgent».

У процесі проєктування було визначено архітектуру системи, яка складається з frontend, backend, ШІ-сервісу та PostgreSQL. Backend реалізовано з використанням шарового підходу, що розділяє HTTP-обробники, бізнес-логіку та доступ до даних. Така структура підвищує підтримуваність коду та полегшує тестування окремих модулів.

Було спроектовано базу даних, яка охоплює користувачів, транзакції, категорії, фінансові цілі, історію чату, журнал дій, налаштування дашборда та додаткові інтеграції. Для безпечної роботи з даними застосовано перевірку власника сутностей, JWT-автентифікацію, bcrypt-хешування паролів, soft-delete та журнал дій для можливого скасування.

Особливістю реалізації є ШІ-чат із підтвердженням дій. Система не дозволяє мовній моделі самостійно змінювати фінансові записи. ШІ-сервіс може сформулювати очікувану дію, але backend виконує її лише після підтвердження користувачем. Це рішення є важливим для фінансової предметної області, оскільки поєднує зручність природномовного інтерфейсу з контрольованістю звичайного вебзастосунку.

Тестування основних сценаріїв показало працездатність ключових модулів: автентифікації, транзакцій, категорій, аналітики, фінансових цілей, CSV-експорту, ШІ-чату, виявлення нетипових витрат і скасування останньої дії. Розроблений програмний продукт може бути використаний як основа для подальшого розвитку, зокрема для покращення ШІ-категоризації, розширення інтеграцій і додавання більш детальних аналітичних сценаріїв.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Spendee: офіційний вебсайт. URL: <https://www.spendee.com/> (дата звернення: 13.06.2026).
2. PocketGuard: офіційний вебсайт. URL: <https://pocketguard.com/> (дата звернення: 13.06.2026).
3. YNAB. The Four Rules. URL: <https://www.ynab.com/the-four-rules> (дата звернення: 13.06.2026).
4. Monobank API: документація. URL: <https://api.monobank.ua/docs/> (дата звернення: 13.06.2026).
5. Go Documentation. URL: <https://go.dev/doc/> (дата звернення: 13.06.2026).
6. Gin Web Framework: документація. URL: <https://gin-gonic.com/docs/> (дата звернення: 13.06.2026).
7. PostgreSQL Documentation. URL: <https://www.postgresql.org/docs/> (дата звернення: 13.06.2026).
8. Python 3 Documentation. URL: <https://docs.python.org/3/> (дата звернення: 13.06.2026).
9. FastAPI Documentation. URL: <https://fastapi.tiangolo.com/> (дата звернення: 13.06.2026).
10. React Documentation. URL: <https://react.dev/> (дата звернення: 13.06.2026).
11. TypeScript Documentation. URL: <https://www.typescriptlang.org/docs/> (дата звернення: 13.06.2026).
12. Vite Documentation. URL: <https://vite.dev/guide/> (дата звернення: 13.06.2026).
13. Docker Compose Documentation. URL: <https://docs.docker.com/compose/> (дата звернення: 13.06.2026).
14. Salt Edge. Account Information API. URL: https://www.saltdge.com/products/account_information (дата звернення: 13.06.2026).
15. Bray T. The JavaScript Object Notation (JSON) Data Interchange Format: RFC 8259. 2017. URL: <https://www.rfc-editor.org/rfc/rfc8259> (дата звернення: 13.06.2026).

- 16.MDN Web Docs. HTTP. URL: <https://developer.mozilla.org/en-US/docs/Web/HTTP> (дата звернення: 13.06.2026).
- 17.Martin R. C. The Clean Architecture. URL: <https://blog.cleancoder.com/uncle-bob/2012/08/13/the-clean-architecture.html> (дата звернення: 13.06.2026).
- 18.Anthropic. Tool use: documentation. URL: <https://docs.anthropic.com/en/docs/agents-and-tools/tool-use/overview> (дата звернення: 13.06.2026).
- 19.Anthropic. Messages API: documentation. URL: <https://docs.anthropic.com/en/api/messages> (дата звернення: 13.06.2026).
- 20.Jones M., Bradley J., Sakimura N. JSON Web Token (JWT): RFC 7519. 2015. URL: <https://www.rfc-editor.org/rfc/rfc7519> (дата звернення: 13.06.2026).
- 21.OWASP. API Security Top 10. URL: <https://owasp.org/API-Security/> (дата звернення: 13.06.2026).
- 22.Lewis J., Fowler M. Microservices. URL: <https://martinfowler.com/articles/microservices.html> (дата звернення: 13.06.2026).
- 23.Recharts: документація. URL: <https://recharts.org/> (дата звернення: 13.06.2026).
- 24.Git Documentation. URL: <https://git-scm.com/doc> (дата звернення: 13.06.2026).
- 25.Dockerfile reference: документація Docker. URL: <https://docs.docker.com/reference/dockerfile/> (дата звернення: 13.06.2026).
- 26.OWASP. Password Storage Cheat Sheet. URL: https://cheatsheetseries.owasp.org/cheatsheets/Password_Storage_Cheat_Sheet.html (дата звернення: 13.06.2026).
- 27.PostgreSQL Documentation. JSON Types. URL: <https://www.postgresql.org/docs/current/datatype-json.html> (дата звернення: 13.06.2026).
- 28.Wiggins A. The Twelve-Factor App. Config. URL: <https://12factor.net/config> (дата звернення: 13.06.2026).

29.CSV Media Type / Internet Assigned Numbers Authority. URL: <https://www.iana.org/assignments/media-types/text/csv> (дата звернення: 13.06.2026).

30.Go Packages. bcrypt. URL: <https://pkg.go.dev/golang.org/x/crypto/bcrypt> (дата звернення: 13.06.2026).

ДОДАТКИ

					ІІЗ.КР.Б–121–26-ІІЗ	69

Повна таблиця тестування основних сценаріїв роботи системи «FinAgent»

У додатку А наведено розширену таблицю перевірки основних користувацьких і системних сценаріїв. Таблиця доповнює скорочений перелік тестів, поданий у розділі 3, і містить передумови, дії користувача, очікуваний результат та статус виконання.

Таблиця А.1

Повна таблиця тестування основних сценаріїв роботи системи «FinAgent»

№	Сценарій	Передумови	Дії користувача	Очікуваний результат	Статус
1	Реєстрація	Користувач не має акаунта	Заповнити форму реєстрації	Створюється акаунт, користувач переходить у систему	Пройдено
2	Вхід	Існує акаунт	Ввести email і пароль	Видається JWT, відкривається dashboard	Пройдено
3	Невірний пароль	Існує акаунт	Ввести неправильний пароль	Форма показує помилку без редіректу	Пройдено
4	Перегляд профілю	Користувач авторизований	Відкрити сторінку профілю	Показуються дані користувача	Пройдено
5	Створення категорії	Користувач авторизований	Додати категорію доходу/витрати	Категорія з'являється у списку	Пройдено
6	Створення транзакції	Є категорія	Додати дохід або витрату	Транзакція зберігається і впливає на баланс	Пройдено
7	Фільтрація транзакцій	Є кілька транзакцій	Задати тип/дату/категорію	Таблиця показує тільки відповідні записи	Пройдено

8	CSV export	Є транзакції	Натиснути експорт CSV	Завантажується CSV-файл з транзакціями	Пройдено
9	Dashboard	Є фінансові дані	Відкрити головну сторінку	Показуються баланс, графіки, останні транзакції	Пройдено
10	Фінансова ціль	Користувач авторизований	Створити ціль і поповнити її	Прогрес цілі оновлюється	Пройдено
11	Monobank token	Є API token	Підключити Monobank у профілі	Токен зберігається зашифровано	Пройдено
12	Імпорт Monobank	Monobank підключений	Запустити імпорт	Нові операції додаються у транзакції	Пройдено
13	ШІ-чат	Є транзакції/цілі	Поставити питання про фінанси	Асистент відповідає українською	Пройдено
14	ШІ що якщо	Є баланс	Запитати «чи можу купити...»	Асистент рахує можливий залишок	Пройдено
15	Очікувана ШІ-дія	Користувач просить додати транзакцію	Асистент готує дію	Дані не змінюються без підтвердження	Пройдено
16	Підтвердження ШІ-дії	Є pendingAction	Натиснути підтвердження	Дія виконується і записується в action log	Пройдено
17	Undo ШІ-дії	Є виконана ШІ-дія	Натиснути undo	Остання дія скасовується	Пройдено
18	Аномальні витрати	Є нетипова велика витрата	Відкрити сторінку аномалій	Показується нетипова витрата	Пройдено
19	Мобільний інтерфейс	Вузкий екран	Відкрити dashboard/transactions/profile	Навігація і таблиці доступні	Пройдено

Продовження табл. А.1

20	Logout	Користувач авторизований	Натиснути вихід	JWT очищується, користувач повертається на login	Пройдено
----	--------	-----------------------------	-----------------	--	----------

Фрагмент конфігурації Docker Compose для запуску системи «FinAgent»

У додатку Б наведено фрагмент конфігураційного файлу docker-compose.yml, який використовується для локального запуску основних компонентів системи. У конфігурації визначено базу даних PostgreSQL, ШІ-сервіс, backend і frontend. Реальні значення паролів, токенів і ключів доступу не подаються, оскільки вони передаються через змінні середовища.

Лістинг коду з файлу docker-compose.yml

```
services:
  postgres:
    image: postgres:16-alpine
    container_name: finagent_postgres
    restart: unless-stopped
    environment:
      POSTGRES_DB: finagent
      POSTGRES_USER: finagent_user
      POSTGRES_PASSWORD: ${POSTGRES_PASSWORD}
    ports:
      - "127.0.0.1:5433:5432"
    volumes:
      - postgres_data:/var/lib/postgresql/data

  ai:
    build:
      context: ./financial-ai-agent
    container_name: finagent_ai
    restart: unless-stopped
    environment:
      ANTHROPIC_API_KEY: ${ANTHROPIC_API_KEY}
      ANTHROPIC_MODEL: ${ANTHROPIC_MODEL}
      AI_SERVICE_PORT: 8000
      INTERNAL_API_TOKEN: ${INTERNAL_API_TOKEN}
    expose:
      - "8000"

  backend:
    build:
      context: ./backend
    container_name: finagent_backend
    restart: unless-stopped
    environment:
      DATABASE_URL: ${DATABASE_URL}
      JWT_SECRET: ${JWT_SECRET}
      APP_PORT: 8080
```

```

    AI_SERVICE_URL: http://ai:8000
    AI_INTERNAL_TOKEN: ${INTERNAL_API_TOKEN}
    ENCRYPTION_KEY: ${ENCRYPTION_KEY}
  ports:
    - "127.0.0.1:8080:8080"
  depends_on:
    - postgres
    - ai

frontend:
  build:
    context: ./frontend
  container_name: finagent_frontend
  restart: unless-stopped
  ports:
    - "127.0.0.1:80:80"
  depends_on:
    - backend

volumes:
  postgres_data:

```

Такий файл дозволяє запускати всі частини системи узгоджено. Backend отримує адресу ІІІ-сервісу через внутрішню мережу Docker, а PostgreSQL використовується як основне сховище фінансових і службових даних. Наявність змінних середовища дає змогу не зберігати секретні параметри безпосередньо у файлі конфігурації.

Фрагмент реалізації підтвердження дії, запропонованої ІІІ-сервісом

У додатку В наведено фрагменти коду, які демонструють механізм контрольованого виконання дій, запропонованих ІІІ-сервісом. Основна ідея полягає в тому, що мовна модель не змінює фінансові дані безпосередньо. Вона формує об'єкт `pendingAction`, а `backend` виконує його тільки після підтвердження користувачем.

Фрагмент коду структури `PendingAction`

```
type PendingAction struct {
    ActionType      string `json:"actionType"`
    EntityType      string `json:"entityType"`
    Amount          float64 `json:"amount,omitempty"`
    TransactionType string `json:"transactionType,omitempty"`
    Description     string `json:"description,omitempty"`
    CategoryID     *int64 `json:"categoryId,omitempty"`
    CategoryName    string `json:"categoryName,omitempty"`
    TransactionDate string `json:"transactionDate,omitempty"`
    TransactionID   int64  `json:"transactionId,omitempty"`
    GoalID         int64  `json:"goalId,omitempty"`
    GoalTitle      string `json:"goalTitle,omitempty"`
    TargetAmount   float64 `json:"targetAmount,omitempty"`
    Deadline       *string `json:"deadline,omitempty"`
    ConfirmText    string `json:"confirmText"`
    RequiresConfirm bool   `json:"requiresConfirm"`
}
```

Якщо результат роботи ІІІ-сервісу містить дію, `backend` не виконує її одразу. Спочатку дія перетворюється на `pendingAction` і повертається на `frontend` у вигляді картки підтвердження.

Фрагмент коду формування `pendingAction`

```
if result.Action != nil && result.Action.ActionType != "" {
    pending, clarify := h.resolveAction(ctx, userID, result.Action,
        categories, goals)
    if pending != nil {
        resp["pendingAction"] = pending
        resp["response"] = "Підготував дію. Підтвердіть її нижче."
        resp["intent"] = "ACTION"
        finalIntent = "ACTION"
    } else if clarify != "" {
        resp["response"] = clarify
    }
}
```

```
}
```

Виконання запропонованої дії відбувається через окремий endpoint після підтвердження користувача. Це дозволяє відокремити етап формування пропозиції від етапу фактичної зміни даних у базі.

Фрагмент виконання підтвердженої дії

```
func (h *ChatHandler) Execute(c *gin.Context) {
    var p PendingAction
    if err := c.ShouldBindJSON(&p); err != nil {
        RespondError(c, http.StatusBadRequest, "INVALID_INPUT",
err.Error())
        return
    }

    userID := middleware.UserIDFromContext(c)

    executed, err := h.runAction(c.Request.Context(), userID, &p)
    if err != nil {
        HandleError(c, err)
        return
    }

    RespondOK(c, http.StatusOK, gin.H{
        "executed": true,
        "action": executed,
    })
}
```

Після виконання дії створюється запис у журналі action_log. Він містить тип дії, сутність, ідентифікатор зміненого запису, службові дані payload та ознаку скасування. Це дає змогу реалізувати відкат останньої III-дії.

Лістинг коду структури ActionLog

```
type ActionLog struct {
    ID          int64          `db:"id"`
    UserID      int64          `db:"user_id"`
    ActionType  ActionType     `db:"action_type"`
    EntityType  EntityType     `db:"entity_type"`
    EntityID    int64          `db:"entity_id"`
    Payload     json.RawMessage `db:"payload"`
    Undone      bool           `db:"undone"`
    CreatedAt   time.Time      `db:"created_at"`
}
```

Отже, механізм pendingAction забезпечує контрольовану взаємодію з III-сервісом. Користувач бачить запропоновану дію, підтверджує або відхиляє її, а

backend відповідає за перевірку прав доступу, виконання зміни та збереження інформації для можливого undo.