МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

Чорноморський національний університет імені Петра Могили

Факультет комп'ютерних наук

Кафедра інтелектуальних інформаційних систем

ДОПУЩЕНО ДО ЗАХИСТУ

Завідувач кафедри інтелектуальних інформаційних систем

_____ Євген СІДЕНКО

«____»_____ 2026 р.

КВАЛІФІКАЦІЙНА РОБОТА

НА ЗДОБУТТЯ ОСВІТНЬОГО СТУПЕНЯ БАКАЛАВРА

ІНТЕЛЕКТУАЛЬНА СИСТЕМА УПРАВЛІННЯ ОСОБИСТИМ І КОЛЕКТИВНИМ БЮДЖЕТОМ ІЗ ВИКОРИСТАННЯМ МОВНОЇ МОДЕЛІ

Спеціальність 122 Комп'ютерні науки

Освітня програма «Комп'ютерні науки»

Здобувачка _____ Марія ДВОРЕЦЬКА

«___»_____ 2026 р.

Керівник канд. фіз-мат. наук, доцент _____ Інесса КУЛАКОВСЬКА

«___»_____ 2026 р.

Миколаїв – 2026

Чорноморський національний університет імені Петра Могили
(повне найменування закладу вищої освіти)

Факультет	Комп’ютерних наук
Кафедра	Інтелектуальних інформаційних систем
Рівень вищої освіти	Перший (бакалаврський)
Освітній ступень	Бакалавр
Спеціальність	122 Комп’ютерні науки
Освітня програма	Комп’ютерні науки

ЗАТВЕРДЖУЮ

Завідувач кафедри інтелектуальних інформаційних систем

_____ Євген СІДЕНКО

«____» _____ 2025 р.

ЗАВДАННЯ
на кваліфікаційну роботу здобувача

Дворецька Марія Михайлівна
((прізвище, ім’я, по батькові здобувача)

1. Тема кваліфікаційної роботи: «Інтелектуальна система управління особистим і колективним бюджетом із використанням мовної моделі».

Керівник роботи: Кулаковська Інесса Василівна, доцент кафедри інтелектуальних інформаційних систем, канд. фіз-мат. наук, доцент.

Затверджена наказом ЧНУ ім. Петра Могили від «25» грудня 2025 р. № 353.

2. Строк представлення кваліфікаційної роботи «___» _____ 2026 р.

3. Очікуваний результат роботи та початкові дані, якщо такі потрібні: інтелектуальна система управління особистим та колективним бюджетом із крос-валютним обліком та інтегрованим ШІ-асистентом на базі протоколу MCP. Початковими даними є структура фінансових транзакцій, ієрархія категорій витрат та специфікації Model Context Protocol для взаємодії з великими мовними моделями.

4. Перелік питань, що підлягають розробці: аналіз архітектурних рішень для побудови масштабованих фінансових систем та протоколів інтеграції з ШІ; проєктування реляційної моделі даних та алгоритмів мультивалютної обробки транзакцій; програмна реалізація серверної частини на Symfony та клієнтського Angular-застосунку з підтримкою асинхронних MCP-інструментів; розгортання системи в Docker-середовищі та тестування функціональної придатності модулів.
5. Перелік графічних матеріалів: презентація з візуалізацією архітектури системи, UML-моделей взаємодії компонентів та інтерфейсів користувача.

Керівник роботи _____ Інесса КУЛАКОВСЬКА
(Особистий підпис) *(Власне ім'я ПРІЗВИЩЕ)*

Здобувачка _____ Марія ДВОРЕЦЬКА
(Особистий підпис) *(Власне ім'я ПРІЗВИЩЕ)*

Дата видачі завдання «23» грудня 2025 р.

КАЛЕНДАРНИЙ ПЛАН
кваліфікаційної роботи

Тема: Інтелектуальна система управління особистим і колективним бюджетом із використанням мовної моделі

№	Найменування роботи	Початок	Закінчення	Примітки
1	Отримання завдання на виконання КР	21.12.2025	24.12.2025	Виконано
2	Аналіз предметної області, фіксація функціональних меж застосунку та постановка архітектурних задач	25.12.2025	30.01.2026	Виконано
3	Огляд літературних джерел за темою КР, аналіз існуючих аналогів та сучасних практик побудови Stateless API систем	31.01.2026	01.03.2026	Виконано
4	Дослідження специфікацій великих мовних моделей, протоколу Model Context Protocol (MCP) та Clean Architecture	02.03.2026	01.04.2026	Виконано
5	Програмна реалізація ядра Symfony (JWT, Voters), фінансових модулів, MCP-сервера та фронтенду Angular	02.04.2026	24.05.2026	Виконано
6	Перший попередній захист КР на засіданні комісії кафедри	25.05.2026	25.05.2026	Виконано
7	Корегування роботи за результатами попереднього захисту	26.05.2026	04.06.2026	Виконано
8	Другий попередній захист КР на засіданні комісії кафедри	05.06.2026	05.06.2026	Виконано
9	Доробка та остаточне оформлення КР	06.06.2026	14.06.2026	Виконано
10	Подання КР, її електронної копії та інших документів (відгуку, рецензії) до захисту	15.06.2026	17.06.2026	Виконано

Керівник роботи _____ Інесса КУЛАКОВСЬКА
(Особистий підпис) *(Власне ім'я ПРІЗВИЩЕ)*

Здобувач _____ Марія ДВОРЕЦЬКА
(Особистий підпис) *(Власне ім'я ПРІЗВИЩЕ)*

Дата складання календарного плану
«29» січня 2026 р.

АНОТАЦІЯ

до кваліфікаційної роботи
здобувачки групи 401з ЧНУ ім. Петра Могили

Дворецької Марії Михайлівни

на тему: **«ІНТЕЛЕКТУАЛЬНА СИСТЕМА УПРАВЛІННЯ ОСОБИСТИМ І КОЛЕКТИВНИМ БЮДЖЕТОМ ІЗ ВИКОРИСТАННЯМ МОВНОЇ МОДЕЛІ»**

Актуальність даної роботи полягає у використанні можливостей штучного інтелекту для автоматизації рутинних фінансових операцій та підвищення зручності роботи користувача. Це вдовольнить потребу у зручному інструменті систематизації та управління особистими й спільними бюджетами та спростить процес ведення обліку витрат та доходів, аналізу фінансової активності та формування звітності.

Об’єктом роботи є системи управління фінансами фізичних осіб, протокол MCP, засоби та технології розробки веб-орієнтованих програмних продуктів.

Предметом роботи є методи та засоби розроблення та реалізації веб-орієнтованої системи управління бюджетом фізичної особи або їх групи та реалізації взаємодії із мовними моделями на основі протоколу MCP.

Метою роботи є створення інтелектуальної системи управління бюджетом користувача із підтримкою протоколу MCP для інтеграції з мовними моделями для автоматизації обліку, категоризації витрат, формування звітності та розподілу фінансових потоків між учасниками.

В результаті виконання роботи було розроблено інтелектуальну систему управління особистим та колективним бюджетом із крос-валютним обліком та інтегрованим ШІ-асистентом. Досліджено інфраструктурні можливості відкритого протоколу Model Context Protocol (MCP) від Anthropic для забезпечення безперешкодної природно-мовної взаємодії між великими мовними моделями та сервісним шаром системи. Проаналізовано математичні моделі рівномірного масштабування лімітів ковзних планів, а також розроблено та розгорнуто повнофункціональне програмне забезпечення, що об’єднує клієнтський Angular-застосунок та серверне ядро на Symfony.

Дана робота складається з чотирьох розділів. У першому розділі проведено аналіз архітектурних рішень для побудови масштабованих фінансових систем та

протоколів інтеграції з ШІ. Другий розділ присвячений об’єктно-орієнтованому проєктуванню, UML-моделюванню бізнес-сценаріїв та формалізації реляційної структури бази даних у третій нормальній формі. У третьому розділі наведено результати програмної реалізації серверної частини (backend) фінансового ядра системи та інструментів MCP-сервера. В четвертому розділі деталізовано архітектуру клієнтського Angular SPA застосунку, описано multistage Docker сценарії виробничого розгортання та представлено протокол перевірки функціональної придатності й тестування модулів. Загальний обсяг роботи – 100 сторінок. Кваліфікаційна робота містить 2 додатки, 37 рисунків, 23 таблиці і 35 джерел посилання.

Ключові слова: управління бюджетом, крос-валютний облік, Model Context Protocol (MCP), великі мовні моделі (LLM), Symfony, Angular, Docker, REST API.

ABSTRACT

to the qualification work by the student of the group 401z of Petro Mohyla Black Sea National University

Dvoretska Mariia

„INTELLIGENT SYSTEM FOR MANAGING PERSONAL AND COLLECTIVE BUDGETS USING A LANGUAGE MODEL”

The relevance of this work lies in the use of artificial intelligence capabilities to automate routine financial operations and improve user convenience. This will meet the need for a convenient tool for organizing and managing personal and shared budgets and will simplify the process of tracking expenses and income, analyzing financial activity, and generating reports.

The object of this work includes personal finance management systems, the MCP protocol, and tools and technologies for developing web-based software products.

The subject of this work is on methods and tools for developing and implementing a web-based budget management system for individuals or groups of individuals, as well as for implementing interaction with language models based on the MCP protocol..

The purpose of the work is to create an intelligent user budget management system with MCP protocol support for integration with language models to automate accounting, categorize expenses, generate reports, and distribute financial flows among participants.

As a result of this work, an intelligent system for managing personal and collective budgets was developed, featuring cross-currency accounting and an integrated AI assistant. The infrastructure capabilities of Anthropic’s open Model Context Protocol (MCP) were investigated to ensure seamless natural language interaction between large language models and the system’s service layer. Mathematical models for uniform scaling of rolling plan limits were analyzed, and fully functional software was developed and deployed, combining a client-side Angular application and a server-side core built on Symfony.

This work consists of four chapters. The first chapter analyzes architectural solutions for building scalable financial systems and AI integration protocols. The second chapter is devoted to object-oriented design, UML modeling of business scenarios, and formalization of the relational database structure in third normal form. The third chapter presents the results of the software implementation of the backend of the system's financial core and MCP server tools. The fourth chapter details the architecture of the client-side Angular SPA application, describes multistage Docker scenarios for production deployment, and presents a protocol for functional verification and module testing.

The total length of the thesis is 100 pages. The thesis contains 2 appendices, 37 figures, 23 tables, and 35 references.

Key words: budget management, cross-currency accounting, Model Context Protocol (MCP), large language models (LLM), Symfony, Angular, Docker, REST API.

ЗМІСТ

СКОРОЧЕННЯ ТА УМОВНІ ПОЗНАКИ

СУБД	– система управління базами даних
ШІ	– штучний інтелект
ПЗ	– програмне забезречення
AI	– Artificial Intelligence
API	– Application Programming Interface
DI	– Dependency Injection
DTO	– Data Transfer Object
HTTP	– Hypertext Transfer Protocol
JSON	– JavaScript Object Notation
JWT	– JSON Web Token
LLM	– Large Language Model
MCP	– Model Context Protocol
ORM	– Object-Relational Mapping
PFM	– Personal Financial Management
REST API	– Representational State Transfer Application Programming Interface
SPA	– Single Page Application
SQL	– Structured Query Language
UML	– Unified Modeling Language

ВСТУП

У сучасних умовах люди все частіше стикаються з необхідністю обробляти значні обсяги інформації, пов’язаної з повсякденними фінансовими операціями. Це створює потребу у зручних інструментах систематизації та управління особистими й спільними бюджетами. Ведення обліку витрат та доходів, аналіз фінансової активності та формування звітності потребують часу й дисципліни, а для групи осіб (пари, сім’ї чи друзів) цей процес стає ще складнішим. Тому актуальним є використання можливостей штучного інтелекту для автоматизації рутинних фінансових операцій та підвищення зручності роботи користувача.

Метою роботи є створення інтелектуальної системи управління бюджетом користувача із підтримкою протоколу MCP для інтеграції з мовними моделями для автоматизації обліку, категоризації витрат, формування звітності та розподілу фінансових потоків між учасниками.

Об'єкт: системи управління фінансами фізичних осіб, протокол MCP, засоби та технології розробки веб-орієнтованих програмних продуктів.

Предмет: методи та засоби розроблення та реалізації веб-орієнтованої системи управління бюджетом фізичної особи або їх групи та реалізації взаємодії із мовними моделями на основі протоколу MCP.

Для досягнення мети необхідно вирішити наступні задачі:

– провести аналіз існуючих систем управління фінансами, їх переваг та недоліків, сформулювати вимоги до функціоналу, архітектури та інтерфейсу майбутньої системи;

– виконати моделювання системи: реалізувати діаграми активностей, послідовностей, потоків та класів. Розробити структуру бази даних;

– обґрунтувати вибір технологій та середовищ розробки. Реалізувати ПЗ з використанням обраних інструментів в обраному середовищі розробки;

– розробити механізм інтеграції із мовними моделями на основі протоколу MCP. Провести тестування функціональних можливостей системи та оцінити її ефективність.

Завданням роботи є розроблення інтелектуальної системи управління особистим та колективним бюджетом з використанням мовної моделі, яка забезпечує можливість додавання та редагування витрат, формування звітності у різних часових зрізах, розподіл витрат між користувачами та застосування функцій штучного інтелекту для спрощення взаємодії з даними.

Робота складається з фахового розділу і спеціальної частини з охорони праці. Пояснювальна записка складається зі вступу, трьох розділів, висновків та додатків

У першому розділі проаналізовано предметну область, виконано огляд існуючих систем управління фінансами, розглянуто їх функціональні можливості, архітектурні підходи та недоліки. Обґрунтовано вибір технологічного стеку основі результатів аналізу та сформульовано вимоги до розроблюваної системи.

У другому розділі виконано проєктування та моделювання системи. Побудовано UML-діаграми прецедентів, класів, послідовностей та станів. Описано логічну структуру та спроектовано реляційну модель бази даних.

У третьому розділі описано архітектурні рішення та реалізацію бекенду системи управління бюджетом: структуру REST API, MCP-сервер для інтеграції асистента на основі мовної моделі, механізм автентифікації, сервісний шар та роботу з базою даних.

У четвертому розділі наведено реалізацію фронтенду: Angular SPA, компонентну архітектуру, інтерцептори, маршрутизацію та взаємодію з API. Описано ключові модулі, інтерфейс користувача та результати тестування.

Декларація про використання ШІ. Під час підготовки кваліфікаційної роботи було використано інструмент Gemini 1.5 Pro. Його було застосовано для таких допоміжних завдань: автоматизація інформаційного пошуку та систематизація джерел літератури; генерація декларативного коду для візуалізації складних архітектурних діаграм; лексичне редагування рукопису та адаптивний переклад іншомовних інженерних специфікацій. Я перевірила всю інформацію, отриману від ШІ, на точність та достовірність і несу повну відповідальність за зміст цієї роботи.

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ ТА ОБҐРУНТУВАННЯ ТЕХНОЛОГІЧНОГО СТЕКУ

1.1 Специфіка управління та проблеми обліку особистих та колективних фінансів у сучасних умовах

Управління особистими фінансами є одним із ключових чинників фінансового добробуту людини. За даними глобальних досліджень фінансової грамотності (S&P Global Financial Literacy Survey), лише близько 52% дорослого населення розвинених країн і менше 40% населення країн, що розвиваються, демонструють достатній для комфортного життя рівень фінансової грамотності. В Україні ця цифра залишається нижчою за середньоєвропейський рівень, що підтверджується дослідженнями Національного банку України щодо фінансової обізнаності населення [1].

Систематичний облік доходів і витрат дозволить виявляти нераціональні витрати та оптимізувати структуру бюджету, планувати накопичення для досягнення фінансових цілей, контролювати виконання бюджетних планів у режимі реального часу та аналізувати динаміку фінансового стану за будь-який обраний період.

Незважаючи на очевидні переваги, більшість людей або взагалі не ведуть облік фінансів, або ведуть його нерегулярно [2]. Однією з причин є висока трудомісткість ручного введення даних. Щоразу заповнювати форму з полями «сума», «категорія», «рахунок», «дата» – для більшості користувачів це відчутний психологічний бар'єр, особливо після кожної дрібної покупки. Відсутність зручних мобільних рішень так само демотивуючий фактор. Чимало застосунків зручні лише на комп’ютері та ноутбуці, або не мають мобільно-орієнтованого інтерфейсу [3]. Складність категоризації впливає на швидкість розподілу витрат по категоріях і вимагає зусиль при прийняті рішення – це відволікає від основної мети: фіксації самого факту витрат. Інший фактор: відсутність підтримки рідної мови та валюти, бо більшість популярних застосунків орієнтовані на ринки США та Євросоюзу.

Кількісна більшість проблем полягає у складності рутинної дії нотування витрат, щоб користувачі не лінувався заповнювати свій «щоденник витрат» та тримав дисципліну в слідкуванні за своїми фінансами. Рішенням може стати інструмент, який допоможе оптимізувати введення бюджету на регулярній основі. Система з вбудованим AI – дає змогу спростити процес категоризації та запису витрат, наприклад при введенні запиту природною мовою: «Кава та сендвіч 250 грн», система сама розпізнає категорію, суму та валюту.

У вирішенні останньо згаданої проблеми, було реалізувано багатовалютність. Для України ця тема є особливо актуальною одразу з кількох причин:

– міграційні процеси. Значна частина українців проживає або тимчасово перебуває за кордоном і оперує кількома валютами одночасно: гривнею (UAH), євро (EUR), злотими (PLN), фунтами стерлінгів (GBP) тощо;

– заощадження в іноземній валюті. Широко розповсюджена практика зберігання коштів у доларах або євро поряд із гривневими рахунками;

– нестабільний обмінний курс. Регулярна зміна курсів вимагає фіксації курсу обміну на момент транзакції для коректного ретроспективного аналізу.

Таким чином, система управління бюджетом для українських користувачів повинна підтримувати щонайменше 5–6 валют (UAH, USD, EUR, PLN, GBP, CZK) та зберігати історію курсів обміну для кожної міжвалютної операції.

Окремий пласт задач – пов'язаний із колективним управлінням фінансами – ведення спільного бюджету родини, пари, студентської групи або робочої команди. Спільне проживання, спільні подорожі або оренда житла групою людей – у цих випадках виникають принципово нові вимоги:

– спільний доступ до даних для кількох осіб одночасно;

– рольова модель: потрібно розрізняти власника бюджету, тих хто може редагувати, і тих хто лише переглядає;

– прозорість витрат: кожен учасник бачить, хто і що витратив;

– відокремленість: особисті та загальні витрати мають зберігатися незалежно, але мати єдиний інтерфейс.

Стандартні рішення для особистих фінансів зазвичай не підтримують такий рівень груповості, багатовалютні витрати та можливість запису людською мовою, або реалізують їх через платні підписки (наприклад, YNAB Together, Honeydue).

1.2 Аналіз наявних рішень для автоматизації обліку витрат та їх недоліків

Оскільки проблема не нова вже існує чимала кількість спроб та способів для її вирішення, які можна поділити на три основні категорії [1, 4]: слідкування витрат безпосередньо в банківських застосунках (Monobank, Revolut, PKO BP), сторонні програми та застосунки типу PFM (Money Lover, Wallet, Spendee, YNAB) та старий перевірений метод – табличні процесори (Excel, Google Sheets).

Кожна з категорій рішень має свої переваги та недоліки, ось основні з них:

– банківські застосунки – їх основним плюсом є автоматичне підтягування транзакцій та безпека, оскільки жоден сторонній ресурс не отримує персональну інформацію про користувача та його фінансові потоки. Однак в цьому полягає і головний мінус – обмеженість лише однією екосистемою, тобто готівка та витрати і рахунки в іншому банку не будуть враховуватись. Окрім цього, не всі банківські системи передбачають можливість взаємного перегляду витрат між користувачами, а тому це слабкий інструмент для колективного бюджету. Не згадуючи навіть, про аспект аналізу та планування бюджету. Майже не існує банківський систем, які б пропонували подібний функціонал;

– спеціалізовані PFM-застосунки – такі застосунки в більшості випадків є доволі гарно візуалізовані та оптимізовані, що позитивно впливає на мотивацію користування. Підтримка мультивалютності часто стає в пригоді користувачам, які живуть за кордоном і мають кілька різних карток, або ж у випадку подорожі, коли валюта звичайних витрат змінюється. Основним мінусом таких платформ є платний функціонал, що є доволі дратуючим фактором, оскільки інколи доволі сильно урізає можливості користувача. Так само деколи інтерфейс може здатися складним і заплутаним, через бажання розробників зробити продукт, який

включатиме в себе все, але в більшості випадків, їх продукт не буде містити інтегрованої мовної моделі, а представлятиме скоріш просто статистичні звіти;

– табличні нотатники – максимально гнучкий та до того ж безкоштовний інструмент, який має свою прихильну аудиторію користувачів, які після спроб знайти зручний застосунок повертались до перевіреної класики бухгалтерії малого бізнесу. З мінусами все доволі просто – це відсутність адаптації під мобільний пристрій, навіть не зважаючи, що існують застосунки для редагування exel файлів на телефоні, вони не зручні та мало функціональні. Це призводить до складності та малої ефективності введення даних «на ходу». Окрім цього – майже нульова автоматизація, не враховуючи скрипти, які не кожен може і хоче програмувати.

Проаналізувавши всі перераховані категорії рішень було побудовано таблицю порівняння вже існуючих способів та власної системи по основним критеріям (табл. 1.1).

Таблиця 1.1 – Порівняльна таблиця наявних рішень та власного

Рішення	*Автоматизація*	*Колективний доступ*	*Аналітика*	*Введення даних*
Банківські застосунки	Висока (картки)	Обмежений / відсутній	Базова	Автоматичне
Спеціалізовані PFM (Wallet)	Середня (імпорт)	Платний / середній	Просунута	Ручне / Складне
Excel/Sheets	Низька / середня (скрипти)	Високий	Залежить від юзера	Ручне
Власна система	Висока (AI/MCP)	Високий (групова система)	Варіативна (залежить від налаштувань)	Ручне (етапами) / природна мова (чат-бот)

Підсумовуючи все вище описане, ось ключові недоліки у вже існуючих розв’язаннях, які треба врахувати при створені власної системи:

– відсутність природномовного інтерфейсу. Жодне з розглянутих рішень не має інтеграції з великими мовними моделями через стандартизований протокол, що дозволяв би вводити транзакції через розмовний інтерфейс;

– обмежений груповий бюджет. Більшість рішень або не підтримують спільний бюджет взагалі, або реалізують його примітивно (лише для 2 осіб, без ролей);

– мовний і регіональний бар'єр. Більшість якісних рішень не мають підтримки українських реалій (UAH, українська мова, відповідні категорії);

– хмарна залежність або відсутність власного розгортання. Комерційні рішення зберігають дані на своїх серверах, що породжує питання конфіденційності. Open-source альтернативи вимагають складного налаштування;

– висока вартість. Якісні рішення є платними за умов, неприйнятних для більшості українських користувачів.

Отже, способів та засобів на введення фінансів стає все більше і з часом цей ринок розвивається, але основною перешкодою залишається так зване «тертя» (friction) між користувачем і системою, з усіма їх мінусами та незручностями. Таким чином, існує потреба у створенні відкритої, мультиплатформенної системи, яка мінімізує зусилля на введення даних не без участі інтегрованої системи AI.

1.3 Великі мовні моделі (LLM) та протокол MCP як інструменти інтелектуалізації програмних систем

Інтегрування до системи великих мовних моделей сильно змінює спосіб взаємодії користувача з даними [5], зокрема:

– інтелектуальний класифікатор [6]: на відміну від жорстких алгоритмів, які представлені в більшості сторонніх розв’язань, LLM здатна зрозуміти контекст транзакції та присвоїти вказану підкатегорію, наприклад «Starbucks» та «Netflix», до вже існуючої категорії, наприклад «Кафе» та «Підписки» відповідно;

– інтерфейс природньою мовою: замість заповнення форм так само існує здатність передбачити можливість користувачу описувати реченням його операцію, а мовна модель сама розпізнає ключові слова та занесе витрату чи дохід, або ж дасть відповідь на питання стосовно неї. Приклад таких речень, «Запиши 500 гривень на продукти» або «Скільки я витратив на каву минулого тижня?»;

– синтез аналітики: модель можна навчити не тільки виводити цифри як звіт, а й робити текстові висновки, наприклад: «Ваші витрати на транспорт зросли на 20% порівняно з минулим місяцем через часті поїздки на таксі»;

– аналіз та планування: можливість аналізувати попередню історію витрат за певний період часу, зокрема кількох місяців, та створення на їх основі плану на майбутній тиждень, місяць, чи кілька місяців, пропорційно розподіляючи задану суму між категоріями витрат.

Протокол MCP [7], в свою чергу, як архітектурний міст в контексті даного проекту і є однією з ключових технічних частин. Це відкритий стандарт, який дозволяє моделям штучного інтелекту безпечно та структуровано взаємодіяти з локальними або віддаленими даними та інструментами.

Як це працює: раніше для кожного AI-асистента доводилося писати власну, часами унікальну, логіку підключення до бази даних. За використанням цієї технології Symfony-сервер стає MCP-сервером і надає ШІ-моделі набір інструментів та ресурсів для виконання завдань. Тобто, коли користувач пише в свій запит в чат, LLM через MCP-протокол звертається до сервера з питанням користувача, в результаті чого вдається отримати необхідні дані, з яких в подальшому і формується відповідь користувачу [8].

Таким чином, вдається досягти:

– відокремлення логіки, коли модель ШІ не має прямого доступу до бази даних (MySQL) і працює лише через суворо визначений інтерфейс MCP-сервера;

– контекстній обізнаності – модель отримає лише ті дані, які зараз необхідні для відповіді на конкретний запит, не більше, ні менше – це дозволяє економити ресурси та позитивно впливає на рівень безпеки;

– стандартизації, тобто використання MCP робить новостворену систему сумісною з різними моделями: OpenAI, Anthropic, Llama, без переписування коду інтеграції;

– підтримки різних типів транспорту – архітектура системи дозволяє MCP-серверу взаємодіяти як через стандартний ввід/вивід (stdio) для локальних

середовищ, так і через HTTP SSE (Server-Sent Events) для веб-орієнтованих клієнтів.

1.4 Оцінка переваг фреймворку Symfony та екосистеми Angular для розробки складних вебзастосунків

Symfony – це високонадійний PHP-фреймворк [9], який використовує архітектурний шаблон MVC (Model-View-Controller) та впроваджує найкращі практики інженерії ПЗ. В рамках практичної реалізації розглядається використання новлї актуальнлї версії, яка гарантує тривалу підтримку та виправлення помилок безпеки, що є критичним для систем, які оперують фінансовими даними користувачів для зберігання їх приватності.

Окрім цього, контейнер сервісів Symfony дозволяє створювати слабкозв'язані компоненти, це, в свою чергу, полегшує тестування та дозволяє легко інтегрувати сторонні сервіси для взаємодії з AI. Таким чином планується впровадити залежності, які допомагатимуть працювати системі як одне ціле.

Фреймворк Symfony [10] включає в себе кілька важливих бібліотек, які будуть інтегровані в практичну реалізацію проекту. Одна з них, це бібліотека Doctrine ORM, яка забезпечуватиме високий рівень абстракції при роботі з базою MySQL, дозволяючи при розробці працювати з об'єктами (Entities) замість написання складних SQL-запитів. Це сприяє мінімізації ризику помилок при роботі з транзакціями та при взаємодії з базами даних в цілому.

Ще однією важливою складовою є бібліотека Security Bundle, яка у свою чергу, є потужним механізмом автентифікації та автоматизації, що підтримує кастомні ролі та виробничий доступ до ресурсів, що необхідно для реалізації груп колективного бюджету.

Тепер варто розглянути екосистему Angular [11] в контексті створення сучасного Frontend. Angular від Google обрано як основний інструмент для побудови SPA через його сувору структуру та орієнтованість на Enterprise-рішення.

Це є важливим аспектом, враховуючи, що поточне ПЗ пов’язане з фінансами і потребує серйозного та фундаментально продуманого підходу до розробки.

Angular має низку інших переваг, наприклад компонентно-орієнтована архітектура. Дана екосистема дозволяє розділити інтерфейс на незалежні модулі, це вагомо спрощує розробку складних дашбордів із графіками та таблицями, що є доволі вагомим при задачах проекту ніші фінансів.

Використовуючи TypeScript вдасться дотримуватись статичної типізації, що в свою чергу зменшить кількість помилок на етапі розробки та допоможе зробити код більш зрозумілим для стороннього читача або розробника і підтримуваним в процесі розробки, більш легким до змін та модифікацій, що є критично вадливим для великих проектів.

Двостороннє зв'язування даних дозволить автоматично синхронізувати стан інтерфейсу з моделлю даних, що забезпечує швидкий відгук системи при зміні фінансових показників. Такий підхід забезпечує зручність при роботі з формами та полями не тільки користувачу, але й спростить розробку.

Так само, Angular надає потужні інструменти для роботи з REST API, включаючи інтерцептори для автоматичного додавання JWT-токенів у заголовки запитів, що забезпечує безшовну безпеку.

Тобто, поєднання Symfony та Angular дозволяє реалізувати архітектуру Decoupled, де серверна частина – Symfony, виступає виключно як постачальник даних через REST API, а клієнтська частина – Angular, відповідає за візуалізацію та користувацький досвід. Це дозволяє в майбутньому легко замінити або додати нові клієнти, не змінюючи логіку сервера.

1.5 Постановка задачі на розробку інтелектуальної системи

На основі проведеного аналізу предметної галузі, огляду існуючих рішень та дослідження технологій (LLM, MCP, Symfony, Angular) сформульовано наступну задачу: «Розробити вебзастосунок для управління особистим та груповим бюджетом», що забезпечує:

– ведення повного обліку фінансових операцій (транзакцій) з категоризацією, прив'язкою до рахунків та підтримкою кількох валют;

– управління бюджетними планами на різні часові горизонти;

– спільний доступ до фінансових даних у межах груп із рольовою моделлю;

– аналітику та звітність у вигляді діаграм і таблиць;

– природномовний інтерфейс введення транзакцій через інтеграцію з Claude AI за протоколом MCP.

Для досягнення поставленої мети визначено наступні ключові завдання:

– розробка ядра системи (Backend), що включатиме в себе створення REST API на базі Symfony. Для цього необхідно буде реалізувати надійну схеми бази даних (MySQL) з підтримкою м'якого видалення (Soft delete) та індексації. Окрім цього, важливим є впровадження автентифікації на основі JWT (access та refresh токени), оскільки безпека є пріоритетною в ситуаціях роботи з персональними даними користувача;

– реалізація інтелектуального шару: розробка MCP-сервера для забезпечення взаємодії мовної моделі з фінансовими даними користувача та створення набору інструментів (Tools) для автоматичної категорізації;

– створення клієнтських інтерфейсів: розробка SPA-застосунку на Angular 17+ з динамічною візуалізацією звітів та реалізація оперативного введення даних природною мовою.

Щодо вимог до системи, вони поділяються на функціональні та нефункціональні. До перших варто віднести:

– управління обліковими записами: система забезпечує реєстрацію користувачів із підтвердженням email та вхід через JWT-токени (`access + refresh`). Нові профілі отримують обмежену роль `ROLE_CANDIDATE` до активації адміністратором, після чого набувають статусу `ROLE_USER` із повним доступом та можливістю редагування чи видалення акаунту;

– управління рахунками: користувач може створювати, редагувати та видаляти необмежену кількість рахунків різних типів (картки, готівка, гаманці) із фіксацією валюти й поточного балансу. Один із рахунків призначається «за замовчуванням» для автоматичного підставляння під час створення фінансових операцій;

– управління категоріями: система підтримує дворівневу ієрархічну структуру категорій та підкатегорій з типами income, expense або both. Окрім базових глобальних категорій, доступних усім, користувачі мають можливість створювати власні персоналізовані елементи;

– облік транзакцій: сервіс забезпечує реєстрацію, редагування, видалення та перегляд доходів і витрат із фільтрацією та пагінацією. Кожна транзакція містить суму, дату, прив'язку до рахунку й категорії, підтримує міжвалютний обмін за курсом та може належати до особистого або групового бюджету;

– планування бюджету: користувачі можуть встановлювати часові ліміти (від щоденних до щорічних) для категорій у фіксованому або ковзному форматах. Система автоматично розраховує виконання лімітів у режимі «план/факт», враховуючи пропорційність розподілу коштів за минулі дні для рівномірних планів;

– управління групами: користувач може ініціювати створення групи, стаючи її власником із правом керування ролями (owner та member) та видалення учасників. Приєднання нових користувачів відбувається через надсилання токенізованих запрошень на email, при цьому кожен учасник має право самостійно залишити групу;

– аналітика та звіти: система генерує комплексні фінансові звіти, включаючи загальний баланс, кругові діаграми розподілу витрат за категоріями та графіки динаміки грошових потоків. Окремо реалізовано аналітику виконання планів бюджету та детальний аудит оборотів по кожному окремому рахунку;

– AI-інтеграція через MCP: застосунок містить вбудований MCP-сервер через stdio-транспорт для безпосередньої інтеграції з Claude Desktop. Сервер автентифікує AI-сесії за допомогою JWT і надає інструменти для перегляду довідників та автоматичного створення транзакцій мовною моделлю;

– адміністрування: адміністратори із рольовою моделлю `ROLE_ADMIN` мають повний доступ до моніторингу списку користувачів системи. Вони володіють повноваженнями блокувати профілі, змінювати їхні ролі та активувати нові облікові записи кандидатів.

Нефункціональні в свою чергу виглядатимуть так:

– безпека: доступ до всіх ендпоінтів API (за винятком авторизації) суворо обмежений JWT-валідацією, а паролі користувачів захищені стійким хешуванням bcrypt. Ізоляція даних реалізована на рівні сервісного шару з обов'язковим поверненням стандартних HTTP-кодів помилок (401, 403, 404);

– продуктивність: час відповіді серверного API на більшість запитів не повинен перевищувати 500 мс. Оптимізація клієнтської частини досягається за допомогою механізму Lazy Loading для завантаження модулів Angular та кешування статичних довідників через shareReplay;

– розгортання: архітектура застосунку підтримує швидке локальне розгортання в ізольованих контейнерах за допомогою `docker compose up`. Для сумісності з недорогими shared-хостингами передбачено роботу без Docker, а JWT-токени дублюються в cookie для обходу серверних обмежень на заголовки;

– розширюваність: платформа підтримує гнучке масштабування MCP-інструментів через реалізацію ToolInterface та автоматичну реєстрацію за допомогою DI-тегів. REST API є повністю версіонованим `(/api/v1/)`, що гарантує зворотну сумісність при додаванні нових функцій;

– підтримуваність: програмний код бекенду спроєктовано за принципами SOLID (Single responsibility, Open-closed, Liskov substitution, Interface segregation, Dependency inversion) з чітким розмежуванням обов'язків між тонкими контролерами, бізнес-сервісами та репозиторіями. Фронтенд-частина побудована

на базі сучасних standalone-компонентів Angular із впровадженням залежностей через механізм Dependency Injection.

Висновки до розділу 1

У першому розділі було проведено комплексний аналіз предметної області, розглянуто існуючі підходи до управління фінансами та обґрунтовано вибір технологічного стеку для розробки інтелектуальної системи. Як результат було виявлено, що існуючі рішення у вигляді PFM-сервісів, банківських застосунків і таблиць мають суттєві недоліки: складність ручного введення даних та відсутність інструментів для глибокого аналізу. Це обґрунтовує необхідність розробки інтелектуальної системи, яка мінімізує складнощі комунікації між користувачем та інтерфейсом за допомогою штучного інтелекту.

Ключовим інноваційним елементом роботи визначено використання великих мовних моделей та протоколу Model Context Protocol. Впровадження MCP дозволяє створити безпечну архітектуру, де ШІ взаємодіє з даними через стандартизований інтерфейс, забезпечуючи контекстну обізнаність асистента без ризику для конфіденційності фінансової інформації.

Для реалізації проєкту обрано професійний технологічний стек: Symfony для побудови стабільного Backend API та Angular 17+ для створення динамічного клієнтського інтерфейсу. Ця комбінація у поєднанні з JWT-автентифікацією забезпечує масштабованість системи та відповідність сучасним стандартам безпеки вебзастосунків.

На основі проведеного аналізу сформульовано технічне завдання на розробку багатокомпонентної системи з використанням REST API, Web UI. Визначено основні функціональні вимоги, серед яких: облік транзакцій, управління спільними бюджетами та введення витрат природною мовою, що є фундаментом для подальшого проєктування системи. Правила оформлення звітності, схем, текстових матеріалів та підсумкових додатків лістингів вихідного коду виконуються у суворій відповідності до національного стандарту ДСТУ 8302:2015 [12].

2 ПРОЄКТУВАННЯ ТА МОДЕЛЮВАННЯ СИСТЕМИ УПРАВЛІННЯ БЮДЖЕТОМ

2.1 Побудова діаграми прецедентів для визначення функціональних меж системи

На етапі проектування інтелектуальної системи управління бюджетом одним з першочергових завдань є визначення функціональних меж застосунку та ролей користувачів. Для візуальної демонстрації взаємодії зовнішніх акторів з ключовими сервісами системи було використано діаграму прецедентів (Use Case Diagram) [13, 14]. Це дозволило краще зрозуміти структуру та взаємозв’язки, які є критичними для коректної реалізації системи.

Побудова діаграм прецедентів дозволяє зафіксувати межі програмного продукту, визначити ролі зовнішніх суб'єктів та описати всі можливі варіанти взаємодії користувача чи сторонніх сервісів із ядром системи.

У системі управління бюджетом та інтелектуальної обробки транзакцій визначено п'ять зовнішніх акторів, кожен з яких володіє унікальним набором прав доступу та сценаріїв взаємодії. Детальний опис акторів наведено у таблиці 2.1.

Таблиця 2.1 – Класифікація та ідентифікатори акторів системи

Актор	*Ідентифікатор*	*Опис ролі та функціональних обмежень*
Гість	—	Неавторизований відвідувач вебзастосунку, що не має активної сесії. Доступний лише базовий функціонал створення облікового запису.
Кандидат	`ROLE_CANDIDATE`	Зареєстрований користувач, обліковий запис якого ще не пройшов верифікацію та активацію адміністратором. Доступ до фінансових модулів заблоковано.
Користувач	`ROLE_USER`	Активований суб'єкт системи. Володіє повним доступом до особистих рахунків, категорій, транзакцій, аналітики та групових фінансових планів.
Адміністратор	`ROLE_ADMIN`	Суперкористувач, який здійснює глобальний моніторинг системи. Успадковує всі прецеденти `ROLE_USER` та додатково керує життєвим циклом акаунтів.
Claude Desktop	MCP-клієнт	Зовнішній AI-агент, що здійснює програмну взаємодію із системою від імені авторизованого користувача через протокол Model Context Protocol.

Важливою архітектурною особливістю системи є ієрархія акторів. Роль `Адміністратор` пов'язана відношенням узагальнення (успадкування) з роллю `Користувач`. Це означає, що адміністратор автоматично отримує права на виконання всіх бізнес-сценаріїв звичайного користувача без необхідності дублювання зв'язків на діаграмах.

Оскільки розроблювана система містить розгалужену бізнес-логіку та інтеграцію з великими мовними моделями, загальна модель прецедентів була декомпонована на чотири взаємопов'язані функціональні підсистеми (рис. 2.1-2.4). Це дозволяє уникнути нагромадження зв'язків та спрощує читання схем.

Даний модуль (рис. 2.1) описує початкові етапи взаємодії користувачів із системою, механізми безпеки JWT (JSON Web Tokens), а також інструменти модерації з боку адміністратора.

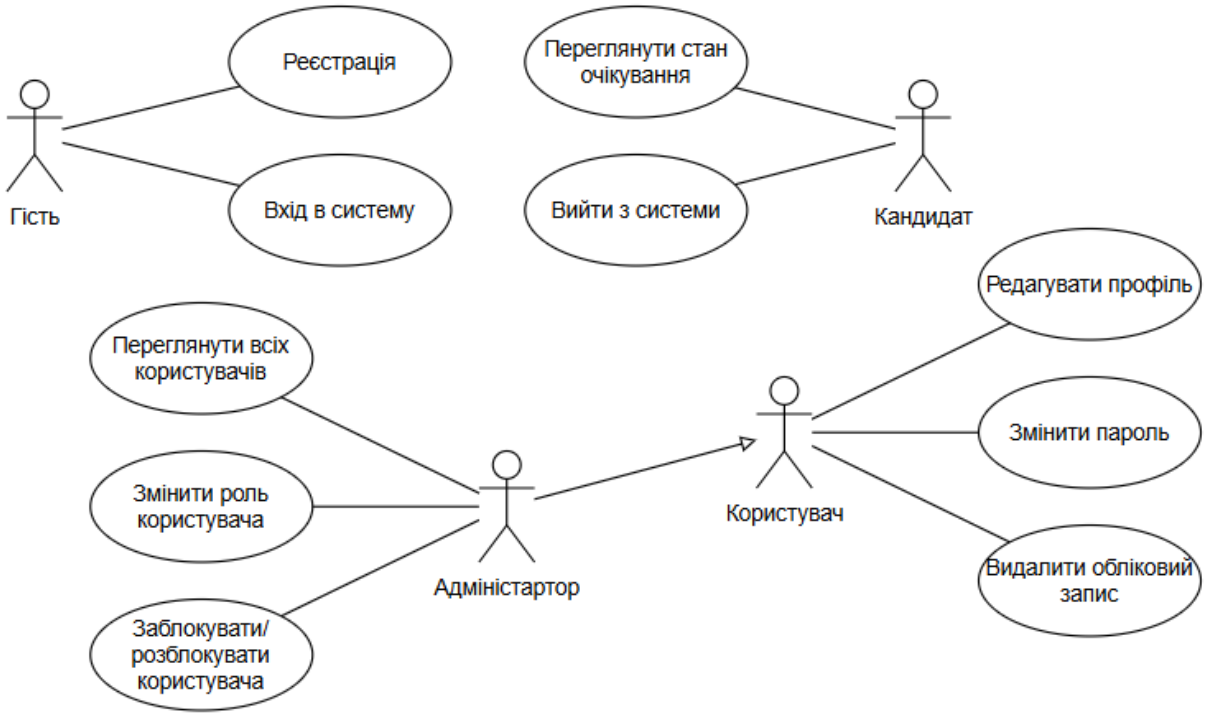

Рисунок 2.1 – Підсистема «Доступ, автентифікація та адміністрування».

Наступний блок (рис. 2.2) є основним для ролі `Користувач` і покриває класичні CRUD-операції (Create, Read, Update, Delete) [14] над особистими фінансовими даними, а також формування складних аналітичних зрізів та звітів.

Рисунок 2.2 – Підсистема «Особистий фінансовий менеджмент та аналітика»

Модуль зображений на рисунку 2.3 фіксує логіку взаємодії користувачів у межах сумісного ведення домашнього господарства або бізнес-проєктів. Він регламентує процеси делегування доступу через токенізовані запрошення.

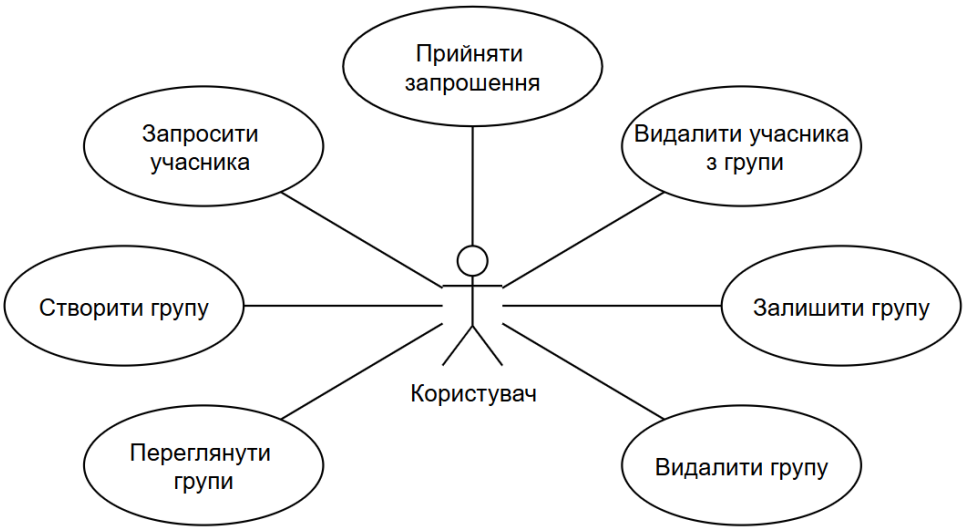

Рисунок 2.3 – Підсистема «Колективний бюджет та управління групами»

Інтелектуальний модуль (рис. 2.4) відображає специфіку архітектури системи як сервера контексту. Зовнішній клієнт `Claude Desktop` викликає чітко регламентовані інструменти протоколу MCP. Прецедент створення транзакції через ШІ містить зв'язок `<<include>>` (включення) [13] до базового прецеденту

створення транзакції користувачем, оскільки вони використовують єдиний сервісний шар Symfony.

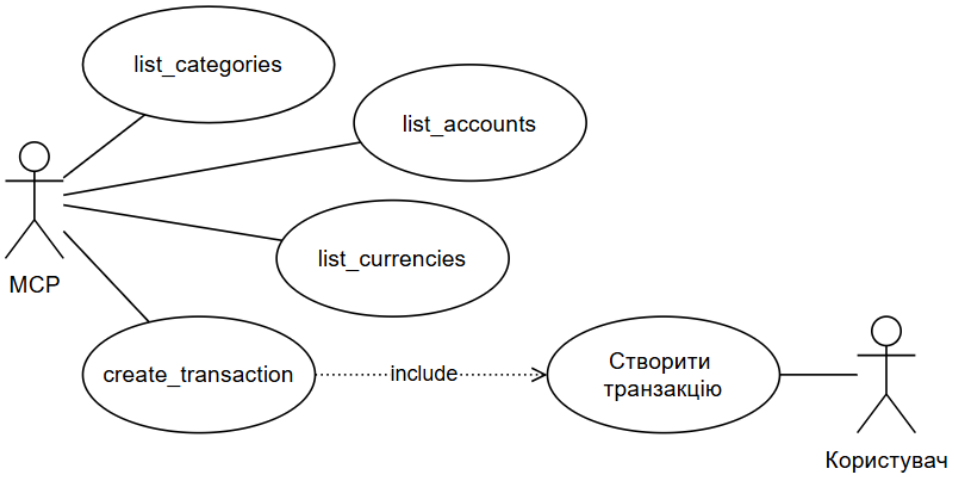

Рисунок 2.4 – Інтелектуальний модуль «MCP-інструменти (AI-агент)»

Для узагальнення прав доступу та наочного відображення розподілу функцій між різними типами акторів сформовано матрицю відповідності (табл. 2.2).

Таблиця 2.2 – Матриця розподілу функціональних прецедентів за ролями

Прецедент / Функціональний блок	*Гість*	*Кандидат*	*Користувач*	*Адміністратор*	*MCP*
Реєстрація нового акаунту	✓	—	—	—	—
Вхід у систему та JWT-автентифікація	✓	✓	✓	✓	—
Моніторинг статусу активації профілю	—	✓	—	—	—
CRUD-операції над рахунками	—	—	✓	✓	—
CRUD-операції над категоріями	—	—	✓	✓	—
Перегляд та керування лімітами	—	—	✓	✓	—
Створення фінансових транзакцій	—	—	✓	✓	✓
Модифікація та видалення транзакцій	—	—	✓	✓	—
Управління групами та інвайтами	—	—	✓	✓	—
Перегляд графіків аналітики та звітів	—	—	✓	✓	—
Адміністрування користувачів та блокування	—	—	—	✓	—
Читання довідників через MCP (`list_*`)	—	—	—	—	✓

Функціональні можливості системи, відповідно до діаграми, було згруповано за логічними модулями (табл. 2.2), що дозволить чітко визначити межі відповідальності кожного компонента API.

2.2 Об’єктно-орієнтоване моделювання структури системи: діаграма класів

На етапі об’єктно-орієнтованого моделювання було розроблено статичну структуру системи, яка відображає взаємозв’язки між сутностями предметної області. Діаграма класів (Class Diagram) формалізує внутрішню структуру системи, визначаючи класи, їхні внутрішні атрибути, методи, інкапсуляцію даних, а також типи статичних взаємозв'язків між ними [13].

Для забезпечення високої масштабованості, ремонтопридатності та слабкої зв'язності компонентів (Low Coupling) архітектуру розроблюваної системи декомпоновано на п'ять логічних шарів згідно з паттерном Clean Architecture [15]:

– Domain (Entity) Layer – рівень сутностей предметної області. Представлений POPO-класами (Plain Old PHP Objects), які мапуються на таблиці реляційної СУБД за допомогою Doctrine ORM [13, 16];

– Repository Layer – класи шлюзів доступу до бази даних, що ізолюють SQL/DQL (Doctrine Query Language) запити від бізнес-логіки [13];

– Service Layer – рівень бізнес-сервісів, що містить транзакційну логіку, координацію об'єктів та виконання розрахунків;

– MCP Layer – інфраструктурний прошарок сервера контексту, що забезпечує стандартизований інтерфейс для взаємодії зі сторонніми LLM-клієнтами;

– Controller Layer – рівень представлення REST API контролерів, відповідальних за прийом HTTP-запитів, десеріалізацію DTO та повернення відповідей.

Загальну архітектурну модель класів розділено на дві функціональні діаграми (рис. 2.5 та 2.6): структуру сутностей [17] предметної області (Domain) та структуру взаємодії сервісів з модулем штучного інтелекту (Service & MCP).

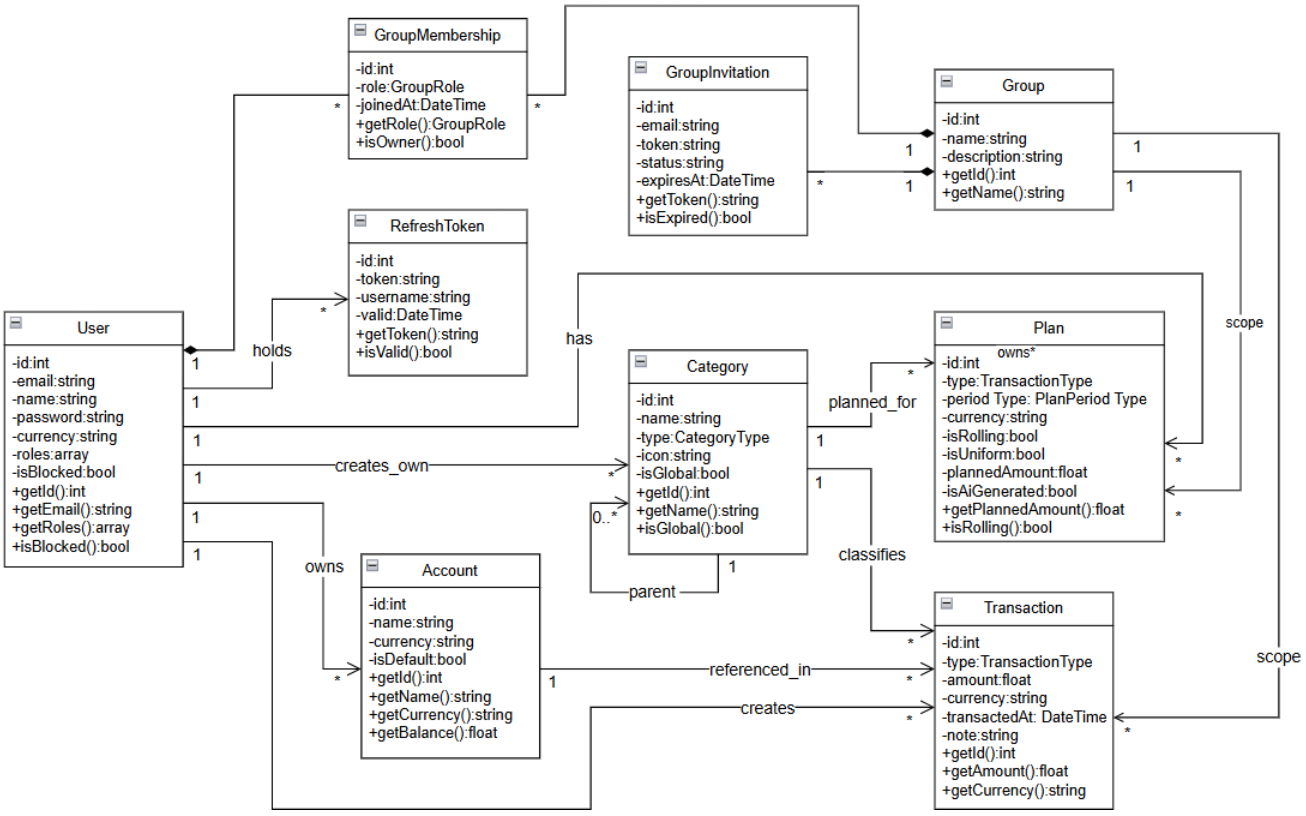

Рисунок 2.5 – Структурна модель предметної області (Domain Layer)

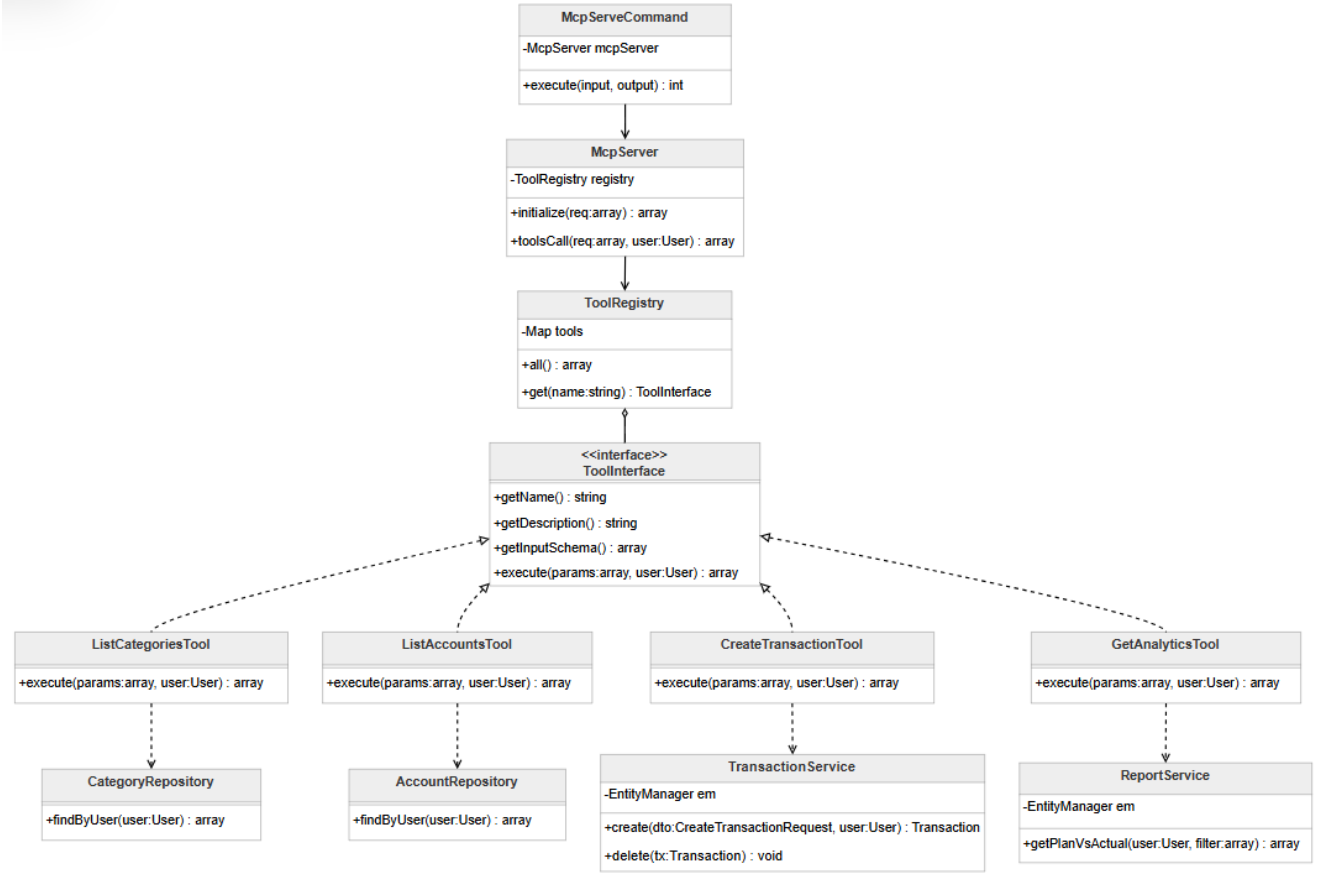

Рисунок 2.6 – Модель сервісного шару та інфраструктури MCP

Цей прошарок фіксує склад і типи даних фінансового застосунку. Об'єкти містять приватні (-) властивості для захисту цілісності даних та публічні (+) методи доступу (getters/setters).

Дана модель (рис. 2.6) відображає архітектуру обробки запитів від штучного інтелекту. Класи інструментів `Tools` імплементують загальний контракт поведінки, заданий інтерфейсом. Зв'язок між сервісами та інфраструктурою MCP побудовано на принципі інверсії залежностей (Dependency Inversion) із SOLID [18].

Правильне визначення зв'язків на рівні об'єктів безпосередньо впливає на цілісність даних та відсутність аномалій у базі даних:

– User ↔ Account (One-to-Many): зв'язок відображає володіння фінансовими рахунками. Користувач виступає коренем агрегації. З метою оптимізації продуктивності поточний баланс рахунку винесено в розряд обчислювальних властивостей (`<<calculated>>`), що розраховуються динамічно через агрегаційні SQL-функції без фізичного дублювання в таблиці;

– Transaction ↔ Category, Account (Many-to-One): транзакція жорстко залежить від наявності рахунку та категорії видатків/доходів. Опціональний зв'язок з класом `Group` (`group_id: int|null`) визначає контекст транзакції: особистий облік або колективний бюджет;

– Category ↔ Category: реалізує паттерн Composite (Компонувальник) в обмеженому форматі. Поле `parent_id` вказує на батьківський клас тієї ж сутності, забезпечуючи деревоподібну ієрархію з глибиною вкладеності в один рівень (підкатегорії);

– Group ↔ GroupMembership ↔ User (Many-to-Many через сутність зв'язку): прямий Many-to-Many зв'язок між користувачами та групами декомпоновано за допомогою проміжного класу `GroupMembership`. Це відношення типу Композиція, оскільки запис членства не має змісту і каскадно видаляється у разі ліквідації об'єкта `Group` або `User`. Клас зберігає додаткові бізнес-атрибути: дату приєднання та роль учасника (`GroupRole`);

– ToolRegistry ↔ ToolInterface: `ToolRegistry` виступає колектором (контейнером) для інструментів ШІ. Використовується слабкий зв'язок агрегації, за якого інструменти реєструються динамічно за допомогою механізму тегування Symfony DI (`#[AutoconfigureTag('mcp.tool')]`) і можуть існувати незалежно від реєстру.

2.3 Опис динаміки взаємодії компонентів через діаграми послідовностей

Діаграми послідовностей (Sequence diagrams) [13] візуалізують динамічний аспект системи, описуючи логіку обміну повідомленнями між об'єктами, компонентами та зовнішніми акторами (рис. 2.7-2.11). В архітектурі розроблюваної системи виділено два ізольованих асинхронних канали взаємодії:

– канал вебінтерфейсу (REST API Architecture): реалізує класичну триланкову схему. Клієнтська SPA на базі фреймворку Angular взаємодіє через вебсервер Nginx з модулем обробки бізнес-логіки PHP-FPM (Symfony Framework), який, у свою чергу, використовує СУБД MySQL як транзакційне сховище;

– канал інтелектуальної взаємодії MCP: забезпечує підключення великих мовних моделей. Клієнт `Claude Desktop` ініціює запуск консольної команди Symfony (`McpServeCommand`) через потоки введення-виведення `stdio`. Обмін повідомленнями відбувається у форматі асинхронних JSON-RPC пакетів [19].

Обидва канали є повністю незалежними на рівні представлення, проте використовують єдиний сервісний шар бізнес-логіки та спільну систему токенізації (JSON Web Tokens) [18] для автентифікації користувачів.

Процедура входу в систему (рис. 2.7) вимагає перевірку облікових даних, захист від несанкціонованого доступу та генерацію криптографічних ключів сесії.

Для забезпечення безперервного користувацького досвіду, UX (User Experience), та захисту каналів зв'язку впроваджено паттерн безшумного оновлення прострочених ключів доступу без розриву поточної сесії (рис. 2.8).

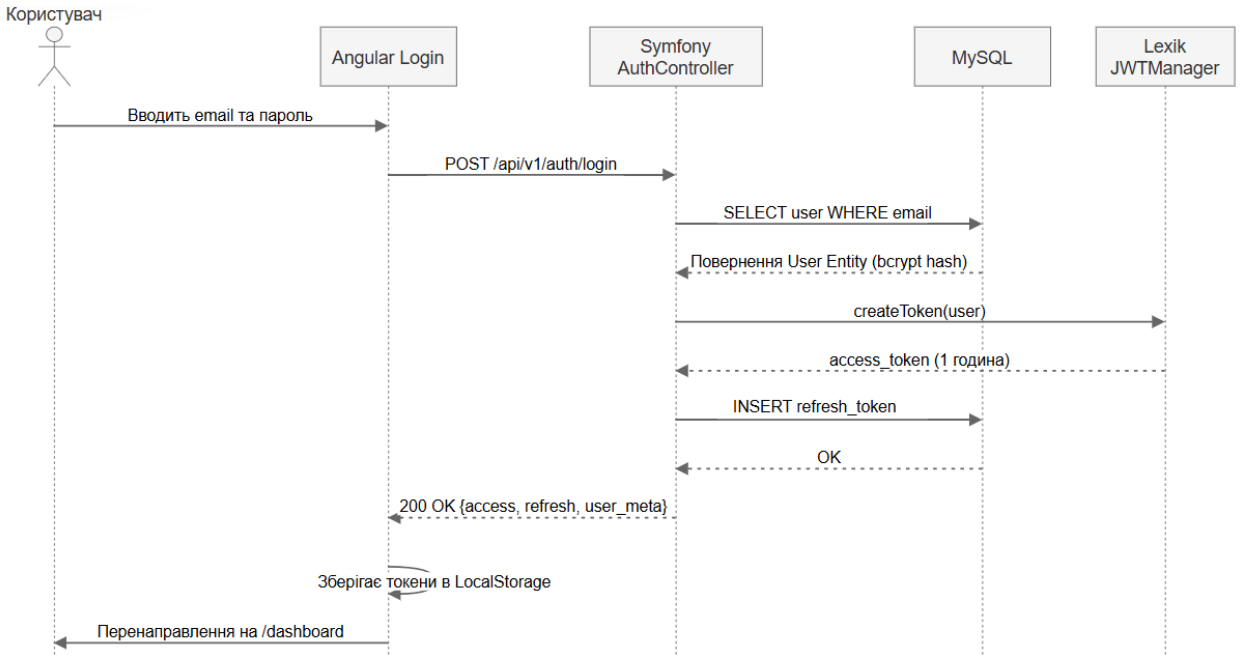

Рисунок 2.7 – Динаміка автентифікації користувача (Login Sequence)

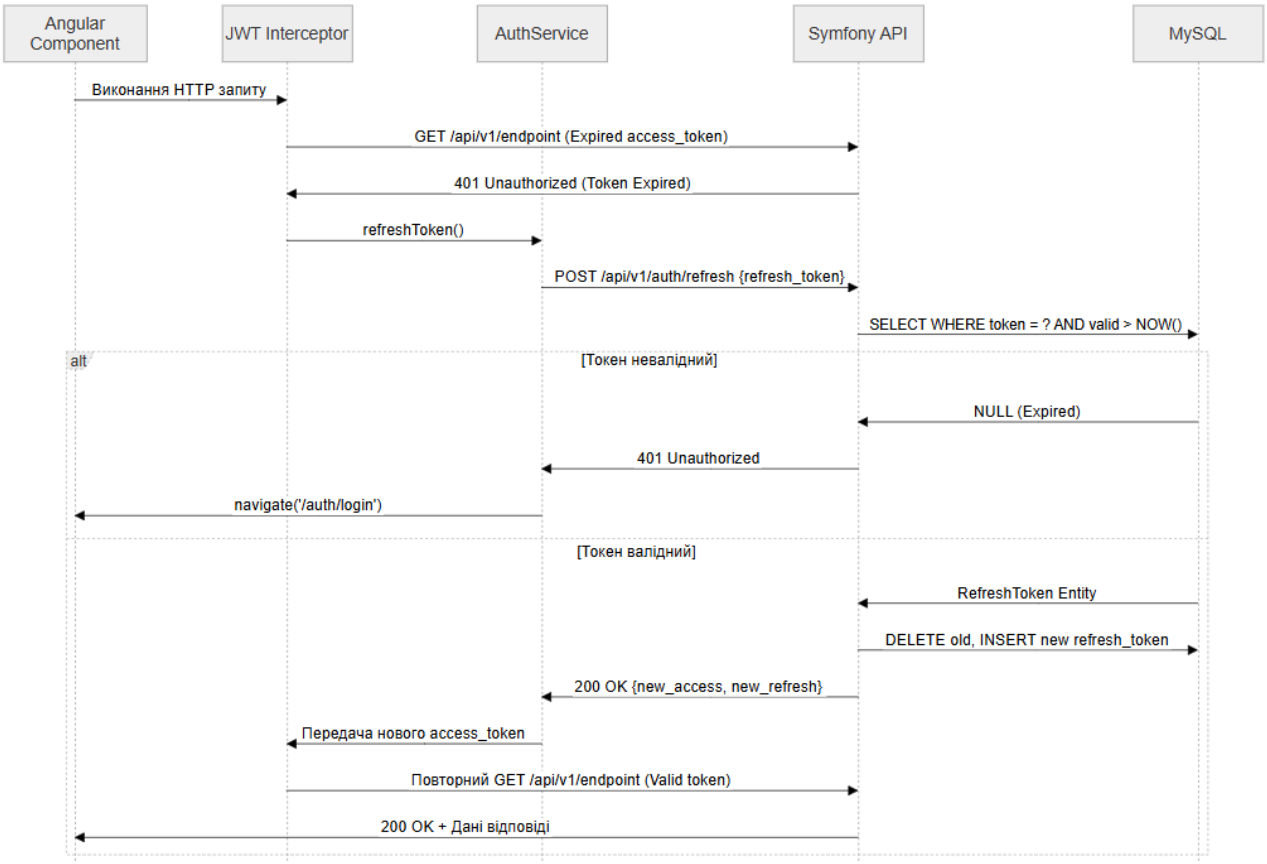

Рисунок 2.8 – Механізм автоматичного оновлення сесії (Token Refresh)

Процес (рис. 2.9) відображає синхронізацію стану клієнтського інтерфейсу з реляційним сховищем даних при реєстрації нових фінансових витрат або доходів.

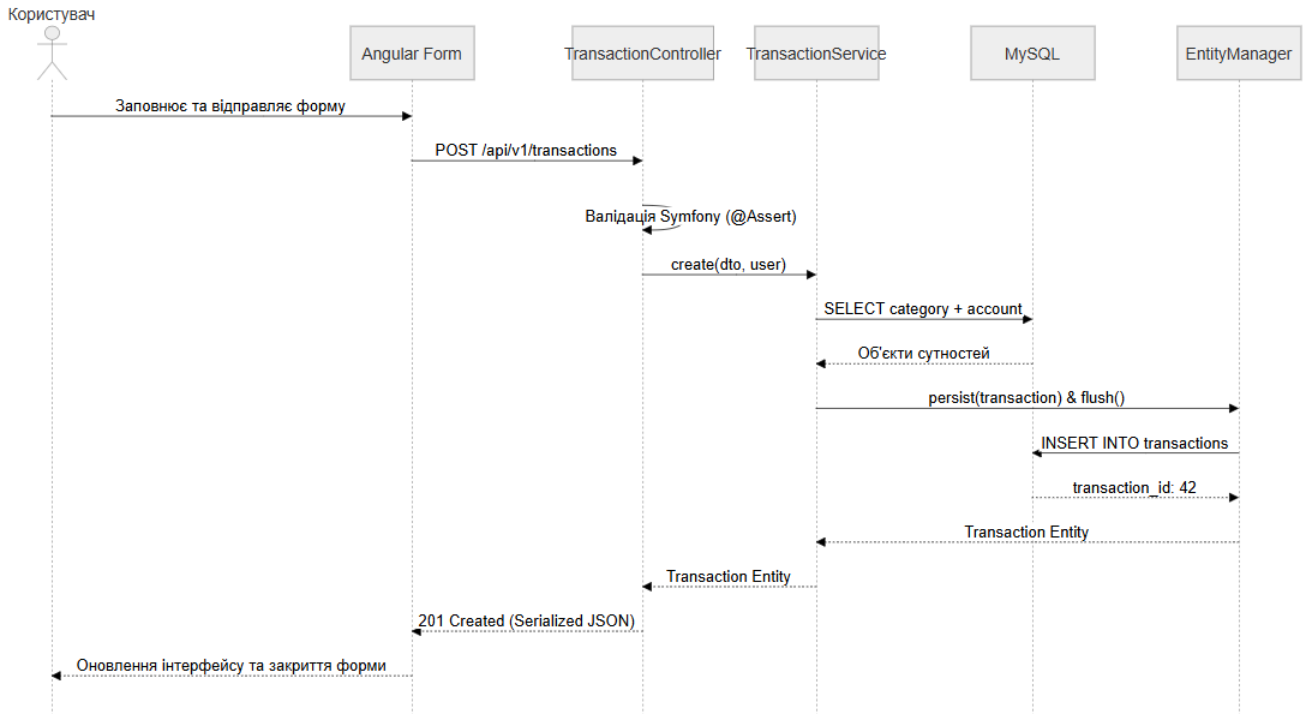

Рисунок 2.9 – Створення транзакції через веб-інтерфейс (Web Transaction)

Обробка ШІ-запиту через MCP протокол описана на рисунку 10.

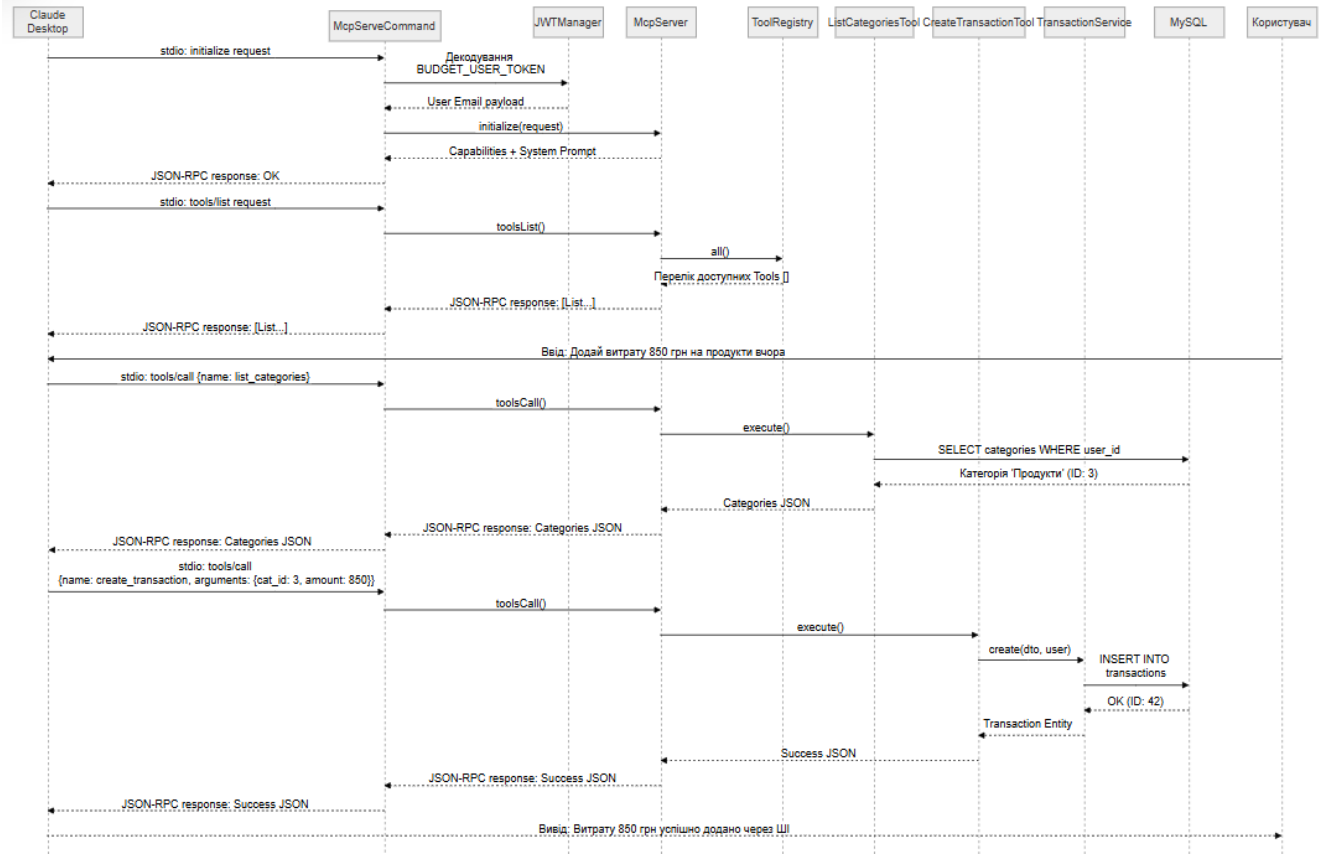

Рисунок 2.10 – Введення транзакцій за допомогою AI через MCP (Claude Desktop)

Найбільш складна частина динаміки системи – це інтелектуальна взаємодія, де API виступає як MCP-сервер, а саме обробка природної мови. Коли користувач надсилає запит, мовна модель ініціює виклик інструменту (`tools/call`). Сервер ідентифікує необхідний сервіс (наприклад, `ReportService`), отримує агреговані дані з таблиці `transactions` (GROUP BY `category_id`) і повертає JSON-результат моделі. Усі етапи виклику інструментів логуються в таблиці `ai_chat_messages` з ролями `tool_call` та `tool_result`

Сценарій описаний на рисунку 2.10 описує специфіку асинхронної обробки природно-мовною моделлю структурованих інструментів MCP-сервера. Процес розділений на етап рукостискання (Handshake) та етап виконання виклику функції (Tool Call).

Формування дашборду аналітики (рис. 2.11) використовує концепцію паралельних запитів для прискорення завантаження важких агрегаційних даних.

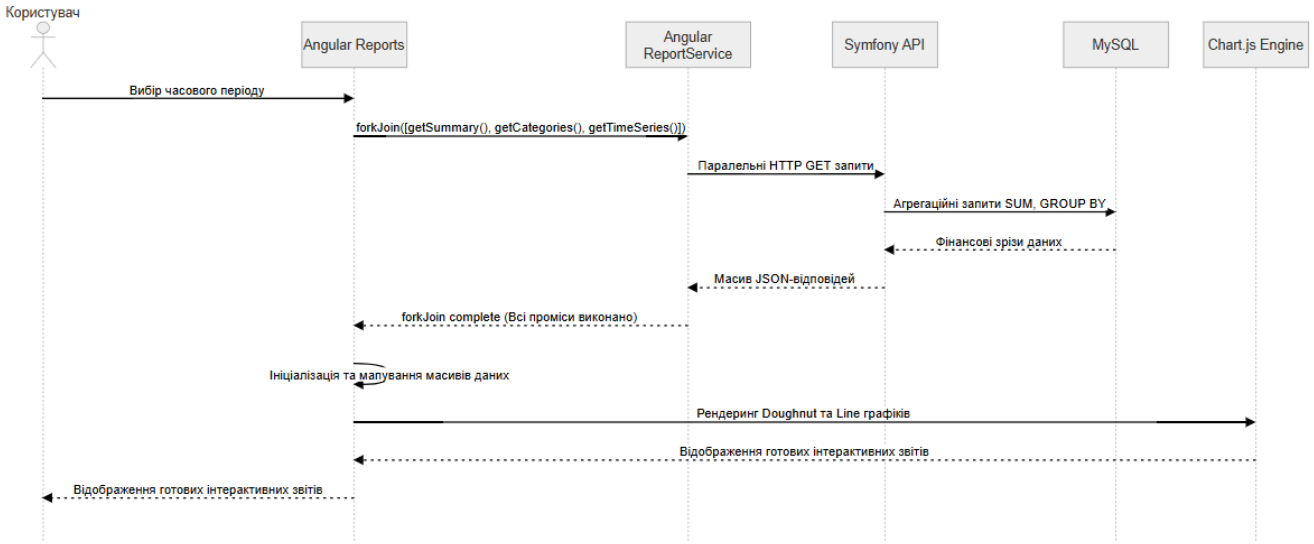

Рисунок 2.11 – Асинхронний запит аналітичної звітності (Report)

Для забезпечення чистоти архітектури та дотримання сучасних практик веброзробки в системі реалізовано інженерні патерни проектування, класифіковані у таблиці 2.3.

Таблиця 2.3 – Патерни проєктування динамічної взаємодії компонентів

Назва патерну	*Область практичного застосування*	*Архітектурна роль у системі*
Interceptor Chain	Angular HTTP Client Module	Реалізує наскрізну функціональність (Cross-Cutting Concern). Автоматично ін'єктує заголовок `Authorization: Bearer` у кожен вихідний запит та перехоплює помилки 401 для тригеру механізму Token Refresh [20].
Observer (BehaviorSubject)	Angular State Management (`AuthService`)	Дозволяє реактивно сповіщати незалежні UI-компоненти застосунку (навігаційну панель, бічне меню, роутер) про динамічну зміну стану авторизації поточного користувача [21].
forkJoin Pattern	Angular Reports Component	Оптимізує мережеве навантаження. Забезпечує паралельне виконання пулу HTTP-запитів до API, блокуючи фінальний рендеринг графіків Chart.js [22] до моменту отримання всіх відповідей.
ShareReplay	Angular Data Services	Кешує та перевикористовує результати статичних REST-запитів (списки валют, системних категорій). Запобігає надлишковому дублюванню мережевих запитів при повторній ініціалізації компонентів.
Strategy (Стратегія)	Symfony MCP ToolRegistry	Ізолює логіку виконання ШІ-інструментів. Кожен окремий інструмент (`Tool`) є незалежною стратегією, яка виконує спільний контракт `ToolInterface`, спрощуючи додавання нових функцій [23].
Command (Команда)	Symfony Console Line Infrastructure	Перетворює стандартний CLI-інтерфейс Symfony на постійно запущений фоновий процес (Loop), який прослуховує потік `stdin`, парсить JSON-RPC пакети та делегує їх ядру сервера.

2.4 Моделювання станів асинхронних процесів та обробки AI-запитів

Для забезпечення цілісності даних та коректної обробки бізнес-логіки в умовах асинхронної взаємодії, необхідно формалізувати стани ключових об'єктів системи. Використання діаграм станів (State Machine Diagrams) [13, 14] та моделювання життєвих циклів критичних сутностей дозволяє зафіксувати правила переходу бізнес-об'єктів між станами під впливом зовнішніх подій, мінімізуючи ризики виникнення некоректних або конфліктних конфігурацій даних у СУБД.

Обліковий запис є базовою сутністю безпеки застосунку. Його життєвий цикл охоплює етапи від первинної реєстрації до потенційного видалення чи блокування, регулюючи рівень доступу до фінансового ядра (рис. 2.12).

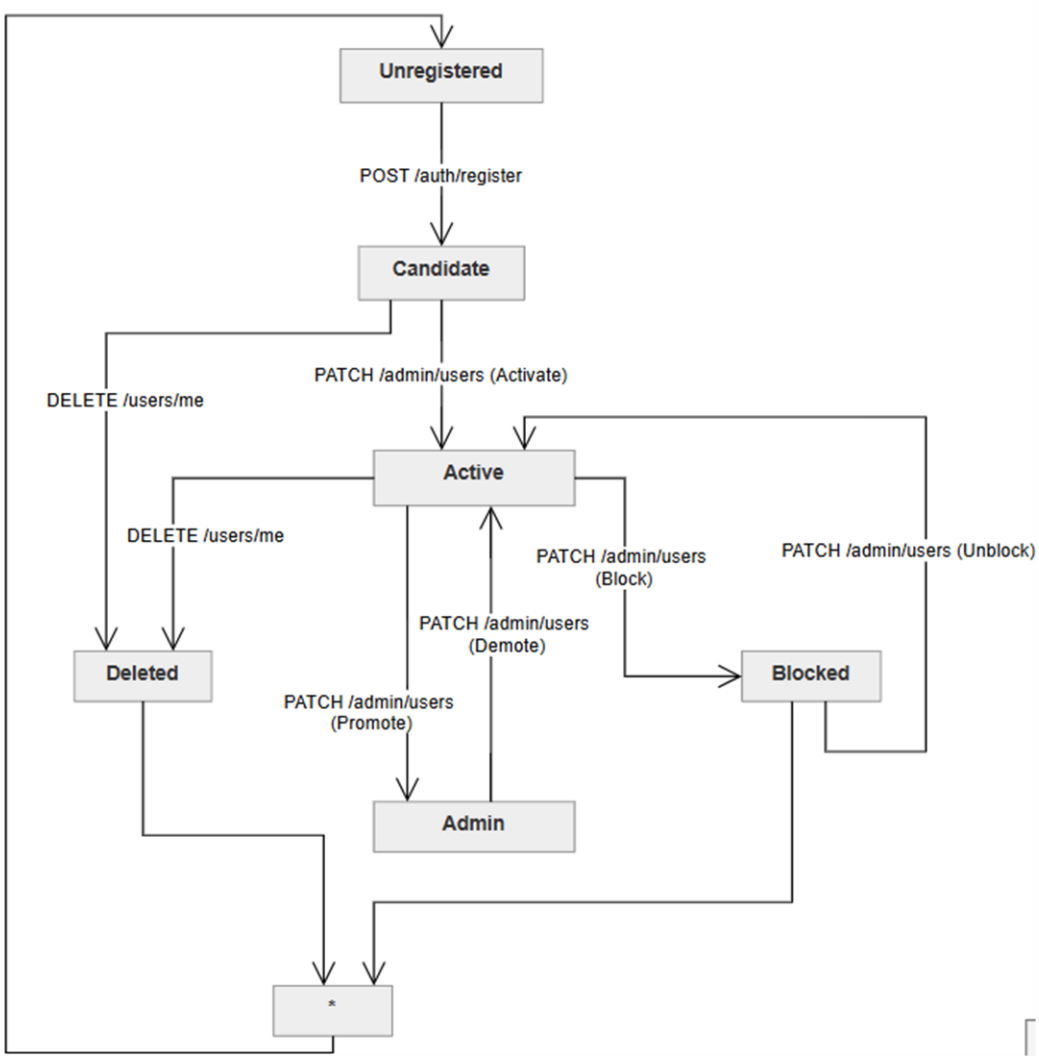

Рисунок 2.12 – Стани облікового запису користувача

Опис станів та матриця операцій:

– `Unregistered`: початкова точка. Користувач не має активного запису. Доступні лише методи реєстрації та авторизації;

– `Candidate`: проміжний етап очікування верифікації адміністратором. Згенерований JWT-токен дає право лише на перегляд сторінки `/pending`. Спроби запитів до фінансових довідників блокуються безпековим шаром;

– `Active`: стабільний робочий стан. Користувач володіє повними правами на ведення бюджету;

– `Admin`: максимальний рівень привілеїв, що включає модерацію інших користувачів системи;

– `Blocked`: ізольований стан. Користувачу відмовляється у генерації нових токенів, а діючі сесії повертають HTTP-код `403 Forbidden`.

Життєвий цикл сесії клієнта описує логіку взаємодії короткоживучих ключів доступу (`access_token`) та довгоживучих ключів поновлення (`refresh_token`) та зображено на рисунку 2.13.

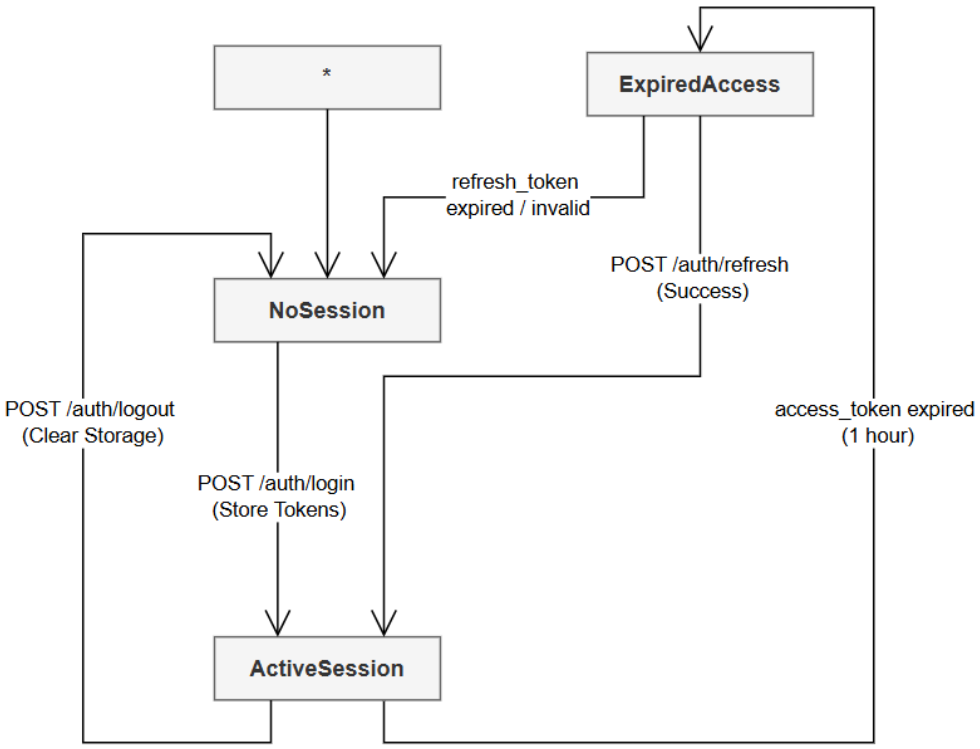

Рисунок 2.13 – Стани JWT-токенів (Автентифікаційна сесія)

У стані `ExpiredAccess` клієнтський перехоплювач `jwtInterceptor` заморожує вихідний пул запитів, ставить їх у чергу (`Retrying queue`) та ініціює асинхронний запит поновлення сесії [14]. Якщо `refresh_token` валідний, сесія повертається у стан `ActiveSession`, а затримані HTTP-запити виконуються прозоро для користувача.

На відміну від HTTP-запитів, сесія протоколу MCP є однонитковим фоновим процесом ОС (`stdio stream`). Вона працює в режимі постійного прослуховування потоку введення і послідовно обробляє JSON-RPC повідомлення (рис. 2.14).

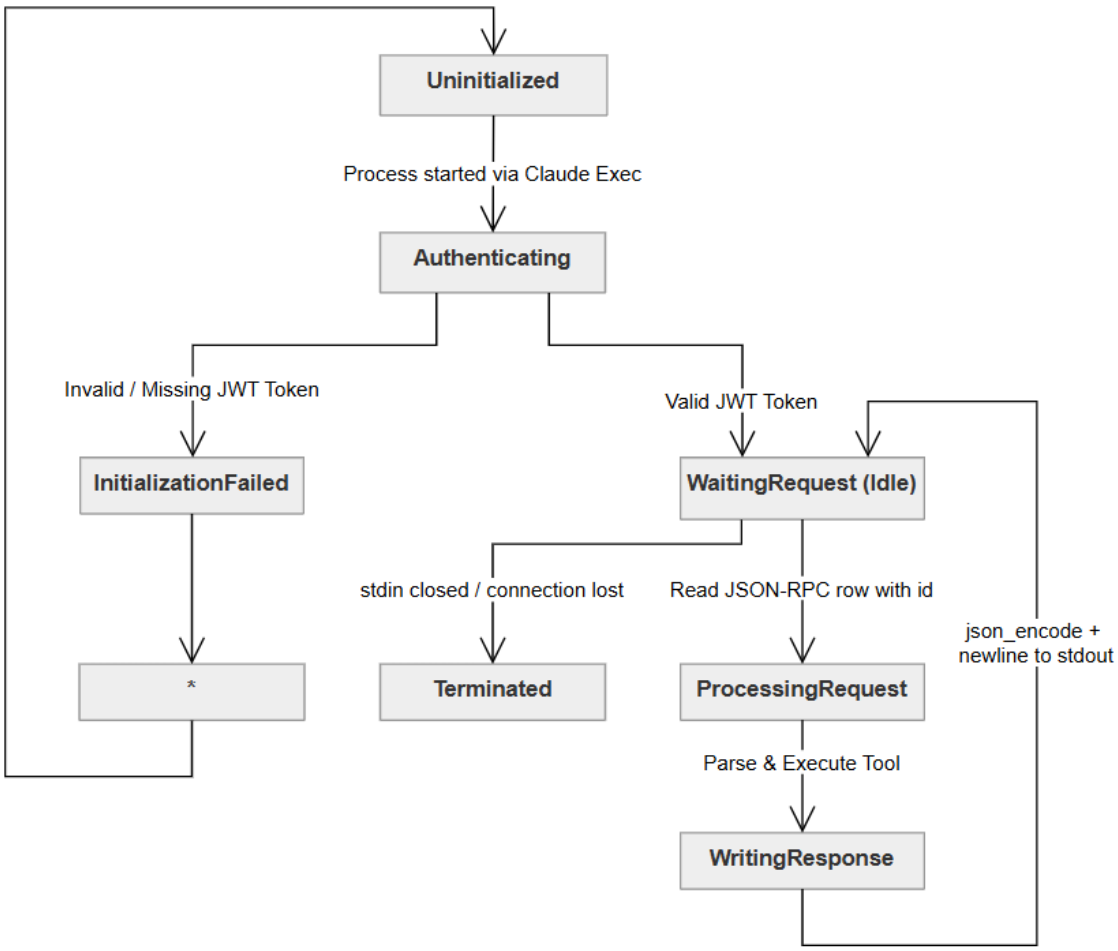

Рисунок 2.14 – Стани MCP-сесії (Claude Desktop ↔ Symfony)

Архітектурні особливості функціонування MCP-сесії:

– однонитковість та блокування: перебуваючи у стані `WaitingRequest (Idle)`, процес заблокований функцією `fgets()`. Отримання повідомлення переводить сесію у стан `ProcessingRequest`. Наступний запит не може бути прочитаний, поки поточний не пройде фазу `WritingResponse`;

– фільтрація нотифікацій: якщо вхідний JSON-RPC пакет не містить поля `id`, система кваліфікує його як асинхронне сповіщення (Notification) і повертає процес у стан `WaitingRequest` без надсилання відповіді у `stdout`;

– ізоляція помилок виконання: винятки (`Exceptions`), що виникають під час роботи інструментів, перехоплюються ядром MCP-сервера. Процес не переривається з кодом `5xx`, а повертає JSON-відповідь зі статусом 200, яка містить маркер `isError: true` та детальний опис проблеми. Це дає змогу ШІ-агенту інтерпретувати помилку і скоригувати свої дії.

Життєвий цикл фінансового плану відображає ступінь освоєння закладених грошових лімітів та динамічно змінює колірні маркери інтерфейсу на основі математичного співвідношення фактичних витрат до запланованих $Fact/Plan$ (рис. 2.15).

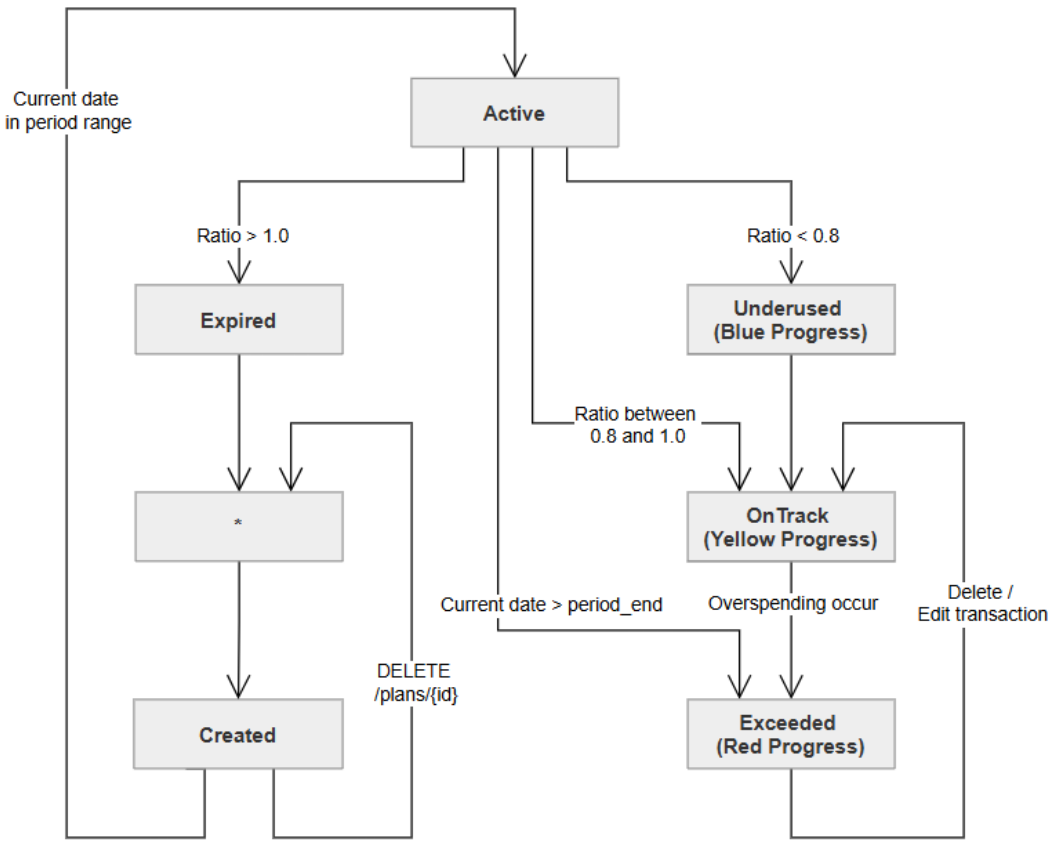

Рисунок 2.15 – Стани плану бюджету та виконання лімітів

Для ковзних (rolling) планів, які динамічно відраховуються від поточної дати, класичне порівняння $Fact/Plan$ є неефективним на початку звітного періоду. Впроваджений алгоритм рівномірного масштабування пропорційно зменшує планову суму відповідно до прожитих днів.

Якщо загальний план на місяць (наприклад, травень, $T_{total} = 31$ день) становить $P_{total} = 3000$ грн, а аналіз виконується на 17-й день місяця ($T_{past} = 17$), то масштабоване значення планового ліміту розраховується за формулою:

$$P_{scaled} = P_{total} * \left(\frac{T_{past}}{T_{total}}\right) = 3000 * \left(\frac{17}{31}\right) \approx 1645.16 \text{ грн.} \quad (2.1)$$

У цьому випадку фактичні витрати розміром 1000 грн порівнюються не з повною сумою 3000 грн (що показало б хибне перевиконання плану – всього 33%), а з масштабованим лімітом $P_{scaled} = (\frac{1000}{1645.16} \approx 61\%)$. Це забезпечує точне відображення стану `OnTrack` або `Underused` на будь-якому етапі часового горизонту.

Модуль колективного бюджетування використовує систему токенізованих запрошень для безпечного делегування доступу (рис. 2.16). Життєвий цикл інвайту обмежений часовим маркером (Time-To-Live).

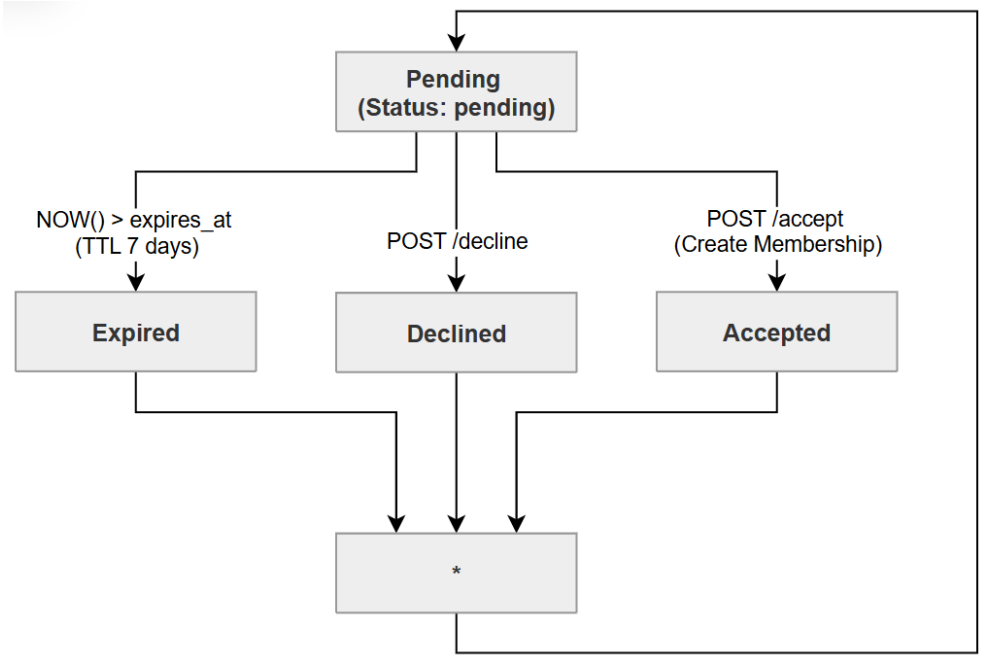

Рисунок 2.16 – Стани запрошення до спільної групи

Запрошення генерує унікальний криптографічний токен (case-insensitive UUID). Перехід у стан `Expired` контролюється як на рівні фонових завдань (cron-

скрипти очищення), так і динамічно – під час спроби активації токена через перевірку умови `SystemTime > Invitation.expiresAt`. Перехід у стан `Accepted` автоматично тригерує створення сутності `GroupMembership` з базовою роллю `member`.

2.5 Проєктування реляційної моделі бази даних та опис зв'язків між сутностями

На основі розробленої діаграми класів та бізнес-сценаріїв було спроєктовано реляційну модель бази даних (рис. 2.17). Вибір СУБД MySQL 8.x [16] зумовлений підтримкою JSON-типів для MCP-повідомлень, високою швидкістю роботи з індексами та повною сумісністю з Doctrine ORM.

Для забезпечення цілісності, відсутності аномалій модифікації та оптимальної швидкодії обробки великих масивів фінансових транзакцій було впроваджено такі інженерні підходи:

- нормалізація до третьої нормальної форми (3NF): усі спроєктовані таблиці приведені до 1NF, 2NF та 3NF. Усунено всі неключові та транзитивні залежності атрибутів. Кожен інформаційний запис містить характеристики лише тієї сутності, яку він безпосередньо описує, що мінімізує надмірність даних;
- вибір стратегії первинних ключів (Surrogate Keys): замість використання громіздких UUID-рядків у ролі первинних ключів (PK), які суттєво уповільнюють індексацію та JOIN-операції в MySQL, обрано цілочисельні сурогатні ключі типу `INT UNSIGNED` з атрибутом автоматичного приросту (`AUTO_INCREMENT`). Це оптимізує споживання оперативної пам'яті (Buffer Pool) [24] та прискорює пошук за індексами;
- декларативна цілісність посилань (Foreign Keys): усі міжтабличні зв'язки жорстко зафіксовані на рівні рушія БД `InnoDB` за допомогою обмежень зовнішніх ключів (FK). Для запобігання появі «сирітських» записів визначено суворі каскадні стратегії (`CASCADE DELETE` для слабких сутностей та `SET NULL` / `RESTRICT` для сильних бізнес-об'єктів) [25];

– хронологічний аудит даних: таблиці, що відображають операційну діяльність (`transactions`, `users`, `plans`, `groups`, `group_invitations`), оснащені обов'язковими полями тимчасових міток (`created_at / transacted_at`) типу `DATETIME` для ведення точного ієрархічного аналізу та побудови часових рядів алгоритмами ШІ;

– фізичне видалення даних: з огляду на вимоги суворої фінансової звітності та консистентності калькуляцій планів у поточній версії відхилено патерн м'якого видалення (Soft Delete). Усі операції видалення є фізичними (`Hard Delete`) із транзакційним перерахунком пов'язаних сум.

Для візуалізації реляційної структури та шляхів доступу до даних схему бази даних спроєктовано у текстовій нотації yUML. Схема відображає ключові таблиці, поля ідентифікаторів та типи кардинальності зв'язків (рис. 2.17).

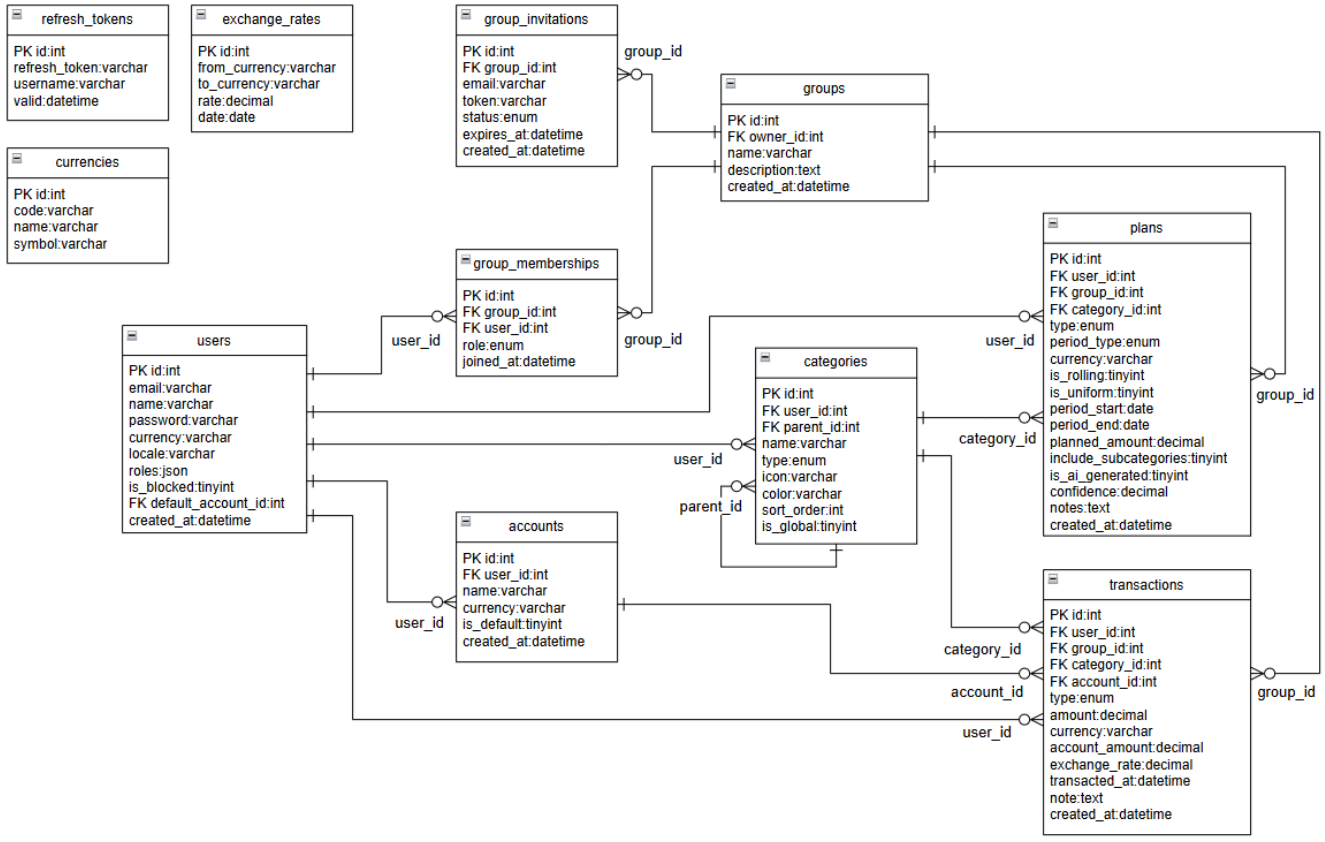

Рисунок 2.17 – Логічна ER-діаграма бази даних

Для детального аналізу каскадних операцій та обмежень цілісності на рівні бази даних розроблено зведену матрицю зовнішніх ключів (табл. 2.4).

Таблиця 2.4 – Специфікація обмежень цілісності реляційних зв'язків

Вихідна таблиця (Маніпулятор)	*Цільова таблиця*	*Тип зв'язку*	*Поле зовнішнього ключа (FK)*	*Поведінка при видаленні (ON DELETE)*
accounts	users	Many-to-One	user_id	CASCADE
transactions	users	Many-to-One	user_id	RESTRICT
transactions	accounts	Many-to-One	account_id	SET NULL
transactions	categories	Many-to-One	category_id	RESTRICT
transactions	groups	Many-to-One	group_id	SET NULL
categories	users	Many-to-One	user_id	SET NULL
categories	categories	Many-to-One	parent_id	SET NULL
plans	users	Many-to-One	user_id	RESTRICT
plans	categories	Many-to-One	category_id	RESTRICT
plans	groups	Many-to-One	group_id	SET NULL
groups	users	Many-to-One	owner_id	RESTRICT
group_memberships	groups	Many-to-One	group_id	CASCADE
group_memberships	users	Many-to-One	user_id	CASCADE
group_invitations	groups	Many-to-One	group_id	CASCADE
users (Self-reference)	accounts	One-to-Many	default_account_id	SET NULL

Спроєктована реляційна модель бази даних забезпечує повну функціональну ізоляцію та цілісність даних інтелектуального фінансового застосунку. Ядром архітектури безпеки виступає таблиця `users` із впровадженим JSON-масивом рольової моделі та зв'язком із таблицею `refresh_tokens`, що гарантує надійне довгострокове збереження автентифікаційних JWT-сесій [14]. Фінансова логіка системи розподілена між рахунками (`accounts`) та транзакціями (`transactions`). При цьому з метою оптимізації швидкодії та запобігання аномаліям розсинхронізації, поточний баланс рахунків не зберігається фізично, а розраховується СКБД динамічно як агрегована різниця між доходами та витратами.

Модуль класифікації реалізовано через таблицю `categories` за допомогою рекурсивного зв'язку `parent_id`, що дозволяє будувати дворівневі ієрархії категорій. Важливою інженерною особливістю є поділ категорій на системні, де `user_id` IS NULL, що робить їх глобально доступними та персоналізовані користувацькі. Центральна високоінтенсивна таблиця `transactions` підтримує складну мультивалютну логіку: у випадку розбіжності між валютою операції та рахунку, система автоматично фіксує крос-курс та конвертовану суму в полях `exchange_rate` та `account_amount`, спираючись на історичні дані з таблиці `exchange_rates`.

Колективний фінансовий менеджмент організовано через таблицю груп (`groups`), яка пов'язана з користувачами відношенням багатьох до багатьох через проміжну сутність `group_memberships`, де фіксуються індивідуальні рівні доступу, а безпечне розширення складу груп контролюється через токенізовані інвайти в `group_invitations`. Нарешті, таблиця `plans` забезпечує гнучке довгострокове планування, зберігаючи не лише часові горизонти лімітів, а й специфічні метрики штучного інтелекту (`is_ai_generated`, `confidence`), що дозволяє прогностичному модулю ШІ через MCP-інтерфейс безперешкодно зчитувати, аналізувати та коригувати фінансові стратегії користувача.

Висновки до розділу 2

У другому розділі виконано архітектурне та системне проєктування інтелектуальної системи управління персональними фінансами. За допомогою інструментів об'єктно-орієнтованого моделювання формалізовано функціональні межі системи та розроблено діаграму прецедентів, яка охоплює понад 40 сценаріїв взаємодії для п'яти акторів. Ключовою особливістю проєкту є дублювання прецеденту створення транзакцій як звичайним користувачем через вебінтерфейс (REST API), так і зовнішнім AI-агентом Claude Desktop через інфраструктуру MCP, що використовують єдиний сервісний шар.

Статичну структуру та бізнес-логіку програмного продукту зафіксовано на рівні діаграми класів, декомпонованої на рівень предметної області (Domain Layer) із 11 базових сутностей та рівень інфраструктури (Service & MCP Layer). Динаміку асинхронної взаємодії компонентів деталізовано за допомогою діаграм послідовностей, які описують процеси JWT-автентифікації, безшумного оновлення сесії, паралельного формування аналітичних звітів через forkJoin, а також складний механізм інтерпретації мовних запитів у структуровані JSON-RPC команди.

Життєві цикли та правила переходу системних об'єктів під впливом зовнішніх подій описано за допомогою діаграм станів. Змодельовано автомати поведінки облікових записів, JWT-сесій, а також логіку функціонування MCP-сервера в однонитковому потоці stdio. Окремо математично обґрунтовано стани виконання бюджетних лімітів для ковзних планів за допомогою алгоритму рівномірного масштабування суми пропорційно прожитим дням місяця.

На основі розроблених UML-моделей спроєктовано реляційну модель бази даних у третій нормальній формі (3NF), що складається з 11 взаємопов'язаних таблиць під управлінням СУБД MySQL. Визначено специфікації полів, типи цілісності даних, індекси та зведену матрицю із 14 каскадних зв'язків, а також зафіксовано покрокову стратегію розгортання схеми через Doctrine Migrations.

3 РЕАЛІЗАЦІЯ СЕРВЕРНОЇ ЧАСТИНИ ТА ІНТЕЛЕКТУАЛЬНИХ МОДУЛІВ

3.1 Розробка ядра системи на Symfony: автентифікація (JWT) та безпека API

Програмна реалізація серверної частини системи базується на сучасних інженерних підходах і побудована з використанням фреймворку Symfony 7.4 та інтерпретатора PHP 8.2+. Відповідно до розробленої раніше п'ятирівневої декомпозиції Clean Architecture [9, 10] , класи системи чітко розділені за своїми обов'язками, що унеможливлює сильну зв'язність компонентів.

Розподіл відповідальності між основними компонентами коду ядра наведено на рисунку 3.1.

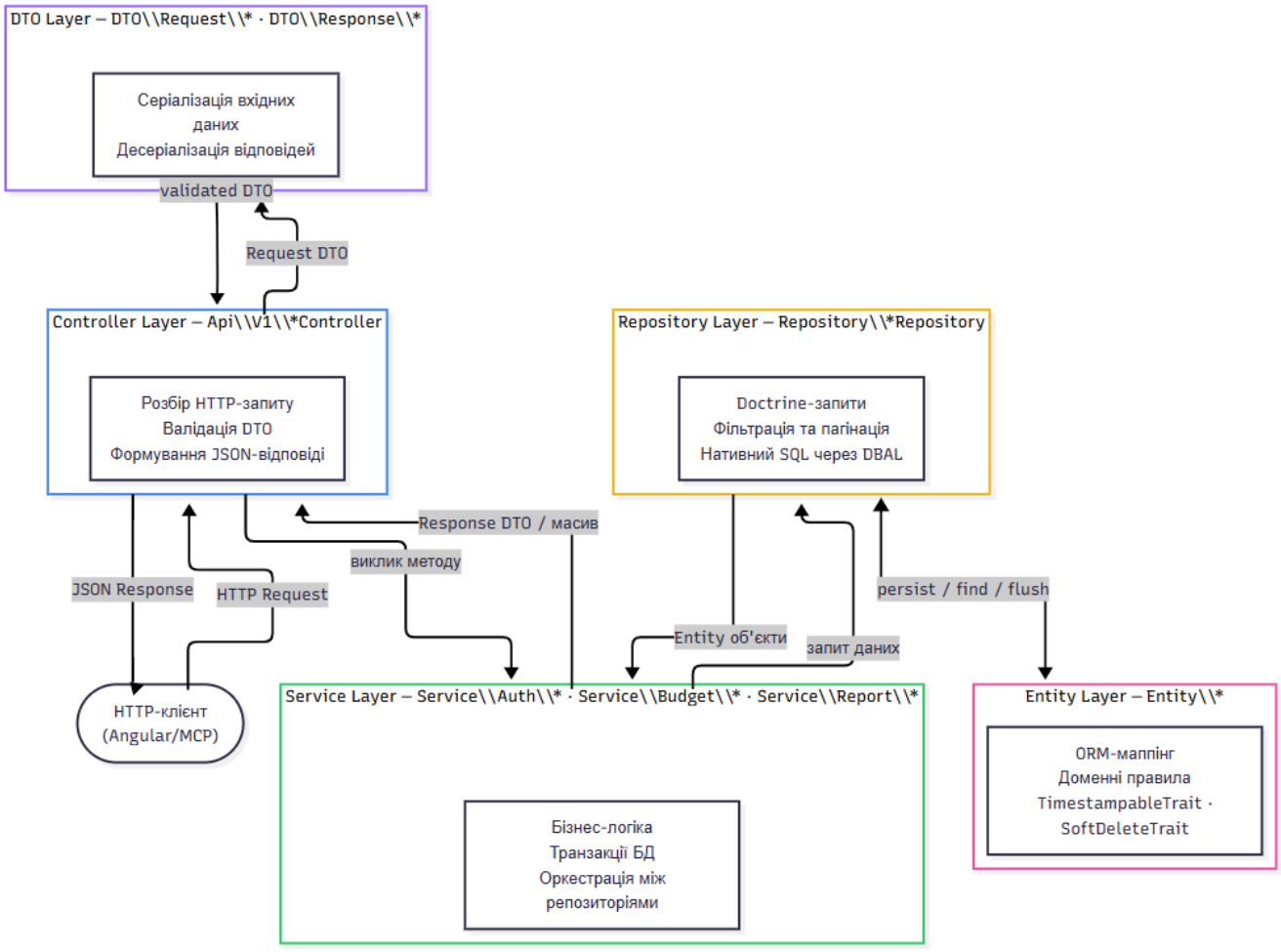

Рисунок 3.1 – Розподіл відповідальності у шарах серверної архітектури

Для забезпечення безпеки stateless-архітектури REST API реалізовано комбінований механізм автентифікації JWT Access Token + Refresh Token (див. додаток А) [18, 26]. На відміну від класичних сесій, кожен запит містить заголовок `Authorization: Bearer <token>`. Програмне налаштування компонента `security.yaml` [27] реалізує розроблену ієрархію акторів (Гість, Кандидат, Користувач, Адміністратор) за допомогою конфігураційної матриці, наведеної в таблиці 3.1.

Таблиця 3.1 – Специфікація правил доступу до підсистем API

Маршрут (URL Pattern)	*Дозволені ролі (Ієрархія)*	*Тип автентифікації*	*Функціональний контекст*
`^/api/v1/auth`	`PUBLIC_ACCESS` (Усі актори)	`Відсутній` (Анонімно)	Реєстрація нового акаунту, вхід та JWT-автентифікація.
`^/api/v1/admin`	`ROLE_ADMIN`	`Stateless JWT`	Керування життєвим циклом акаунтів, модерація, блокування.
`^/api/v1`	`ROLE_USER, ROLE_ADMIN`	`Stateless JWT`	Особистий та колективний фінансовий менеджмент, аналітика.

З метою запобігання дублюванню коду часових міток у площині ORM-мапування крім фіксації часу, впроваджено повторно використовувані PHP-трейти для контролю видалення даних та уніфікації відповідей шлюзу. Їх специфікацію наведено в таблиці 3.2.

Таблиця 3.2 – Функціональна специфікація трейтів та модулів ядра

Компонент / Трейт коду	*Цільовий атрибут / Метод*	*Логіка тригера (Подія)*	*Функціональна роль у системі безпеки*
`Timestampable Trait`	`createdAt, updatedAt`	ORM `PrePersist, PreUpdate`	Автоматична хронологічна фіксація часу створення та модифікації об'єктів.
`SoftDeleteTrait`	`deletedAt (DateTime|null)`	Метод `softDelete()`	Логічне (м'яке) видалення користувачів, транзакцій та рахунків без фізичного знищення рядків для збереження цілісності звітів.
`ExceptionSubscriber`	`Метод onKernelException()`	Перехоплення системних `Exception`	Уніфікація помилок API. Перетворює доменні винятки (AuthException тощо) у JSON-відповіді формату {"detail": "..."} з відповідними HTTP-кодами (401, 403, 404).

3.2 Реалізація бізнес-логіки управління транзакціями та колективними бюджетами

Центральною високоінтенсивною частиною фінансового ядра є `TransactionService`. Під час реєстрації нової фінансової операції сервіс виконує багаторівневу перевірку доменних правил, включаючи валідацію доступності категорій та верифікацію членства в групі у випадку колективного обліку.

Особливе місце посідає програмна реалізація ієрархічного алгоритму [28] визначення суми у валюті рахунку за умови мультивалютних операцій (рис. 3.2).

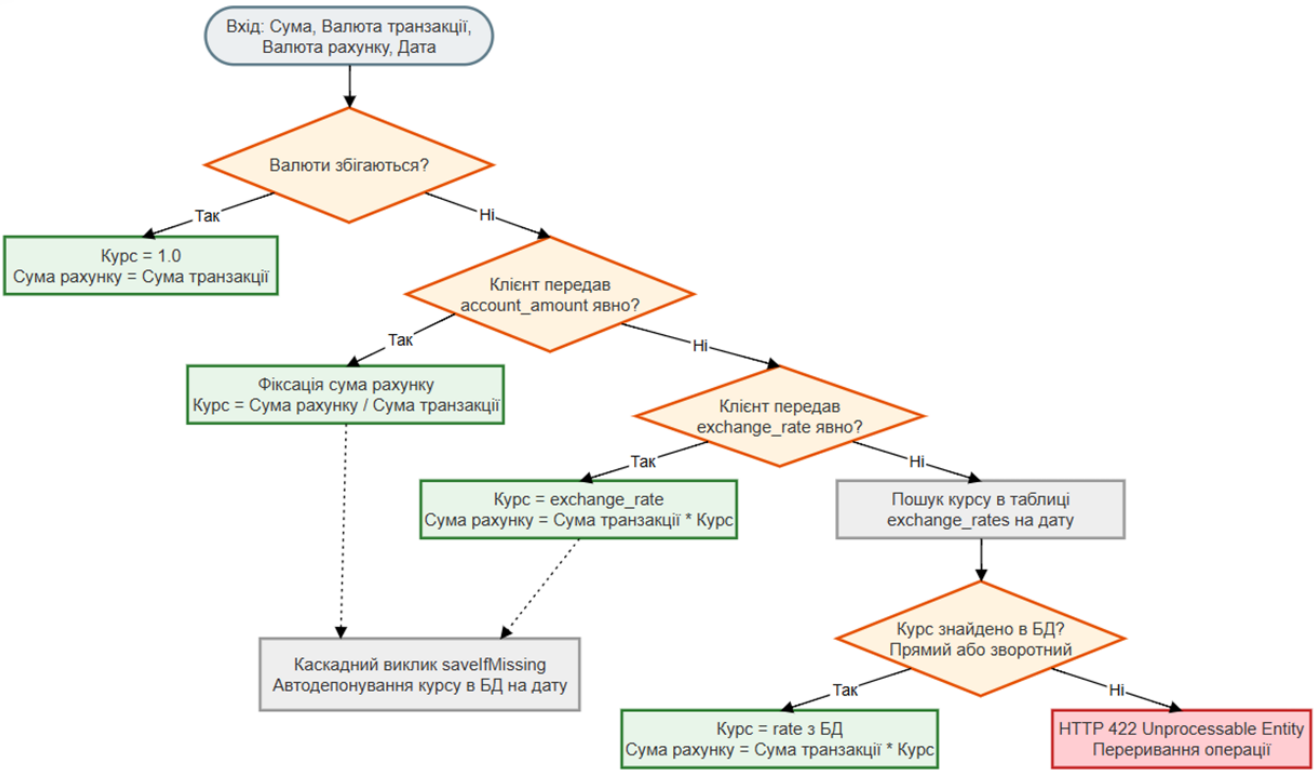

Рисунок 3.2 – Алгоритм визначення суми у валюті рахунку

Метод `resolveAmounts` реалізує п'ять рівнів пріоритету бізнес-правил:

– збіг валют: якщо валюта транзакції та валюта рахунку однакові, конвертація ігнорується, коефіцієнт (курс) дорівнює `1.0`;

– явна сума від клієнта: якщо DTO містить заповнене поле `accountAmount`, воно приймається як цільове, а курс розраховується як частка від ділення суми рахунку на суму транзакції;

– явний курс від клієнта: якщо передано `exchangeRate`, цільова сума рахунку обчислюється як добуток оригінальної суми на цей курс;

– пошук в історичному довіднику: сервіс звертається до `ExchangeRateService` для отримання актуального курсу з таблиці `exchange_rates` на дату операції;

– виняткова ситуація: у разі відсутності даних на всіх рівнях, система перериває транзакцію та генерує HTTP-код `422 Unprocessable Entity`.

Фрагменти коду реалізації описаного вище алгоритму можна знайти в додатоку А.

Модуль колективного бюджетування програмно забезпечує життєвий цикл токенізованих запрошень до груп (рис. 2.16). Процес виходу або видалення учасника з групи контролюється методом GroupService::removeMember(), який містить жорсткий бізнес-запобіжник: якщо цільовий користувач має роль єдиного власника (Owner), система блокує операцію та викидає ConflictException, запобігаючи появі «сирітських» груп без адміністратора.

Алгоритм програмної верифікації параметрів об'єкта запрошення під час його активації користувачем наведено в таблиці 3.3.

Таблиця 3.3 – Алгоритм програмної валідації стану `GroupInvitation`

Етап перевірки	*Цільовий атрибут сутності*	*Криве за умовою бізнес-правила*	*Результат невдалої перевірки*
I Перевірка статусу	`status`	Значення має дорівнювати строго `pending`	Перехід у стан `Rejected/Used`, блокування процесу.
II Часовий маркер	`expiresAt`	Поточний час системи < `Invitation.expiresAt`	Динамічний перехід у стан `Expired`, відмова в доступі.
III Верифікація токена	`token`	Case-insensitive відповідність UUID рядку	HTTP 404 Not Found (Запрошення не існує).

Наступним рівнем ієрархії моделювання бізнес-логіки є класифікація рахунків та внутрішня організація категорій. Вони забезпечують гнучкість персональних налаштувань користувача та деталізовані у таблиці 3.4.

Таблиця 3.4 – Структурна організація доменних довідників (Рахунки та Категорії)

Доменна сутність	*Тип / Рівень доступу*	*Розрахункова поведінка*	*Архітектурна специфіка*
Дефолтний рахунок	Персональний (`is_default = true`)	Автоматично створюється для актора під час реєстрації.	Використовується за замовчуванням, якщо у вхідному DTO транзакції не вказано `account_id`.
Глобальні категорії	Системний (`user_id IS NULL`)	Наповнюються міграцією при ініціалізації бази даних.	Доступні для використання всім акторам системи (напр. "Продукти", "Комунальні послуги").
Персональні категорії	Користувацький (`user_id` заповнено)	Створюються індивідуально користувачем у процесі роботи.	Відображаються та агрегуються лише в межах облікового запису власника. Підтримують вкладеність через поле `parent_id`.

Для управління лімітами бюджету реалізовано підтримку як фіксованих (anchored), так і ковзних (rolling) планів (див. додаток А). Захист від конфліктів конфігурацій реалізовано на рівні унікальних індексів СУБД, конфігурацію яких наведено в таблиці 3.5.

Таблиця 3.5 – Специфікація обмежень унікальності планів

Тип плану бюджету	*Рівень обмеження*	*Складений ключ / Конструкція*	*Функціональна роль*
Фіксований (Anchored)	База даних (MySQL 8.x)	`UNIQUE(user_id, category_id, period_type, period_start)`	Запобігає створенню дублікатів планів на один і той самий календарний період на рівні СКБД [24].
Ковзний (Rolling)	Бізнес-логіка (Service Layer)	Пошуковий запит через `PlanRepository::findActiveRolling()`	Забороняє наявність двох одночасних активних безстрокових стратегій для однієї категорії [16].

Керування сутностями здійснюється через уніфіковану матрицю REST-маршрутів (`GET|POST|PATCH|DELETE /api/v1/plans`).

3.3 Програмна реалізація MCP-сервера для інтеграції з мовними моделями

Інфраструктурний прошарок сервера контексту реалізує протокол MCP, забезпечуючи підключення сторонніх мовних моделей як інтелектуальних агентів. Програмний запуск сервера здійснюється через консольний компонент Symfony (`McpServeCommand`), що працює в режимі безперервного прослуховування стандартних потоків введення-виведення `stdin/stdout` у циклі `Loop`.

Взаємодія побудована на базі патерну `Стратегія` , де кожен інструмент ШІ імплементує єдиний контракт `ToolInterface` [29]. Динамічне розширення та реєстрація нових інструментів забезпечується механізмом тегування DI-контейнера Symfony. Концептуальну схему реєстрації та диспетчеризації JSON-RPC пакетів MCP-сервером представлено за допомогою діаграми на рисунку 3.3.

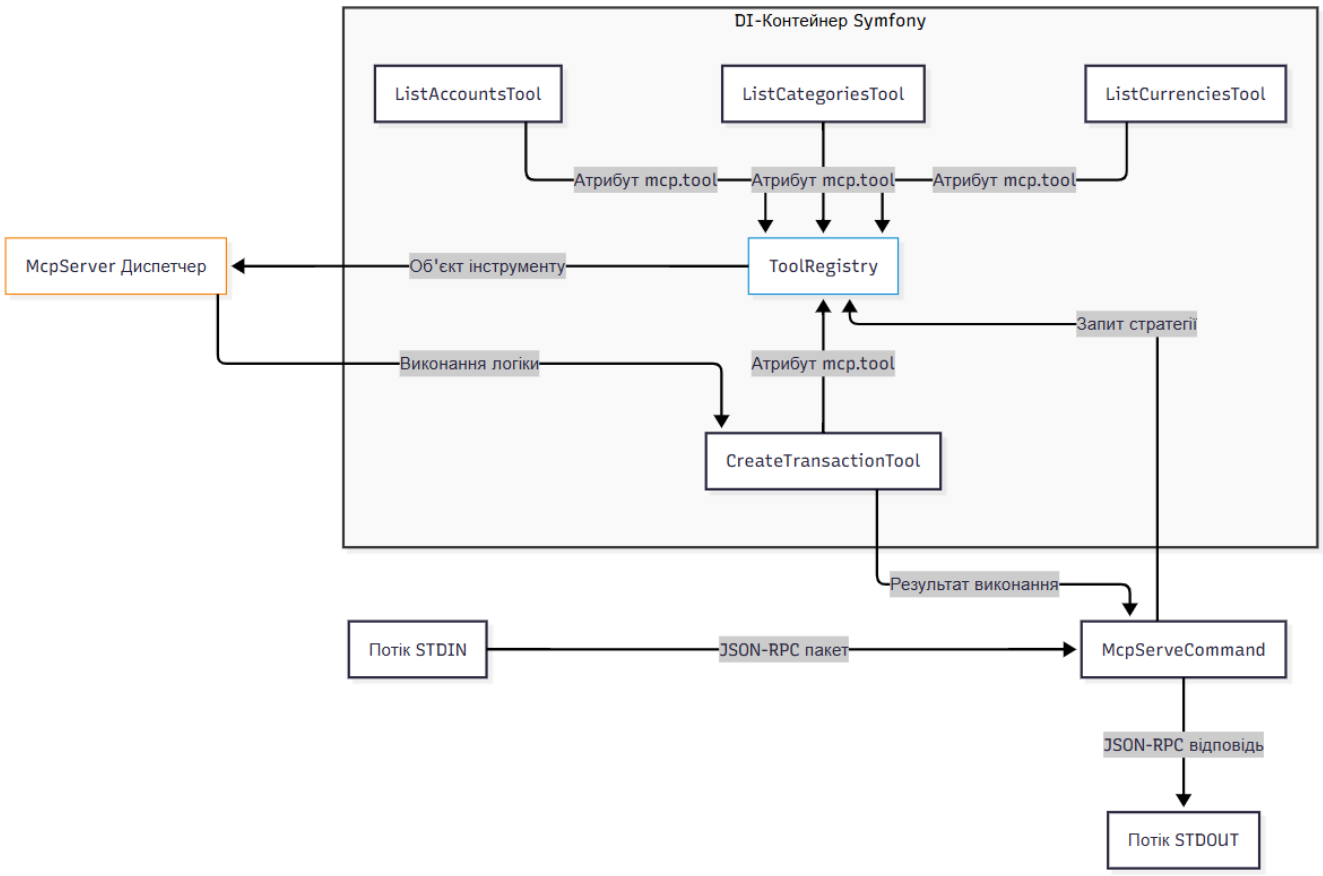

Рисунок 3.3 – Компонентна схема реєстрації та диспетчеризації інструментів MCP

Для автентифікації однопотокової MCP-сесії використовується ізольований канал: токен доступу передається через змінну оточення `BUDGET_USER_TOKEN` під час ініціалізації процесу клієнтом Claude Desktop.

Диспетчер McpServer бере на себе роль координатора запитів від ШІ та послідовно обробляє три базові методи протоколу:

– `initialize`: Етап рукостискання (Handshake). Сервер повертає метадані застосунку та динамічні системні інструкції для LLM, куди автоматично зашивається поточна дата у форматі `Y-m-d`. Це дозволяє моделі орієнтуватися в часі без додаткових запитів;

– `tools/list`: Декларує мовній моделі список усіх доступних інструментів та їхні JSON Schema для генерації аргументів;

– `tools/call`: Безпосередньо виконує бізнес-логіку вибраного інструменту. У разі виникнення помилок виконання, система кваліфікує їх, перехоплює винятки й повертає JSON-відповідь зі статусом 200, яка містить маркер `isError: true` та детальний опис проблеми, що дає змогу ШІ-агенту інтерпретувати помилку і скоригувати свої дії.

Кожен зареєстрований інструмент чітко декларує свою схему параметрів у форматі JSON Schema. Уніфікований список реалізованих інструментів, що надають контекст ШІ-агенту, та типи їхніх відповідей наведено в таблиці 3.6.

Таблиця 3.6 – Специфікація інструментів MCP-сервера

Назва *інструменту*	Обов'язкові параметри (JSON Schema)	Структура вихідних даних (JSON результату)
`list_accounts`	Порожній об'єкт `{}`	Масив зареєстрованих рахунків користувача: `[{id, name, currency, is_default}]`
`list_categories`	`{"type": "income|expense|both"}`	Список доступних дворівневих категорій: `[{id, name, parent_id, is_global}]`.
`list_currencies`	Порожній об'єкт `{}`	Довідник підтримуваних ISO 4217 кодів: `[{code, name, symbol}]`
`create_transaction`	`{type, amount, category_id, currency}`	Статус операції: `{"content": [{"type": "text", "text": "..."}], "isError": false}`

Типовий сценарій природно-мовної взаємодії між актором, AI-агентом (Claude Desktop) та ядром системи представлено на рисунку 3.4.

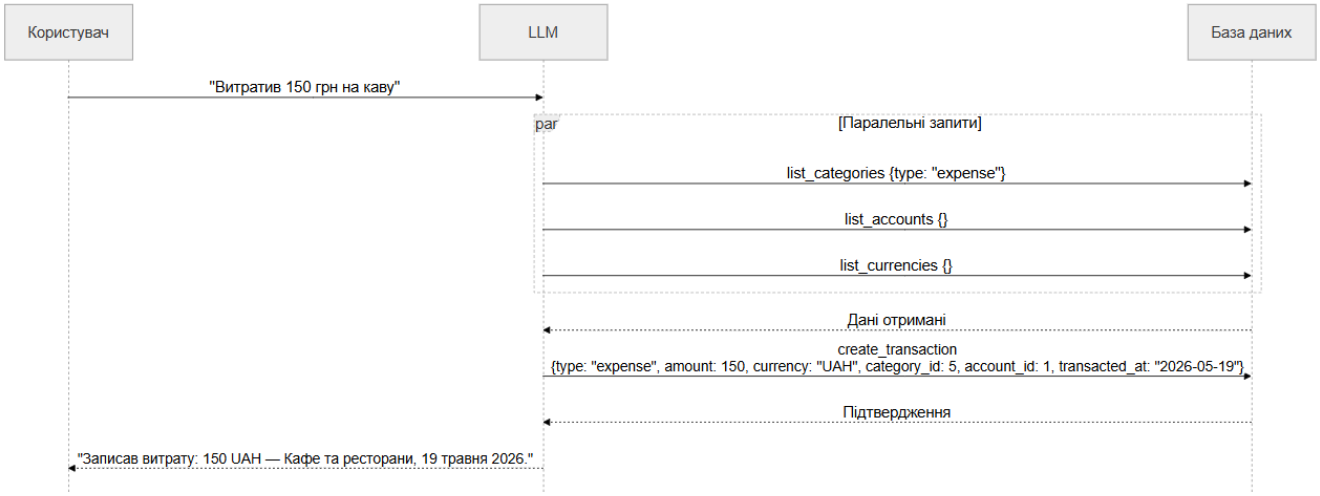

Рисунок 3.4 – Sequence-діаграма типового MCP-запиту

Паралельний виклик трьох look-up інструментів перед основним reduce час відповіді порівняно з послідовними викликами.

3.4 Реалізація модуля звітності та аналізу бюджетних потоків

Для забезпечення високої швидкодії під час формування аналітичних зрізів було прийнято рішення відмовитися від ORM (DQL) на користь нативних SQL-запитів через компонент Doctrine DBAL. Це дозволило використовувати умовні агрегації, підзапити та специфічні математичні функції СКБД, суттєво знижуючи навантаження на оперативну пам'ять сервера.

Найбільш складним алгоритмом модуля є комбінований аналіз план-факту (`getPlanVsActual`), який підтримує точний та діапазонний режими функціонування. Для коректної обробки ковзних (rolling) планів на початку або в середині періоду, у коді реалізовано алгоритм рівномірного масштабування лімітів пропорційно прожитим дням.

Програмна логіка виконання математичного алгоритму масштабування деталізована в таблиці 3.7. В той час, як специфікація аналітичних методів

`ReportService` [30] та структура формування результуючих відповідей для побудови графіків представлені в таблиці 3.8.

Таблиця 3.7 – Покроковий алгоритм розрахунку масштабованого ліміту ковзного плану

Етап	*Розрахункова операція / Логічний крок*	*Математичний вираз / Параметр*	*Опис інженерної поведінки*
1	Визначення еталонної тривалості	T_{total} = `Monthly (30/31 день) | Yearly (365/366)`	Розрахунок повної кількості днів у поточному календарному періоді.
2	Фіксація прожитого часу	T_{past} = `SystemTime - PeriodStart`	Кількість днів, що реально минули від початку дії ліміту до сьогодні.
3	Обчислення коефіцієнта	$K_{scale} = T_{past}/T_{total}$	Пропорційна частка часового відрізка.
4	Масштабування суми ліміту	$P_{scaled} = P_{total} * K_{scale}$	Отримання ефективного ліміту для поточного дня.
5	Агрегація фактичних витрат	$Fact$ = `SUM(account_amount)`	Вибірка фактичних транзакцій з БД за період T_{past}.
6	Розрахунок відсотка	$Percentage = (T_{total}/T_{total}) * 100\%$	Визначення точного стану виконання плану (`OnTrack / Underused`).

Таблиця 3.8 – Специфікація аналітичних методів модуля звітності

Метод API	*Параметри фільтрації (ReportFilterRequest)*	*Механізм SQL-агрегації (Doctrine DBAL)*	*Сфера застосування на інтерфейсі*
`getSummary`	`from, to, scope, account_id`	`SUM(CASE WHEN type = 'income' THEN ...)`	Загальний баланс, віджети доходів та витрат за період.
`getCategoryBreakdown`	`from, to, type, group_id`	`JOIN categories GROUP BY c.id ORDER BY total DESC`	Кругова діаграма (Pie Chart) розподілу витрат за категоріями.
`getTimeSeries`	`from, to, granularity`	`GROUP BY DATE(t.transacted_at) | YEARWEEK()`	Лінійний графік (Line Chart) динаміки фінансових потоків.
`getPlanVsActual`	`from, to, period_type`	Масштабування + підзапити підкатегорій (`parent_id`)	Індикатори виконання лімітів (Progress Bars).
`getAccountSummary`	`account_id, from, to`	`SUM(CASE WHEN transacted_at < :from THEN ...)`	Розрахунок початкового та кінцевого балансу рахунку, обіг у розрізі оригінальних валют.

Важливою інженерною особливістю алгоритму є також підтримка підкатегорій: якщо у плані бюджету встановлено прапорець `include_subcategories = true`, рекурсивна SQL-умова автоматично включає у фактичні витрати транзакції не лише за цільовою категорією, а й за всіма її дочірніми елементами (`parent_id = p.category_id`).

3.5 Реалізація механізму мультивалютності та управління курсами обміну

Відповідно до реляційної ER-моделі бази даних (рис. 2.17), система підтримує повну функціональну ізоляцію валюти операції від базової валюти рахунку. Програмний комплекс `ExchangeRateService` забезпечує виконання критично важливих бізнес-правил, алгоритм пошуку яких (`getRate`) представлено на рисунку 3.5.

Кожна транзакція фіксує фінансові показники у двох вимірах. Це гарантує консистентність історичних звітів незалежно від подальших коливань ринку [24]. Структура збереження мультивалютних полів у сутності `Transaction` наведена в таблиці 3.9.

Таблиця 3.9 – Структура збереження мультивалютних полів транзакції

Назва поля сутності	*Тип даних у PHP / СУБД*	*Функціональне призначення*	*Інженерна поведінка ТХТ*
`amount`	`float / DECIMAL(15,2)`	Фіксація суми в оригінальній валюті здійснення операції.	Вноситься користувачем або розпізнається ШІ [28].
`currency`	`string / CHAR(3)`	Код оригінальної валюти (ISO 4217).	Нормалізується до верхнього регістру (`USD, UAH`).
`accountAmount`	`float / DECIMAL(15,2)`	Сума, конвертована у базову валюту рахунку списання.	Використовується СКБД для розрахунку балансу рахунку [28].
`exchangeRate`	`float / DECIMAL(18,6)`	Фіксований історичний курс обміну на момент транзакції.	Зберігається каскадно для ізоляції від майбутніх змін курсів.

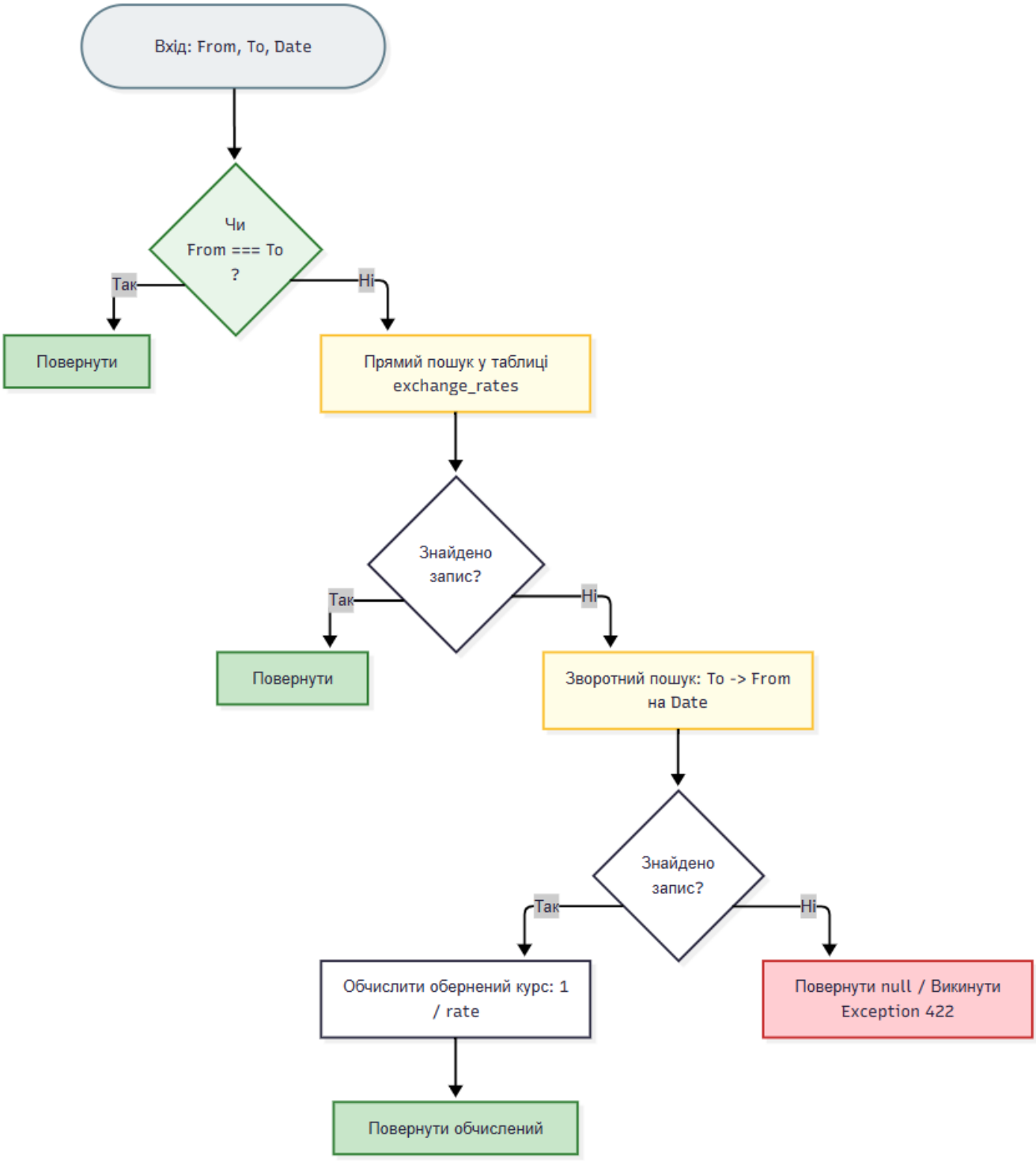

Рисунок 3.5 – Схема алгоритму розрізнення та пошуку курсів обміну (`getRate`)

Незважаючи на гнучкість мультивалютного ядра, на рівні інфраструктури інтеграції з ШІ діє суворе архітектурне обмеження: MCP-інструмент `create_transaction` навмисно позбавлений функції автоматичної крос-валютної конвертації. Якщо валюта транзакції, яку розпізнала модель, не збігається з

базовою валютою вибраного рахунку, сервер припиняє виконання та повертає виняткове повідомлення про помилку виконання інструменту.

Це обмеження прийнято свідомо, оскільки введення курсів через природно-мовний інтерфейс має високі ризики неоднозначного тлумачення і може призвести до спотворення фінансової звітності. Крос-валютні операції користувачу рекомендується виконувати виключно через веб-інтерфейс Angular SPA.

3.6 Конфігурування середовища та робота з чутливими даними

Для забезпечення швидкого розгортання та повної ізоляції середовищ розробки (`dev`) та тестування (`test`), серверна архітектура повністю контейнеризована за допомогою Docker Compose [31]. Інфраструктура складається з чотирьох сервісів, підключених до ізольованої внутрішньої bridge-мережі `budget_net`. Конфігураційні параметри та порти сервісів наведено в таблиці 3.10.

Таблиця 3.10 – Специфікація контейнерів інфраструктури Docker Compose

Назва сервісу	*Базовий образ (Оточення)*	*Порти (Зовнішній:Внутрішній)*	*Функціональні обов'язки та логіка перевірки стану ТХТ*
`php`	`php:8.2-fpm-alpine`	Ізольовано (FPM-сокет)	Виконання PHP-коду, компіляція DI, виконання міграцій Doctrine.
`nginx`	`nginx:1.25-alpine`	`8080:80`	Проксі-сервер REST API запитів, роздача статики, ізоляція індексу.
`mysql`	`mysql:8.0`	`13306:3306`	Транзакційне сховище (InnoDB). Має `healthcheck` (`mysqladmin ping`).
`mysql_test`	`mysql:8.0`	`13307:3306`	Ізольована база даних для виконання автоматизованих тестів.

Безпека чутливих даних реалізована через ієрархічну систему файлів середовища Symfony. Відповідно до інженерних стандартів, файли, які містять реальні секрети, ключі сторонніх провайдерів ШІ (Anthropic/OpenAI) [7] та секрети JWT, додані до виключень системи контролю версій (`.gitignore`). Специфікація розподілу та захисту конфігураційних файлів наведена в таблиці 3.11.

Таблиця 3.11 – Матриця розподілу конфігураційних секретів середовища

Назва файлу	*Сфера застосування (Оточення)*	*Статус у Git репозиторії*	*Перелік ключових змінних (Секретів)*
`.env`	Локальне (Шаблонні значення)	Комітується	`APP_ENV=dev`, `DATABASE_URL` (Шаблон)
`.env.local`	Локальне (Override розробника)	Додано в .gitignore	Реальні паролі до локальної MySQL бази даних.
`.env.prod`	Серверне (Production дефолти)	Комітується	Конфігурація лімітів `JWT_TTL`, `CORS_ALLOW_ORIGIN`.
`.env.prod.local`	Серверне (Production секрети)	Додано в .gitignore	Ключі провайдерів ШІ (`ANTHROPIC_API_KEY`), `JWT_PASSPHRASE`.

Асиметричне шифрування JWT-токенів за алгоритмом RS256 [27, 32] використовує пару криптографічних ключів (`private.pem / public.pem`), взаємодія з якими налаштована через пакет `lexik/jwt-authentication-bundle`. Для забезпечення сумісності з різними типами клієнтських платформ програмно реалізовано два незалежні методи вилучення токена при кожному HTTP-запиті:

– заголовок `Authorization: Bearer <token>` – основний високозахищений метод для взаємодії з Angular SPA застосунком;

– cookie-файл з іменем `BEARER` – резервний варіант (fallback), інтегрований для клієнтських середовищ або специфічних обмежених хостингів, які за замовчуванням блокують або затирають нестандартні HTTP-заголовки авторизації.

Для запобігання атакам типу SQL-ін'єкцій та забезпечення цілісності бізнес-моделей, на межі контролерів впроваджено сувору валідацію вхідних DTO-пакетів за допомогою компонента `Symfony Validator`. Кожен об'єкт запиту проходить автоматизовану перевірку обмежений (Constraints) перед передачею до сервісного шару бізнес-логіки.

Висновки до розділу 3

У третьому розділі успішно реалізовано програмний комплекс серверної частини та інтелектуальних модулів системи управління фінансами на базі фреймворку Symfony 7.4 та мови програмування PHP 8.2+.

Розроблено захищене ядро системи з використанням асиметричної JWT-автентифікації (RS256) та трирівневої рольової моделі, яка повністю покриває права доступу спроєктованих у Розділі 2 акторів. Реалізовано транзакційну логіку фінансового обліку та колективних бюджетів, включаючи каскадний алгоритм резолюції мультивалютних сум та автоматичний контроль життєвого циклу токенізованих запрошень.

Програмно впроваджено інфраструктурний сервер протоколу MCP, що забезпечує асинхронну обробку JSON-RPC повідомлень через stdio-потоки для інтеграції системи з великими мовними моделями за допомогою патерну Стратегія. Оптимізовано модуль звітності шляхом переведення аналітичних запитів на нативний SQL (Doctrine DBAL), а також впроваджено математичний алгоритм рівномірного масштабування лімітів для ковзних планів бюджету пропорційно прожитим дням.

Спроєктовано стійку до ринкових коливань систему мультивалютності із каскадним автозбереженням та пошуком обернених курсів обміну без надлишкового дублювання записів у СКБД. Середовище розгортання повністю контейнеризовано за допомогою чотирьох ізольованих сервісів Docker Compose, а управління секретами та чутливими токенами ШІ-провайдерів відокремлено від коду згідно з вимогами безпеки.

4 ПРОГРАМНА РЕАЛІЗАЦІЯ КЛІЄНТСЬКОЇ ЧАСТИНИ, ІНТЕГРАЦІЇ З ШІ ТА РОЗГОРТАННЯ СИСТЕМИ

4.1 Побудова архітектури Angular-застосунку: сервіси, компоненти, інтерцептори

Клієнтська частина системи розроблена на базі платформи Angular 17 [11] з використанням концепції *standalone-компонентів*. Цей підхід дозволяє повністю відмовитися від громіздких модулів (`NgModule`), завдяки чому кожен компонент самостійно декларує власні залежності. Точкою входу є файл `app.config.ts`, де зосереджено глобальні провайдери. Така конфігурація дозволяє оптимізувати процес збірки через механізм *tree-shaking*, суттєво зменшуючи підсумковий JavaScript-компілят застосунку.

Файлова структура клієнтського коду організована за функціональними рівнями відповідно до вимог офіційного Angular Style Guide. Компонентну схему та взаємозв'язки рівнів представлено на рисунку 4.1.

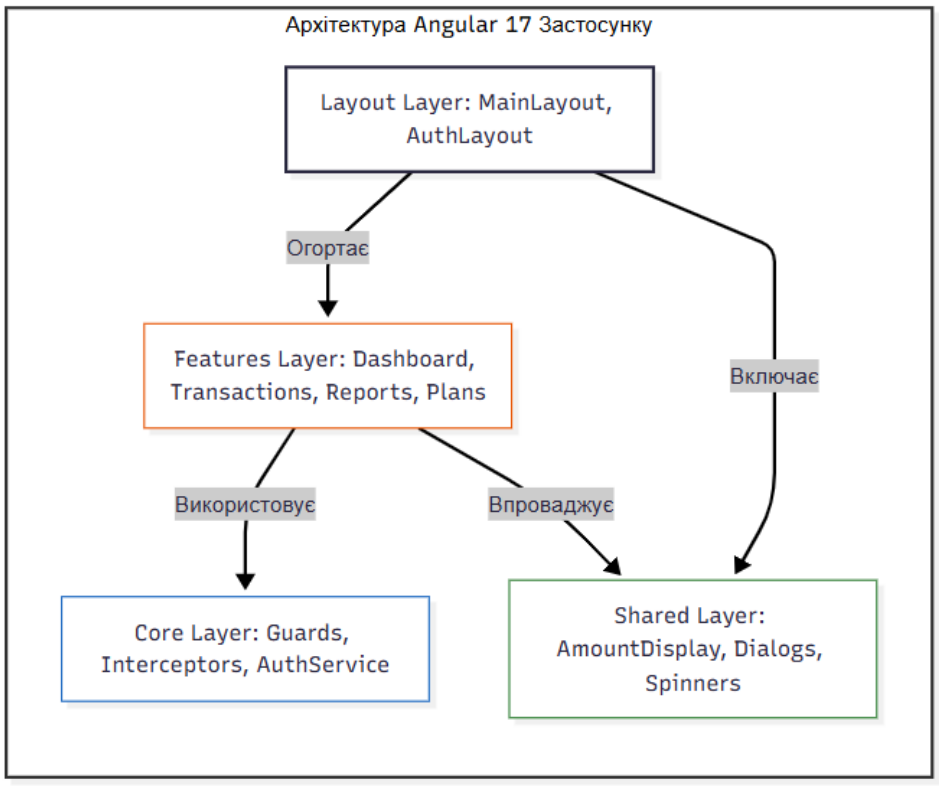

Рисунок 4.1 – Схема взаємодії рівнів клієнтської архітектури

Для оптимізації швидкості відклику інтерфейсу (рис. 4.2) маршрутизація реалізована за технологією ледачого завантаження (Lazy Loading). Feature-модулі компилюються в окремі чанки й підвантажуються браузером виключно за умови першого звернення користувача до відповідного URL (див. додаток Б).

Контроль безпеки та розмежування прав доступу на рівні клієнта здійснюють три функціональні навігаційні захисники (Guards). Специфікацію маршрутів та логіку роботи захисників зведено в таблиці 4.1.

Таблиця 4.1 – Специфікація клієнтських маршрутів та механізмів їх захисту

Категорія маршрутів	*Базовий компонент оболонки (Layout)*	*Цільові URL-адреси сторінок*	*Застосований Guard та його бізнес-логіка*
Аутентифікація	`AuthLayoutComponent` (без навігаційних панелей)	`/auth/login`, `/auth/register`, `/auth/pending`	`guestGuard`: блокує доступ авторизованих користувачів до сторінок входу.
Функціональна зона	`MainLayoutComponent` (із бічним меню `MatSidebar`)	`/dashboard`, `/transactions`, `/reports`, `/plans`	`authGuard`: перевіряє сесію та утримує акторів `candidate` на сторінці очікування.
Адміністрування	`MainLayoutComponent`	`/admin/users`	`adminGuard`: обмежує доступ до модуля управління користувачами лише для `ROLE_ADMIN`.

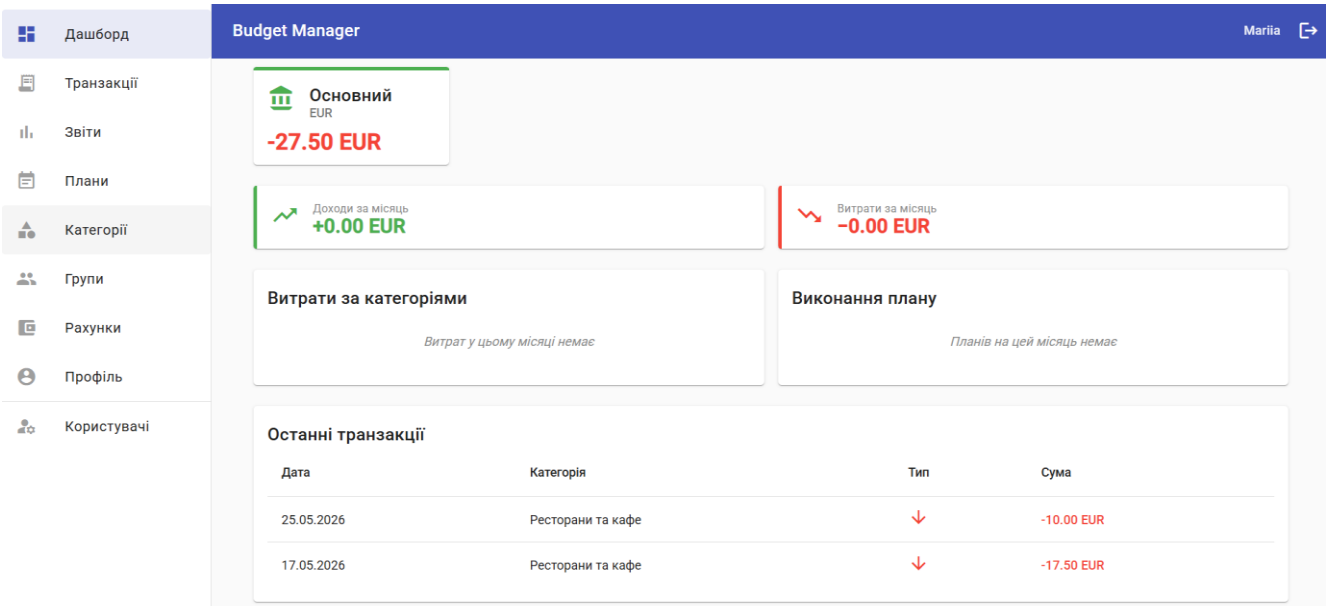

Рисунок 4.2 – Загальний вигляд інтерфейсу системи: бічна навігаційна панель, заголовок та область контенту

На рисунку 4.2 продемонстровано базову компоновку інтерфейсу, де ліва панель `MatSidenav` [33] забезпечує швидкий доступ до розділів, а верхній заголовок відображає поточний стан профілю користувача. Така структура дозволяє актору зберігати контекст навігації під час роботи з різними фінансовими модулями.

Низькорівнева асинхронна робота з сервером автоматизована за допомогою механізму інтерцепторів повідомлень (Interceptor Chain), що дозволяє винести наскрізний функціонал із бізнес-сервісів. У системі реалізовано два класи перехоплювачів. Патерн безшумного оновлення сесії, кешування довідників та уніфікацію поведінки спільних UI-компонентів деталізовано в таблиці 4.2.

Таблиця 4.2 – Інженерна специфікація клієнтських сервісів, інтерцепторів та UI-компонентів

Назва компонента коду	*Архітектурний шар*	*Практичний механізм реалізації*	*Функціональне призначення*
`JWT-Interceptor`	`Core/Network`	Angular HTTP-перехоплювач	Автоматично додає токен до заголовків та прозоро оновлює прострочену сесію через 401 помилку (див. додаток Б) [21]
`Error-Interceptor`	`Core/Network`	Глобальний обробник помилок HTTP	Централізовано транслює помилки API у вигляді спливаючих сповіщень `MatSnackBar`
`AuthService`	`Core/State`	Реактивний потік `BehaviorSubject` [34]	Єдине сховище стану авторизації, що сповіщає UI-вузли про зміну активного користувача (див. додаток Б)
`Category / Currency Services`	`Core/Data`	RxJS оператор `shareReplay(1)`	Реалізує сесійне кешування статичних довідників, запобігаючи надлишковим мережевим запитам
`AmountDisplayComponent`	`Shared/UI`	Умовне форматування + колірна індикація	Рендеринг транзакцій із відповідним знаком (+/-), кількістю знаків та кольором (зелений/червоний)
`ConfirmDialog / EmptyState`	`Shared/UI`	Модальні вікна `MatDialog`, заглушки списків	Стандартизація інтерфейсу при підтвердженні таких дій, як видалення (рис.4.3) та відображенні порожніх станів таблиць

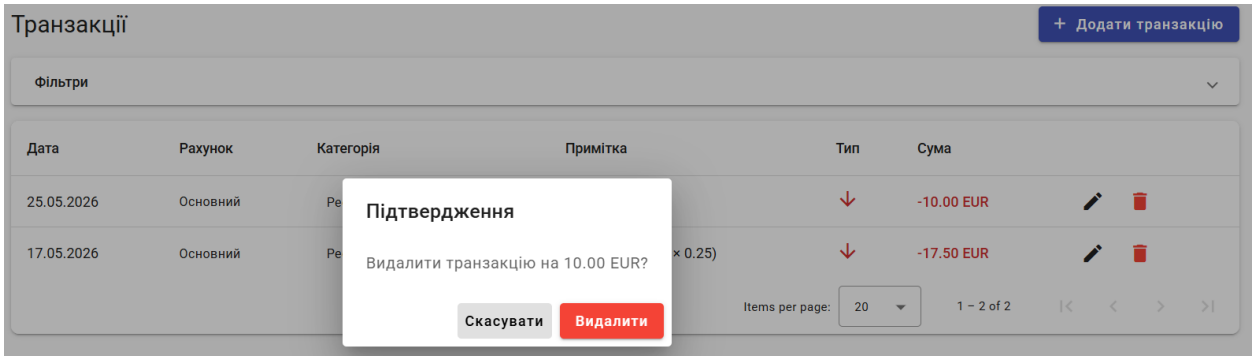

Рисунок 4.3 –Реалізація спільних компонентів: діалог підтвердження видалення

Візуалізація компонентів шару `Shared` (рис. 4.3) демонструє уніфікований підхід до обробки очікування даних, відсутності записів у таблицях та модальних вікон підтвердження. Це забезпечує візуальну консистентність застосунку та мінімізує дублювання коду в feature-модулях.

4.2 Реалізація інтерактивних дашбордів та модулів візуалізації даних

Дашборд є стартовим аналітичним центром користувача, що відображає стан фінансів за поточний місяць (рис. 4.4 та 4.5). Для прискорення завантаження даних застосовано патерн паралельного виконання HTTP-запитів (див. додаток Б). Сторінка містить підсумкові картки доходів/витрат, блок рахунків, кругову діаграму витрат та індикатори виконання бюджетних лімітів (рис. 4.4).

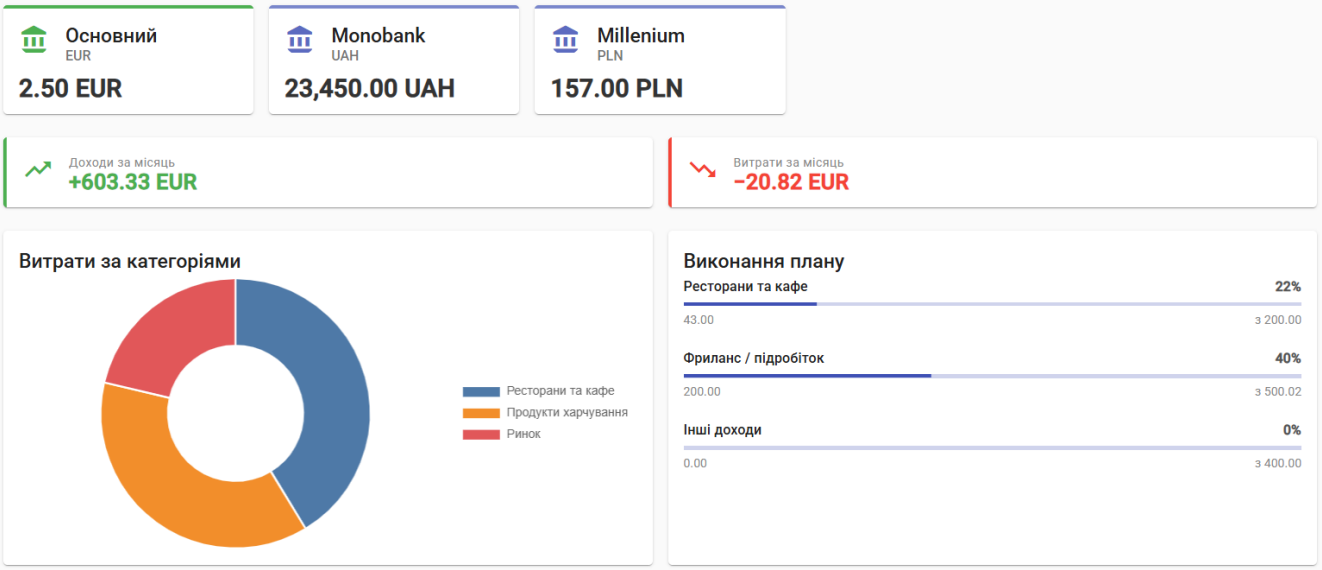

Рисунок 4.4 – Головна сторінка інтерфейсу користувача (дашборд)

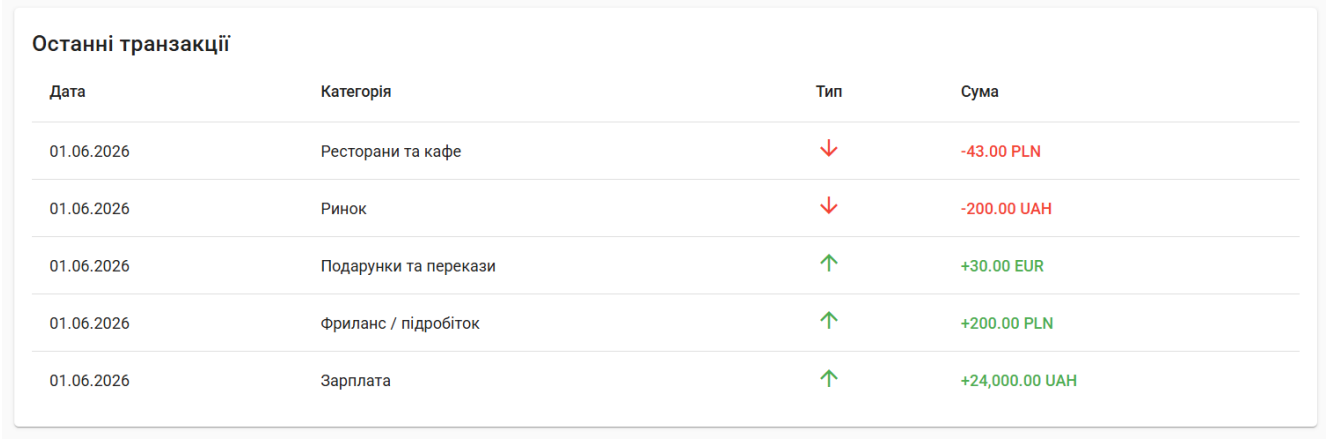

Рисунок 4.5 – Нижня частина дашборду зі списком всіх останніх транзакцій

На рис. 4.5 відображено таблицю останніх транзакцій, куди кожна витрата або дохід підтягуються атвоматично після їх безпосереднього дадання як нових транзакцій.

Сторінка транзакцій реалізує табличний інтерфейс (рис. 4.6) із пагінатором (по 20 записів на сторінку) та розгалужену панель фільтрації за шістьма параметрами. Програмну логіку поведінки модальної форми введення нової фінансової операції, яка забезпечує реактивний зв'язок полів, описано в таблиці 4.3.

Таблиця 4.3 – Логічна матриця станів та поведінки полів форми введення транзакцій

Керуючий елемент форми	*Поточний стан / Дія користувача*	*Реактивна поведінка залежних полів форми*	*Інженерний механізм обробки*
Перемикач типу операції	Вибір значення «Витрата» або «Дохід»	Селектор категорії `MatSelect` миттєво змінює масив даних	Динамічна фільтрація довідника категорій відповідно до обраного типу транзакції
Селектор фінансового рахунку	Обрано рахунок із валютою, що відрізняється від валюти операції	Форма розгортає секцію крос-валютної конвертації	Тригер видимості блоку та ініціація асинхронного запиту історичного курсу в базу даних
Поля мультивалютного блоку	Ручна зміна суми транзакції, курсу або суми рахунку	Перерахунок суміжних полів без участі користувача в режимі реального часу	Реактивне зв'язування через Angular `FormGroup`. Математична калькуляція за формулою: $Сума_{рахунку} = Сума_{транзакції} * Курс$

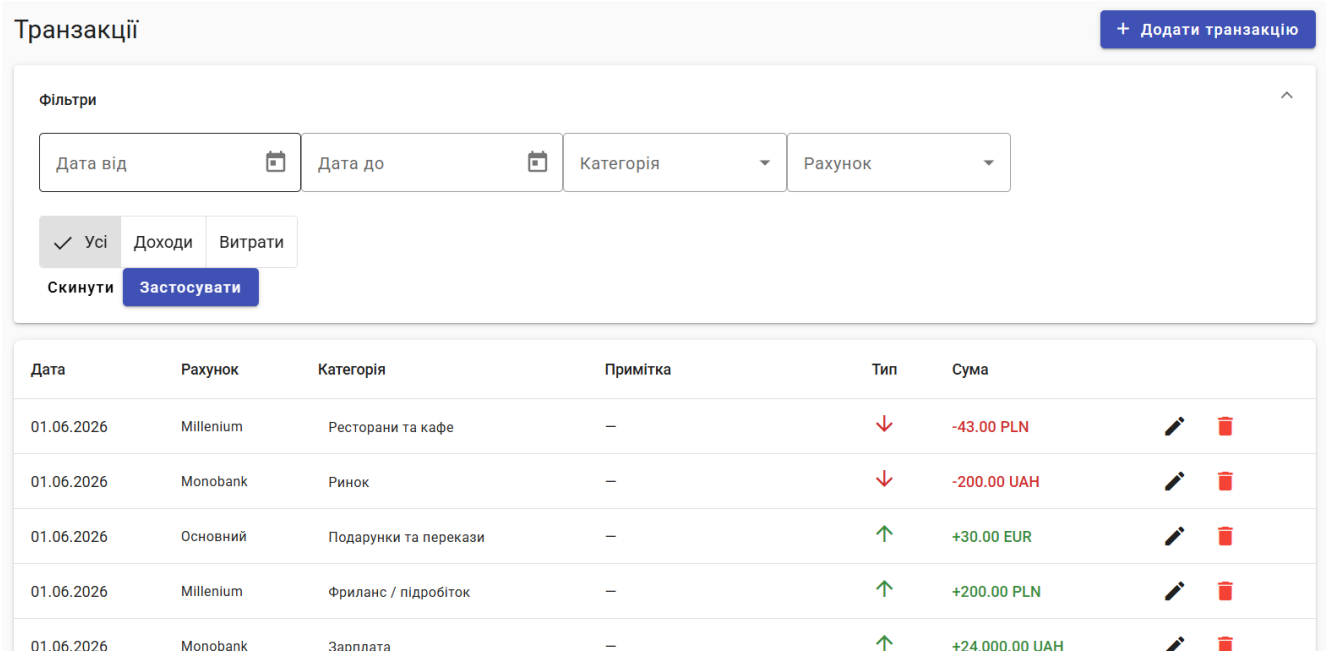

Рисунок 4.6 – Інтерфейс сторінки транзакцій: панель фільтрів та таблиця фінансових записів із пагінатором

Ілюстрація вище відображає основний робочий простір для управління транзакціями, включаючи інструменти пошуку за датою, категорією та рахунками.

Рисунок 4.7 демонструє адаптивність форми введення даних: при виявленні розбіжностей у валютах система автоматично активує поля для роботи з курсом.

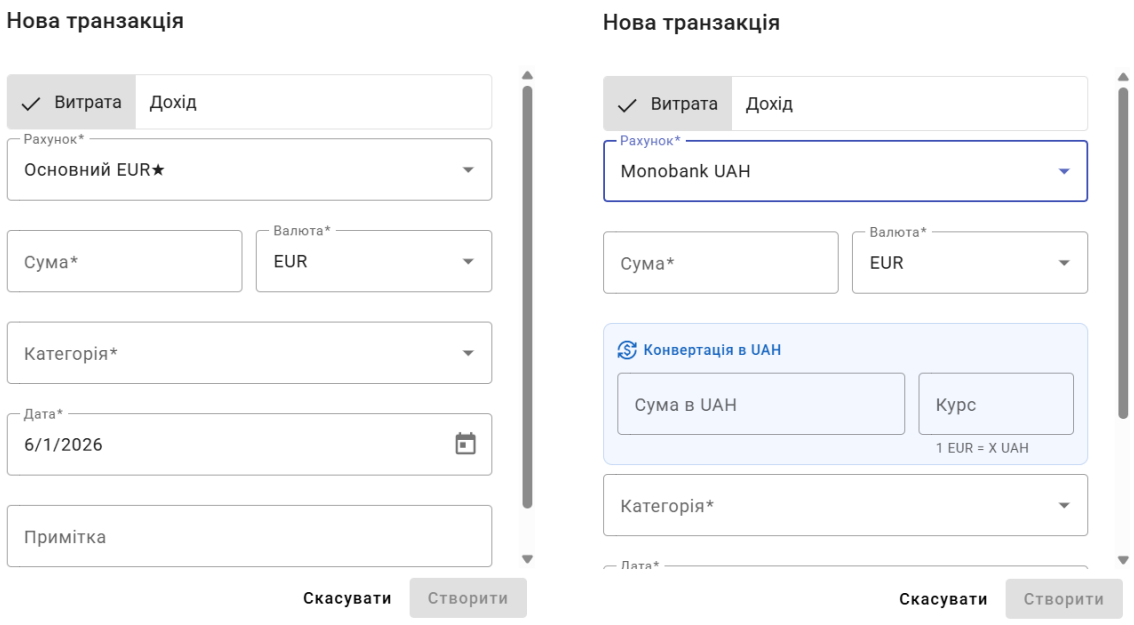

Рисунок 4.7 – Модальне вікно введення транзакції: а) стандартний режим; б) режим крос-валютної конвертації

Аналітичний модуль сторінки звітів є найбільш калькуляційно навантаженим компонентом інтерфейсу. Він оперує сімома фільтрами та агрегує дані у трьох графічних вимірах. Функціональну специфікацію інтерфейсних сторінок аналітики, довгострокового планування лімітів та управління колективними бюджетами наведено в таблиці 4.4.

Таблиця 4.4 – Функціональна специфікація аналітичних та групових модулів інтерфейсу

Інтерфейсний модуль	*Елементи візуалізації на UI*	*Керуючі компоненти та часові горизонти*	*Бізнес-логіка та алгоритми відображення*
Модуль аналітики та звітів	Дві кругові діаграми категорій (доходи/витрати); лінійний графік динаміки; картка зведення рахунку.	Швидкі пресети дат («Цей місяць», «Цей рік»); перемикач гранулярності (день/тиждень/місяць).	Відображає початковий/кінцевий баланси рахунку. Лінійний графік будує два незалежні ряди даних на спільній часовій осі з напівпрозорим заповненням.
Модуль планів бюджету	Картковий `Grid-layout` із колірними прогрес-барами стану лімітів.	Чіпси вибору періоду (Тиждень/Місяць/Рік); навігаційні стрілки зміщення часового вікна.	Розраховує та ілюструє ступінь виконання планів за формулою $Fact/Plan$. Для рівномірних ковзних планів застосовує алгоритм масштабування ліміту строго до прожитих днів.
Модуль категорій	Двохвкладкове ієрархічне дерево (`MatTabs`) [33].	Діалогові форми створення підкатегорій з вибором кольорового маркера.	Реалізує візуальне відображення підкатегорій із фіксованим зміщенням (відступом) відносно батьківської лінії.
Колективне бюджетування	Списки учасників груп, форми генерації інвайтів.	Текстове поле введення email; кнопка копіювання токенізованого UUID посилання.	Дозволяє власнику керувати складом групи. Відображає індивідуальні ролі акторів (`owner`, `member`). Прив'язує транзакції до групового scope.

На рисунку 4.8 представлено інструментарій для поглибленого аналізу структури витрат та доходів за допомогою діаграм Chart.js та швидкої фільтрації періодів.

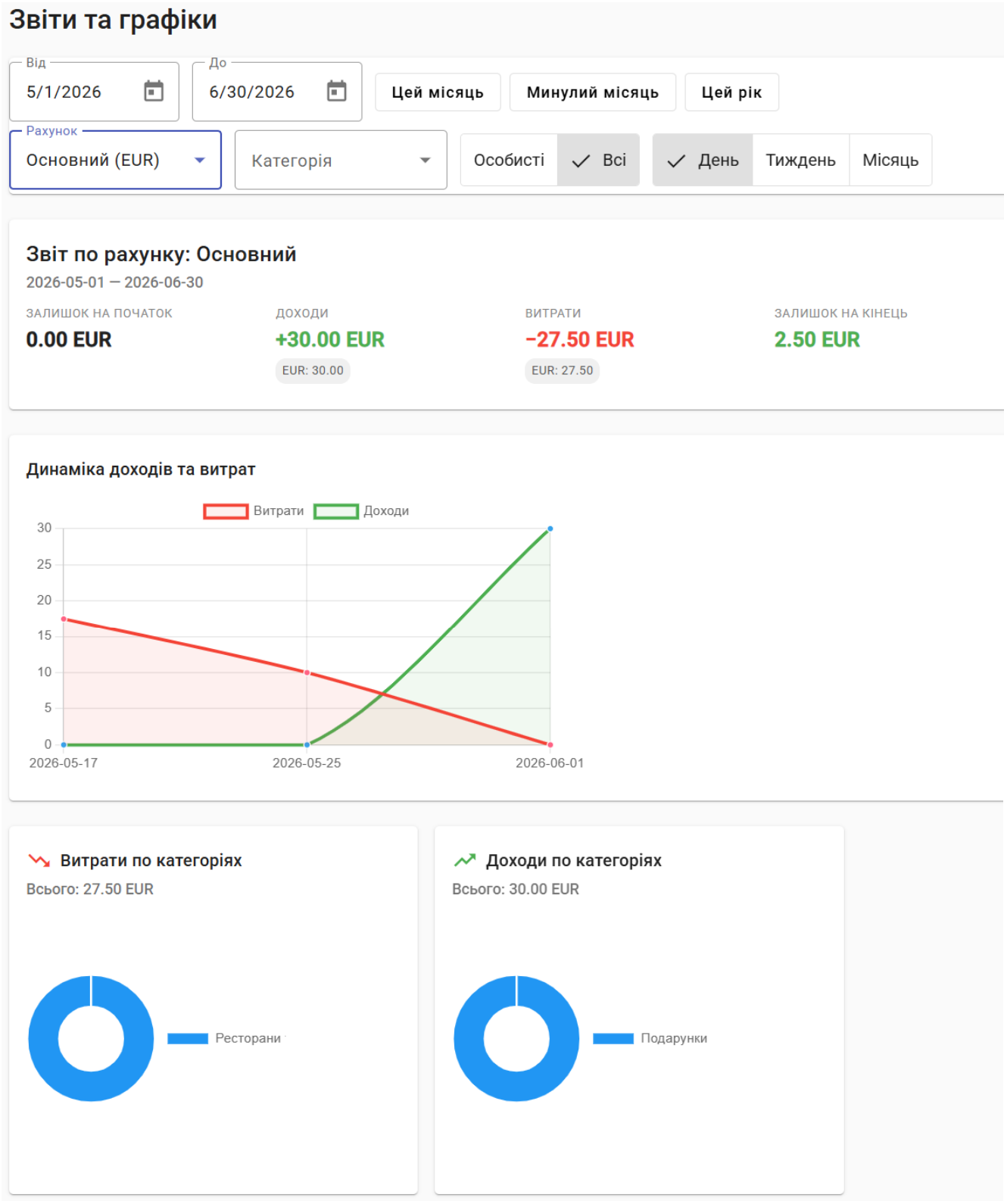

Рисунок 4.8 – Панель фільтрів сторінки звітів, лінійний графік часових рядів динаміки доходів та витрат за обраний період та кругові діаграми розбивки фінансових потоків за категоріями

Графік динаміки (рис. 4.8) візуалізує зміну фінансових потоків у часі, дозволяючи користувачу ідентифікувати пікові навантаження на бюджет за допомогою налаштування гранулярності.

Наступним є огляд сторінки планування бюджету (рис. 4.9).

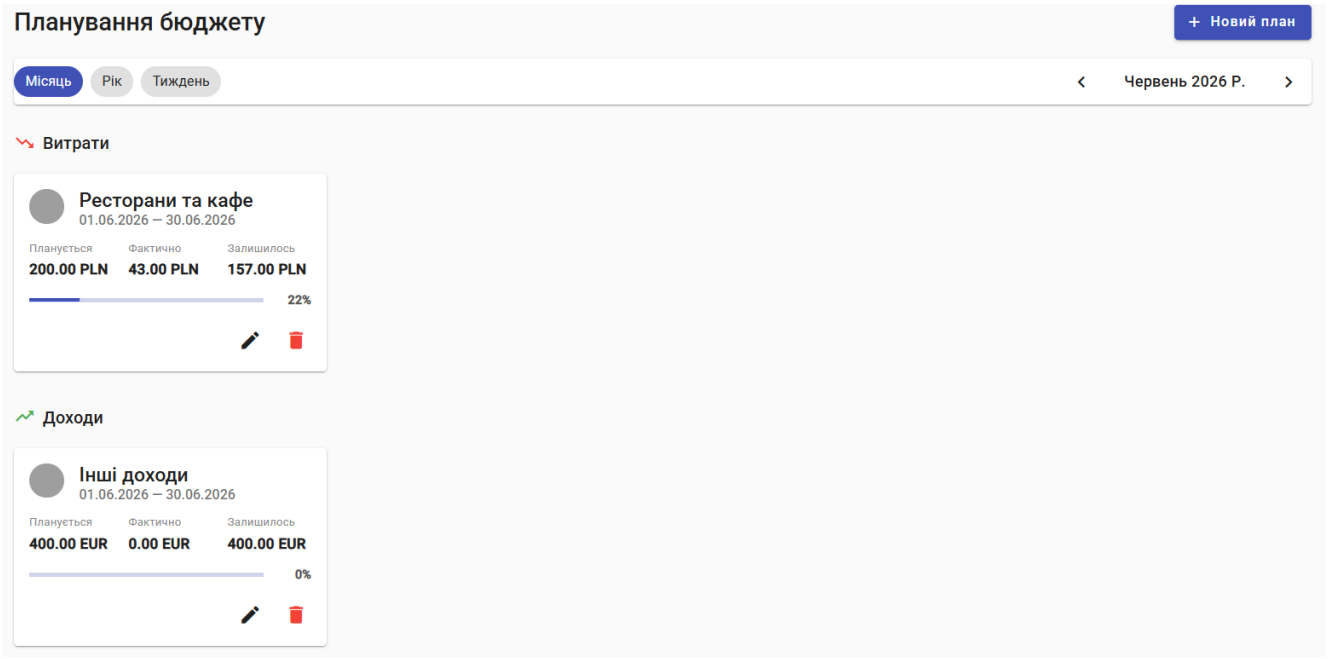

Рисунок 4.9 – Інтерфейс сторінки планів: картки лімітів бюджету з колірною індикацією ступеня виконання

Модуль планування використовує прогрес-бари з колірним кодуванням (синій/жовтий/червоний) для миттєвого інформування користувача про стан освоєння бюджетних лімітів.

4.3 Підключення AI-асистента на базі протоколу MCP та взаємодія з системою через природну мову

Інтелектуальний модуль системи базується на використанні відкритого протоколу MCP [7], що дозволяє відокремити інтерфейс мовної моделі від бізнес-домену, підключивши систему управління бюджетом до зовнішніх LLM-хостів, зокрема, Claude Desktop від Anthropic.

Для успішної побудови каналу інтелектуальної взаємодії мають бути виконані три обов'язкові інфраструктурні умови:

– серверний фоновий процес: розгорнутий та запущений Docker-контейнер `budget_php`, що містить консольну команду `mcp:serve`, яка прослуховує stdio-потоки;

– автентифікаційний токен сесії: наявність діючого короткоживучого криптографічного ключа `access_token` (JWT) [18] поточного користувача;

– конфігурований хост: встановлений локальний клієнт Claude Desktop, що підтримує підключення сторонніх MCP-серверів контексту.

Процес підключення AI-асистента до ядра системи виконується (рис. 4.10 та 4.11) за чітко регламентованим інженерним алгоритмом, етапи якого описано у таблиці 4.5.

Таблиця 4.5 – Інженерна інструкція з підключення системи до Claude Desktop через MCP

Етап підключення	*Область виконання операції*	*Необхідні маніпуляції та інструменти*	*Цільовий результат кроку*
Крок 1. Екстракція JWT токена	Браузер користувача (Angular SPA)	Авторизація в системі → Інструменти розробника (F12) → Сховище Local Storage.	Копіювання активного рядка токена із ключа `access_token`.
Крок 2. Відкриття конфігурації	Інтерфейс Claude Desktop	Меню `Settings` → вкладка `Developer` → активація опції `Edit Config`.	Автоматичне відкриття системного конфігураційного файлу сервера в текстовому редакторі ОС.
Крок 3. Ін'єкція параметрів	Конфігураційний JSON-файл хоста	Додавання назви сервера `"budget"` у блок `"mcpServers"`. Зазначення команди `docker exec` та аргументів консолі Symfony.	Передача згенерованого JWT-токена всередину змінної оточення `BUDGET_USER_TOKEN`.
Крок 4. Ініціалізація та Handshake	Операційна система (Процес)	Повний перезапуск програми Claude Desktop.	Поява іконки інструментів ШІ у полі введення чату, успішне зчитування довідників (`list_*`).

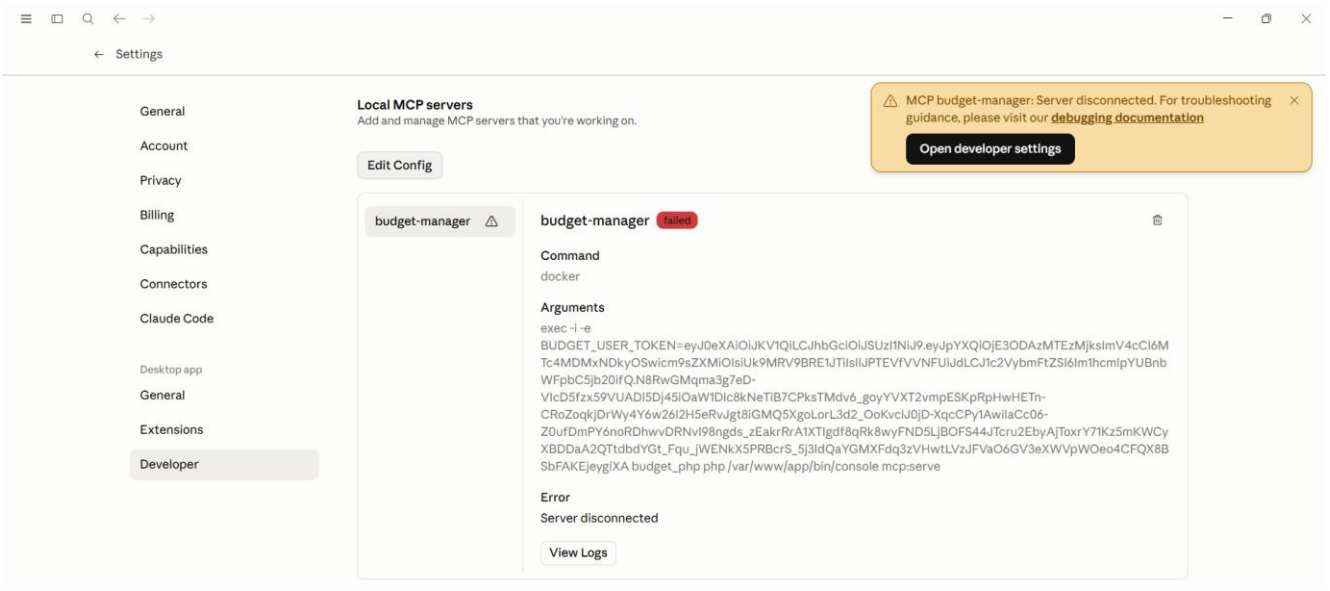

Рисунок 4.10 – Конфігурування інтеграції MCP-сервера у системному файлі налаштувань Claude Desktop

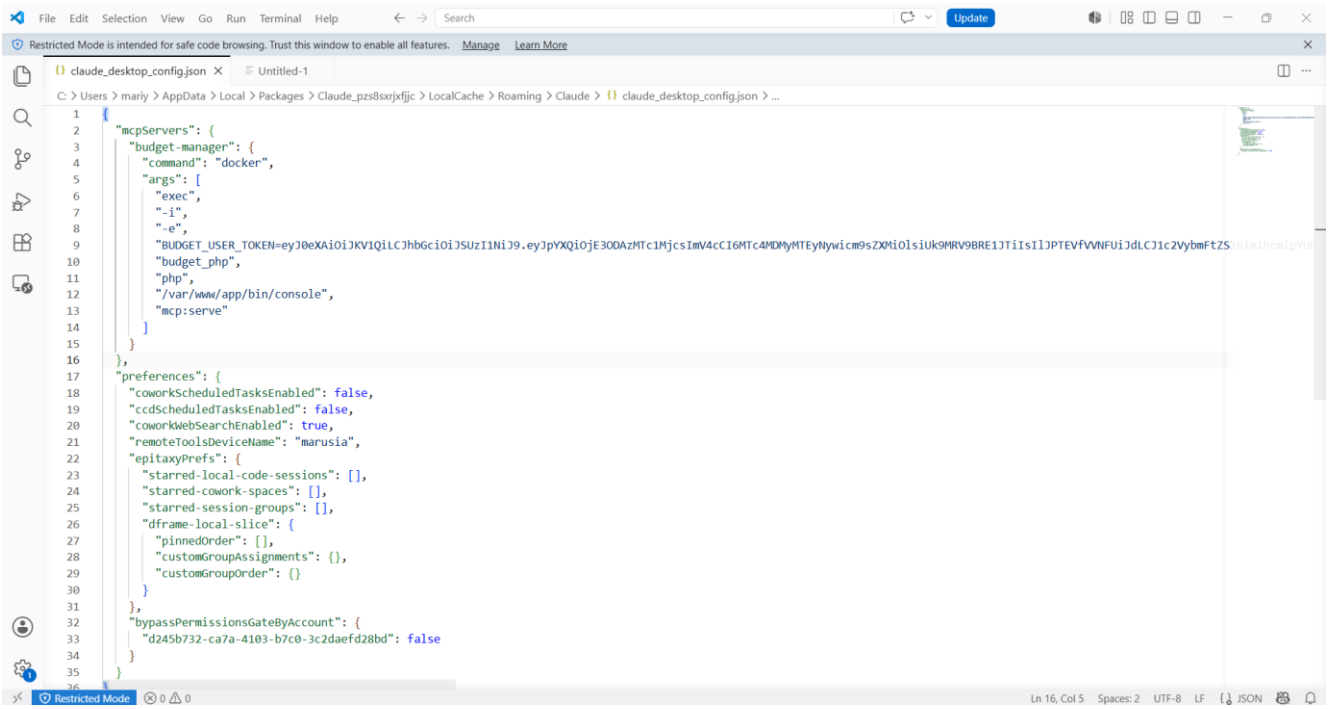

Рисунок 4.11 – Структура JSON-конфігурації

На ілюстрації (рис. 4.11) показано структуру JSON-конфігурації, яка пов'язує зовнішній AI-хост із консольною командою Symfony та передає токен авторизації через змінні оточення. Після коректного підключення інтерфейс Claude Desktop буде виглядати наступним чином (див. рис. 4.12).

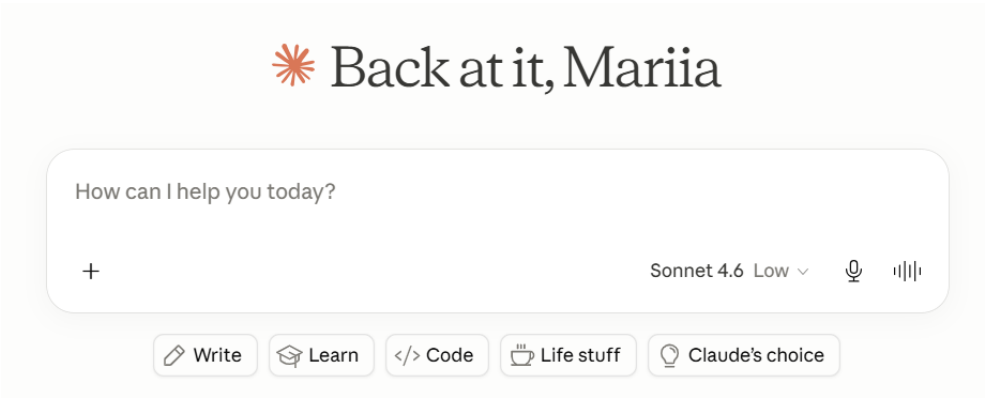

Рисунок 4.12 – Інтерфейс Claude Desktop після успішної ініціалізації з відображенням доступних інструментів API

Скріншот підтверджує успішне «рукостискання» (handshake) між системами, відображаючи список доступних функцій для ШІ, таких як запис транзакцій та перегляд довідників.

Після завершення етапу ініціалізації мовна модель отримує можливість транслювати запити користувача, написані у довільній формі, у суворі структуровані JSON-RPC пакети [19], які розуміє ядро Symfony. Схему обміну повідомленнями при реєстрації фінансової операції через природно-мовний інтерфейс Claude Desktop представлено на рисунку 4.13.

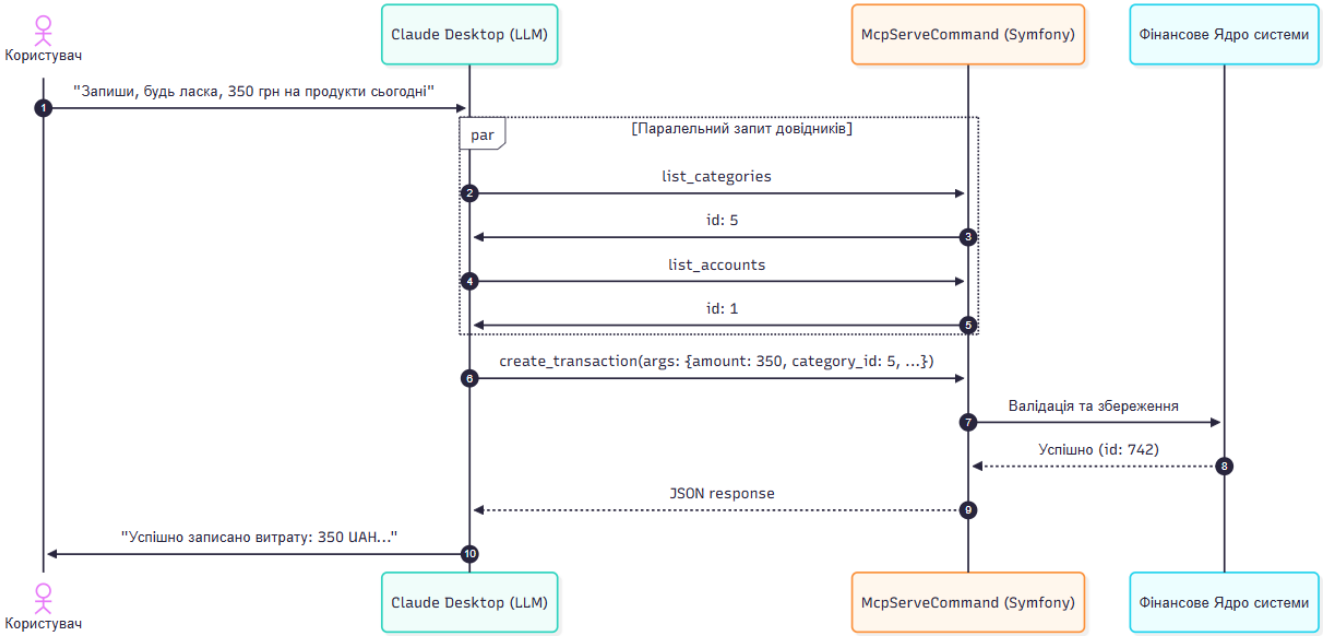

Рисунок 4.13 – Сценарій послідовності викликів обробки ШІ-запиту через MCP

На рисунках 4.14 та 4.15 зафіксовано реальний діалог, де асистент самостійно нормалізує назву категорії та дату, після чого підтверджує успішне внесення запису в базу даних системи.

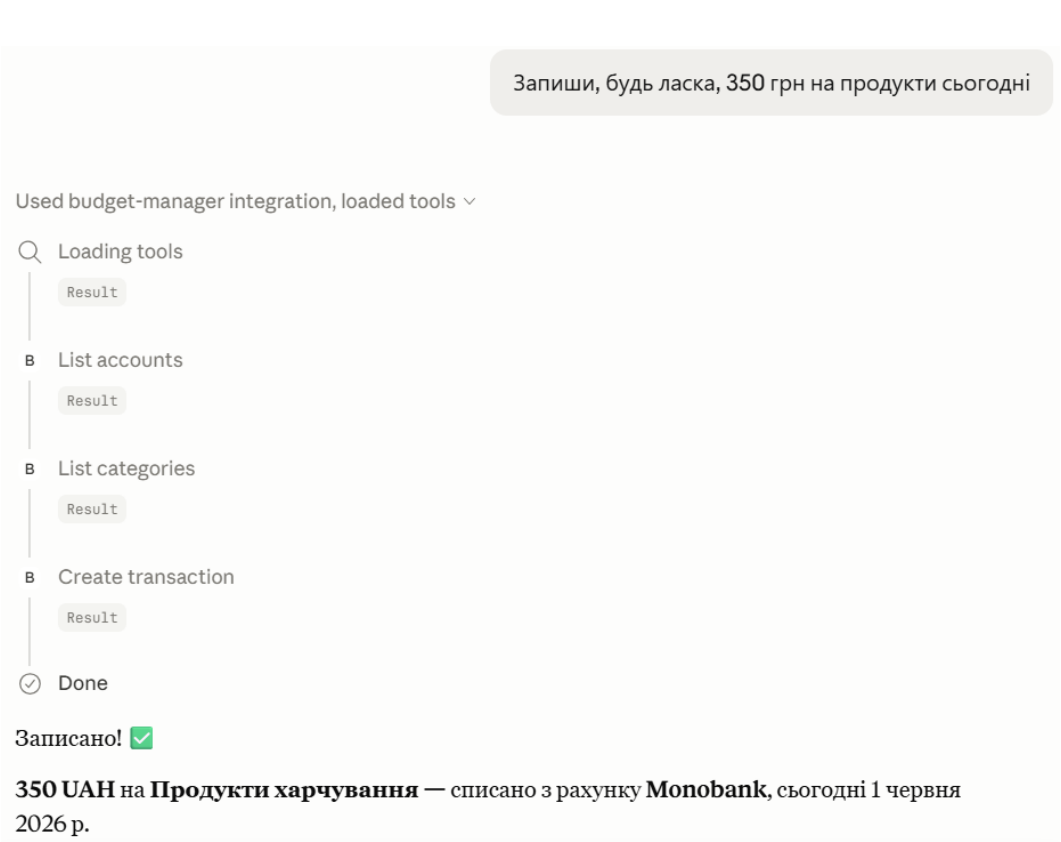

Рисунок 4.14 – Приклад успішного виконання сценарію реєстрації фінансової транзакції за допомогою природно-мовного ШІ-агента

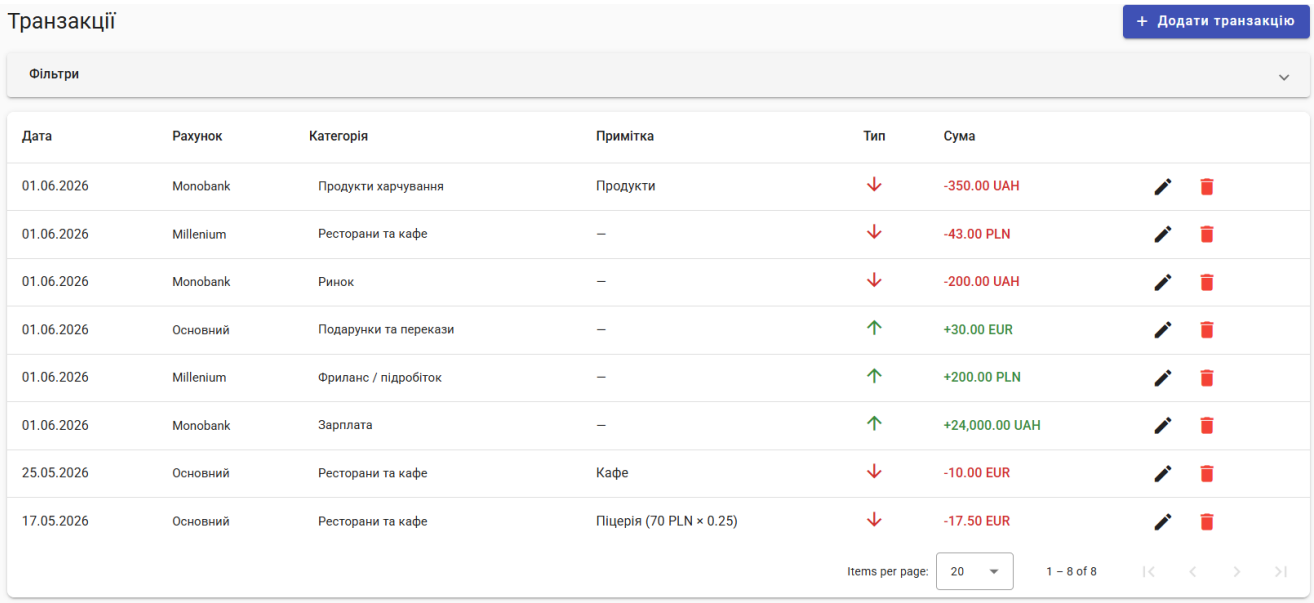

Рисунок 4.15 – Нова транзакція в списку, яка була додана ШІ-агентом

Поточна версія інтелектуального модуля має чітко регламентовані інженерні обмеження:

– ізоляція операцій: MCP-інтерфейс спроєктовано виключно у режимі запису даних (Write-Only). Операції перегляду важкої аналітики, редагування або фізичного видалення транзакцій заблоковані;

– часовий ліміт сесії: оскільки команда `mcp:serve` використовує стандартний токен доступу, через 1 годину (після закінчення TTL) сесія ШІ почне повертати помилку автентифікації;

– мультивалютне блокування: з метою уникнення фінансових аномалій через неоднозначність природної мови, інструмент `create_transaction` автоматично відхиляє транзакції, якщо валюта операції не збігається з валютою рахунку, рекомендуючи користувачу внести операцію через веб-інтерфейс Angular.

4.4 Опис сценаріїв розгортання та перевірка функціональної придатності системи

Архітектура системи розгорнута за допомогою технології контейнеризації Docker. Розробку та перевірку працездатності реалізовано у двох незалежних середовищах: локальному (`development`) та виробничому (`production`).

Локальне середовище розгортається однією командою через Docker Compose й об'єднує чотири контейнери бекенду (PHP-FPM, Nginx, MySQL, MySQL_Test) [31, 35], підключені до ізольованої bridge-мережі `budget_net`. Фронтенд-частина у dev-режимі функціонує в ізольованому Node.js контейнері за допомогою вбудованого сервера Angular (`ng serve`). Для обходу обмежень крос-доменних запитів налаштовано локальний проксі-файл, який перенаправляє шлях `/api` безпосередньо на порт Nginx-бекенда.

Виробниче розгортання фронтенду базується на оптимізованому двоетаповому образі (Multistage Docker Build). Етапи та інженерні особливості цієї технології наведено в таблиці 4.6.

Таблиця 4.6 – Етапи та конфігурація виробничої збірки фронтенду

Етап збірки образу	*Базовий Docker-образ*	*Операції, що виконуються всередині контейнера*	*Результат та оптимізаційні характеристики*
Етап 1. Компіляція	`node:20-alpine`	`npm run build -- --configuration=production`	TypeScript компілюється в мінімізований JavaScript. З використанням tree-shaking
Етап 2. Дистрибуція	`nginx:1.25-alpine`	Копіювання статичного bundle та налаштування SPA-fallback у конфігурації Nginx	Фінальний образ має мінімальну вагу; налаштовано довготривале кешування статики та перенаправлення всіх 404 помилок роутингу на index.html

Для забезпечення високої швидкості завантаження виробничого інтерфейсу на рівні веб-сервера Nginx налаштовано прапорець довготривалого кешування `Cache-Control: public, immutable` терміном на 1 рік для всіх статичних ресурсів. Інвалідація застарілого кешу відбувається автоматично, оскільки Angular при кожній production-збірці додає унікальний криптографічний хеш змісту безпосередньо в імена згенерованих файлів.

Виробниче API серверної частини розгорнуто на віддаленому shared-хостингу InfinityFree за адресою `https://dvm.infinityfree.me`. З огляду на специфіку хостингу, який за замовчуванням затирає HTTP-заголовки авторизації, у коді успішно активовано резервний механізм екстракції сесії через Cookies-файл `BEARER`, що забезпечує стабільність з'єднання у будь-яких мережевих умовах.

Перевірка функціональної придатності розробленої інтелектуальної системи виконувалася методом наскрізного мануального та інтеграційного тестування ключових бізнес-модулів відповідно до зафіксованих у Розділі 2 Use Case сценаріїв. Зведені результати тестування та відповідність очікуваній інженерній поведінці наведено у таблиці 4.7.

Таблиця 4.7 – Протокол перевірки функціональної придатності модулів системи

Назва модуля, що тестувався	*Тестовий сценарій взаємодії*	*Фактична поведінка системи під час перевірки*	*Статус придатності*
Автентифікація	Спроба входу, закінчення сесії	Безшумне виконання `Token Refresh` після закінчення терміну сесії.	Успішно пройдено
Транзакції та валюти	Введення крос-валютної операції	Автоматичне заповнення полів конвертованої суми за історичним курсом	Успішно пройдено
Звітність та аналітика	Вибір діапазону дат, зміна вигляду лінійного графіка	Doctrine DBAL миттєво сформував зведення. Графіки Chart.js коректно перегрупували осі часу за періодами	Успішно пройдено
Бюджетні плани	Перегляд ковзного плану лімітів у середині місяця	Система застосувала впроваджений алгоритм рівномірного масштабування ліміту строго до прожитих днів	Успішно пройдено
Колективний бюджет	Надсилання інвайту, спроба активації користувачем з іншим email	Система згенерувала унікальний токен запрошення. При спробі несанкційного доступу видано помилку верифікації адреси	Успішно пройдено
AI Інтеграція	Природно-мовний запит у Claude	Успішна нормалізація параметрів та збереження транзакції в асинхронному каналі	Успішно пройдено

За результатами комплексного тестування було сформовано перелік інженерних обмежень поточної версії системи, які зафіксовано як вектор подальшої модернізації програмного продукту:

– прогнозування фінансових потоків: ендпоінт інтелектуального прогнозування наразі функціонує в режимі статичної заглушки й потребує впровадження окремої математичної регресійної моделі або інтеграції спеціалізованого AI-промпту аналізу історії транзакцій;

– автоматизація оновлення курсів: поточна архітектура спирається виключно на ручне введення або каскадне автозбереження курсів із транзакцій користувача. Необхідно реалізувати фонові завдання (Cron-скрипти) для щоденної інтеграції з відкритими API Національного банку України чи Fixer-сервісами;

– веб-інтерфейс AI-чату: логічна структура бази даних (`AiChatSession` та `AiChatMessage`) повністю реалізована на бекенді, проте Angular-компонент чату для клієнтської частини наразі розробляється. дії розробки.

Висновки до розділу 4

У четвертому розділі виконано повну програмну реалізацію клієнтської частини системи, побудовано інфраструктуру практичного підключення AI-асистента та проведено тестування функціональної придатності.

Побудовано архітектуру Angular 17 застосунку на базі standalone-компонентів, ледачого завантаження сторінок та рольових навігаційних захисників (`Guards`), що оптимізувало мережеве навантаження та забезпечило безпеку інтерфейсу. Централізовано логіку наскрізного функціоналу мережі за допомогою ланцюжка HTTP-інтерцепторів, які в автоматичному прозорому режимі здійснюють ін'єкцію JWT-токенів, обробку помилок та безшумне поновлення прострочених сесій користувача.

Реалізовано інтерактивні аналітичні дашборди на базі Chart.js, форми транзакцій з реактивним зв'язуванням полів крос-валютної конвертації, ієрархічні дерева категорій та інтерфейси управління токенізованими запрошеннями до груп колективного бюджетування. Практично реалізовано підключення системи до зовнішнього LLM-хоста Claude Desktop через відкритий протокол MCP, що дозволило керувати особистими фінансами за допомогою природно-мовною взаємодії в асинхронному stdio-каналі.

Спроєктовано та впроваджено виробничі сценарії автоматизованого розгортання застосунку за технологією Multistage Docker Build та веб-сервера Nginx, налаштовано довготривале кешування статики з автоматичною інвалідацією через хешування імен файлів. Комплексне наскрізне тестування підтвердило повну функціональну придатність та консистентність усіх розроблених модулів фінансового ядра, колективної взаємодії та ШІ-інтеграції відповідно до вимог технічного завдання.

ВИСНОВКИ

У роботі розроблено інтелектуальну систему управління особистим і колективним бюджетом із природномовним інтерфейсом на базі протоколу MCP. Система реалізована у вигляді повноцінного вебзастосунку з REST API на Symfony 7.4 / PHP 8.2, SPA-клієнтом на Angular 17 та інтеграцією з Claude Desktop через протокол Model Context Protocol.

У першому розділі проведено аналіз предметної галузі та обґрунтовано технологічний стек. Показано, що більшість людей не ведуть систематичного обліку фінансів через трудомісткість ручного введення даних. Порівняльний аналіз трьох основних груп існуючих рішень виявив спільні недоліки: відсутність природномовного інтерфейсу, обмежений колективний бюджет та слабку підтримку гривні та мультивалютності в цілому. Обґрунтовано вибір стеку: Symfony 7.4 з DI-тегованими ітераторами та Console-компонентом для MCP-сервера, Angular 17 зі standalone-компонентами та ReactiveFormsModule, MySQL 8.0 та Docker Compose для відтворюваного середовища. Сформульовано групи функціональних і нефункціональних вимог із виділенням 5 акторів системи.

У другому розділі виконано повне UML-проєктування системи. Побудовано діаграму прецедентів, діаграму класів на двох рівнях – доменному та сервісно-MCP (ToolInterface, ToolRegistry, McpServer, McpServeCommand). Розроблено діаграми послідовностей, що охоплюють ключові сценарії: автентифікацію, оновлення JWT, створення транзакції через веб і через MCP, побудову звіту. Побудовано діаграми станів для моделювання асинхронних процесів: облікового запису, JWT-сесії, MCP-сесії та плану бюджету. Спроектовано реляційну модель бази даних із ключовими таблицями, зв'язками та описом стратегій каскадів і обмежень цілісності.

У третьому розділі описано реалізацію серверної частини. Розроблено схему аутентифікації JWT з access-токеном (TTL 3600 с) та refresh-токеном з ротацією, що зберігається як SHA-256-хеш. Реалізовано трирівневу рольову модель (candidate

→ user → admin) і авторизацію ресурсів через Symfony Security Voter. Бізнес-логіка транзакцій підтримує десять параметрів фільтрації, пагінацію та ієрархічний алгоритм визначення суми в валюті рахунку. Система колективного бюджету забезпечує повний цикл управління групами із захистом від видалення єдиного власника. MCP-сервер побудований на трьох компонентах – McpServer, ToolRegistry, McpServeCommand – і реалізує чотири інструменти; розширюваність досягається через атрибут `#[AutoconfigureTag('mcp.tool')]` без змін конфігурації. Модуль звітності надає п'ять аналітичних ендпоінтів. Алгоритм план-факт підтримує пропорційне масштабування із урахуванням реальної кількості днів місяця і року. Мультивалютність реалізована через зберігання кожної транзакції одночасно в оригінальній та рахунковій валюті зі зворотним пошуком курсів.

У четвертому розділі описано клієнтський інтерфейс і розгортання системи. Angular-застосунок побудований на standalone-компонентах із ледачим завантаженням маршрутів і трьома функціональними guards. HTTP-інтерцептори централізовано обробляють JWT та помилки. Реалізовано повний набір функціональних модулів: дашборд із підсумковими картками і кругловою діаграмою витрат, сторінка транзакцій із реактивною крос-валютною конвертацією, сторінка звітів із сімома фільтрами і трьома типами графіків, модуль планів із кольоровими прогрес-барами, управління категоріями, групами, рахунками та профілем. Підключення AI-асистента зводиться до чотирьох кроків: отримання JWT → конфігурація Claude Desktop → перезапуск → взаємодія природною мовою. Production-розгортання фронтенду реалізовано через multistage Docker build із Nginx, SPA-fallback та однорічним кешуванням статики. Перевірка функціональної придатності методом ручного тестування підтвердила коректну роботу всіх реалізованих модулів.

Таким чином, усі поставлені завдання виконано. Результатом роботи є працездатна система, розгорнута у двох середовищах і перевірена за визначеними сценаріями. Серед виявлених обмежень – ендпоінт прогнозування реалізовано як заглушку, MCP-токен має TTL 1 годину і потребує ручного оновлення, крос-

валютна конвертація недоступна через MCP, вбудований веб-чат AI спроектований на рівні схеми даних, але не реалізований у поточній версії. Перелічені обмеження намічено як напрями подальшого розвитку: підключення AI-провайдера або ынтеграцыя математичної моделі лінійної регресії для прогнозування, реалізація довгоживучого MCP-токена, вбудований AI-чат у веб-інтерфейсі та автоматична синхронізація курсів обміну з публічними провайдерами.

Практична цінність роботи полягає у застосуванні протоколу MCP – відкритого стандарту 2024 року – для реалізації природномовного інтерфейсу до фінансової системи без розробки власного чат-UI. Запропонована архітектура MCP-інтеграції є відтворюваною та може бути перенесена на інші прикладні системи, що потребують взаємодії з LLM-асистентами.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1 Kania K. Fintech and the Future of Personal Finance Management. 2023. URL: https://railwaymen.org/blog/fintech-and-the-future-of-personal-finance-management (дата звернення: 22.04.2026).

2 Kahneman D. Thinking, Fast and Slow. New York : Farrar, Straus and Giroux, 2011. 499 p.

3 Довгань Ж. М., Галіцейська Ю. М. Open-банкінг як тренд розвитку фінансових технологій. Інноваційна економіка. 2021. URL: https://inneco.org/index.php/innecoua/article/view/828/904 (дата звернення: 23.04.2026).

4 Marketing and logistics in the management system: challenges of digital globalization. Proceedings of the International Scientific and Practical Conference. Lviv : Lviv Polytechnic Publishing House, 2024. 487 p.

5 Vaswani A. et al. Attention Is All You Need. 2017. arXiv:1706.03762. URL: https://arxiv.org/abs/1706.03762 (дата звернення: 23.04.2026).

6 Brown T. B. et al. Language Models are Few-Shot Learners. 2020. URL: https://arxiv.org/abs/2005.14165 (дата звернення: 24.04.2026).

7 Anthropic. Model Context Protocol Specification. 2024. URL: https://www.anthropic.com/news/model-context-protocol (дата звернення: 24.04.2026).

8 Bubeck S. et al. Sparks of Artificial General Intelligence: Early experiments with GPT-4. 2023. arXiv:2303.12712. URL: https://arxiv.org/abs/2303.12712 (дата звернення: 27.04.2026).

9 Zandstra M. PHP 8 Objects, Patterns, and Practice. Apress, 2021. 593 p.

10 Symfony 6: The Fast Track. Official Documentation. URL: https://symfony.com/doc/6.4/the-fast-track/en/index.html (дата звернення: 27.04.2026).

11 Angular 17+ Essential Guide: Master the Revolutionary Changes That Transformed Modern Development. 2023. URL: https://javascript.plainenglish.io/angular-17-essential-guide-master-the-revolutionary-

changes-that-transformed-modern-development-6e915d170280 (дата звернення: 27.04.2026).

12 ДСТУ 8302:2015. Бібліографічне посилання. Загальні положення та правила складання. [Чинний від 2016–07–01]. Київ : ДП «УкрНДНЦ», 2016. 20 с.

13 Fowler M., Scott K. UML Distilled: A Brief Guide to the Standard Object Modeling Language. 2nd ed. Addison-Wesley, 2000. 224 p.

14 Quatrani T. Visual Modeling with Rational Rose 2002 and UML. Addison-Wesley, 2002. URL: https://books.google.com.mt/books?id=B8pt9PPqC-kC (дата звернення: 28.04.2026).

15 Martin R. C. Clean Architecture: A Craftsman's Guide to Software Structure and Design. Prentice Hall, 2017. 432 p.

16 Fowler M. Patterns of Enterprise Application Architecture. Addison-Wesley Professional, 2002. 533 p.

17 Evans E. Domain-Driven Design: Tackling Complexity in the Heart of Software. Addison-Wesley, 2003. 560 p.

18 Introduction to JSON Web Tokens. URL: https://www.jwt.io/introduction#what-is-json-web-token (дата звернення: 30.04.2026).

19 JSON RPC 2.0 Specification. 2010. URL: https://www.jsonrpc.org/specification (дата звернення: 30.04.2026).

20 IETF. RFC 7807: Problem Details for HTTP APIs. 2016. URL: https://datatracker.ietf.org/doc/html/rfc7807 (дата звернення: 30.04.2026).

21 Mastering Design Patterns in Angular: Building Scalable and Maintainable Web Applications. 2024. URL: https://roshancloudarchitect.me/mastering-design-patterns-in-angular-building-scalable-and-maintainable-web-applications-fc3d5c682942 (дата звернення: 30.04.2026).

22 Chart.js. Open Source HTML5 Charts Documentation. Version 4.x. URL: https://www.chartjs.org/docs/latest/ (дата звернення: 03.05.2026).

23 Gamma E. et al. Design Patterns: Elements of Reusable Object-Oriented Software. Addison-Wesley, 1994. 395 p.

24 Oracle. MySQL 8.0 Reference Manual. InnoDB Storage Engine Architecture & JSON Data Type Support. URL: https://dev.mysql.com/doc/refman/8.0/en/ (дата звернення: 04.05.2026).

25 Richardson C. Microservices Patterns: With examples in Java. Manning Publications, 2018. 520 p.

26 IETF. RFC 7519: JSON Web Token (JWT). 2015. URL: https://datatracker.ietf.org/doc/html/rfc7519 (дата звернення: 05.05.2026).

27 LexikJWTAuthenticationBundle. Official GitHub Repository. URL: https://github.com/lexik/LexikJWTAuthenticationBundle (дата звернення: 05.05.2026).

28 ISO 4217 Codes for the representation of currencies. International Organization for Standardization. URL: https://www.iso.org/iso-4217-currency-codes.html (дата звернення: 07.05.2026).

29 JSON Schema Specification. Core definitions and data validation. URL: https://json-schema.org/specification (дата звернення: 07.05.2026).

30 Doctrine Database Abstraction Layer (DBAL). Documentation version 3.x. URL: https://www.doctrine-project.org/projects/doctrine-dbal/en/current/index.html (дата звернення: 08.05.2026).

31 Docker Documentation. Compose specification file definition. URL: https://docs.docker.com/reference/compose-file/ (дата звернення: 08.05.2026).

32 NelmioCorsBundle Documentation. Symfony Packages. URL: https://github.com/nelmio/NelmioCorsBundle (дата звернення: 08.05.2026).

33 Angular Material Components Library. Official Documentation. URL: https://material.angular.io/components/categories (дата звернення: 09.05.2026).

34 ReactiveX. RxJS: Reactive Extensions Library for JavaScript. Observables, operators, shareReplay specification. URL: https://rxjs.dev/guide/overview (дата звернення: 09.05.2026).

35 Nginx HTTP Server. Official Documentation and Configuration Guide. URL: https://nginx.org/en/docs/ (дата звернення: 09.05.2026).

ДОДАТОК А

ПРОГРАМНА РЕАЛІЗАЦІЯ СЕРВЕРНОЇ ЧАСТИНИ

"src/Controller/Api/V1/AuthController.php"

```
<?php

declare(strict_types=1);

namespace App\Controller\Api\V1;

use App\DTO\Request\Auth\LoginRequest;
use App\DTO\Request\Auth\RefreshTokenRequest;
use App\DTO\Request\Auth\RegisterRequest;
use App\Service\Auth\AuthService;
use Symfony\Bundle\FrameworkBundle\Controller\AbstractController;
use Symfony\Component\HttpFoundation\JsonResponse;
use Symfony\Component\HttpFoundation\Request;
use Symfony\Component\Routing\Attribute\Route;
use Symfony\Component\Validator\Validator\ValidatorInterface;

#[Route('/auth')]
class AuthController extends AbstractController
{
    public function __construct(
        private readonly AuthService       $authService,
        private readonly ValidatorInterface $validator,
    ) {}

    #[Route('/register', methods: ['POST'])]
    public function register(Request $request): JsonResponse
    {
        $data = json_decode($request->getContent(), true) ?? [];
        $dto  = RegisterRequest::fromArray($data);
        $dto->validate($this->validator);

        $response = $this->authService->register($dto);

        return $this->json($response, 201);
    }

    #[Route('/login', methods: ['POST'])]
    public function login(Request $request): JsonResponse
    {
        $data = json_decode($request->getContent(), true) ?? [];
        $dto  = LoginRequest::fromArray($data);
        $dto->validate($this->validator);

        $response = $this->authService->login($dto, $request);

        return $this->json($response->toArray());
    }

    #[Route('/refresh', methods: ['POST'])]
    public function refresh(Request $request): JsonResponse
    {
        $data = json_decode($request->getContent(), true) ?? [];
```

```
        $dto  = RefreshTokenRequest::fromArray($data);
        $dto->validate($this->validator);

        $response = $this->authService->refresh($dto->refreshToken, $request);

        return $this->json($response->toArray());
    }

    #[Route('/logout', methods: ['POST'])]
    public function logout(Request $request): JsonResponse
    {
        $data = json_decode($request->getContent(), true) ?? [];
        $dto  = RefreshTokenRequest::fromArray($data);
        $dto->validate($this->validator);

        $this->authService->logout($dto->refreshToken);

        return $this->json(null, 204);
    }
}
```

"src/Service/Budget/TransactionService.php"

```
<?php

declare(strict_types=1);

namespace App\Service\Budget;

use App\DTO\Request\Transaction\CreateTransactionRequest;
use App\DTO\Request\Transaction\TransactionFilterRequest;
use App\DTO\Request\Transaction\UpdateTransactionRequest;
use App\DTO\Response\PaginatedResponse;
use App\DTO\Response\TransactionResponse;
use App\Entity\Transaction;
use App\Entity\User;
use App\Enum\TransactionType;
use App\Exception\NotFoundException;
use App\Repository\AccountRepository;
use App\Repository\CategoryRepository;
use App\Repository\GroupMemberRepository;
use App\Repository\GroupRepository;
use App\Repository\TransactionRepository;
use App\Service\Currency\ExchangeRateService;
use DateTimeImmutable;
use Doctrine\ORM\EntityManagerInterface;
use Symfony\Component\HttpKernel\Exception\UnprocessableEntityHttpException;
use Symfony\Component\Security\Core\Exception\AccessDeniedException;

class TransactionService
{
    public function __construct(
        private readonly TransactionRepository $transactionRepository,
        private readonly CategoryRepository    $categoryRepository,
        private readonly GroupRepository       $groupRepository,
        private readonly GroupMemberRepository $groupMemberRepository,
        private readonly AccountRepository     $accountRepository,
        private readonly ExchangeRateService   $exchangeRateService,
        private readonly EntityManagerInterface $em,
```

```
    ) {}

    public function list(User $user, TransactionFilterRequest $filter): PaginatedResponse
    {
        $userId = (int) $user->getId();

        $transactions = $this->transactionRepository->findByUserAndFilters($userId,
$filter);
        $total        = $this->transactionRepository->countByUserAndFilters($userId,
$filter);

        return new PaginatedResponse(
            data:  array_map(
                static fn(Transaction $t) => TransactionResponse::fromEntity($t)-
>toArray(),
                $transactions,
            ),
            total: $total,
            page:  $filter->page,
            limit: $filter->limit,
        );
    }

    public function create(CreateTransactionRequest $dto, User $user): Transaction
    {
        $category = $this->categoryRepository->find($dto->categoryId);
        if ($category === null || $category->isDeleted()) {
            throw new NotFoundException("Category #{$dto->categoryId} not found.");
        }
        if (!$category->isGlobal() && $category->getUser()?->getId() !== $user->getId()) {
            throw new NotFoundException("Category #{$dto->categoryId} not found.");
        }

        $group = null;
        if ($dto->groupId !== null) {
            $group = $this->groupRepository->find($dto->groupId);
            if ($group === null || $group->isDeleted()) {
                throw new NotFoundException("Group #{$dto->groupId} not found.");
            }
            if ($this->groupMemberRepository->findMembership((int) $group->getId(), (int)
$user->getId()) === null) {
                throw new AccessDeniedException("You are not a member of group #{$dto-
>groupId}.");
            }
        }

        $account = null;
        if ($dto->accountId !== null) {
            $account = $this->accountRepository->find($dto->accountId);
            if ($account === null || $account->isDeleted() || $account->getUser()->getId()
!== $user->getId()) {
                throw new NotFoundException("Account #{$dto->accountId} not found.");
            }
        } else {
            $account = $this->accountRepository->findDefaultForUser($user);
        }

        $currency    = $dto->currency ?? ($account?->getCurrency() ?? $user-
>getCurrency());
```

```
        $amount      = (float) $dto->amount;
        $date        = new DateTimeImmutable($dto->transactedAt);

        [$accountAmount, $exchangeRate] = $this->resolveAmounts(
            amount:        $amount,
            currency:      $currency,
            accountCurrency: $account?->getCurrency() ?? $currency,
            date:          $date,
            dtoAccountAmount: $dto->accountAmount,
            dtoExchangeRate:  $dto->exchangeRate,
        );

        $transaction = new Transaction();
        $transaction->setUser($user);
        $transaction->setGroup($group);
        $transaction->setCategory($category);
        $transaction->setAccount($account);
        $transaction->setType(TransactionType::from($dto->type));
        $transaction->setAmount(number_format($amount, 2, '.', ''));
        $transaction->setCurrency($currency);
        $transaction->setAccountAmount(number_format($accountAmount, 2, '.', ''));
        $transaction->setExchangeRate(number_format($exchangeRate, 6, '.', ''));
        $transaction->setTransactedAt($date);
        $transaction->setNote($dto->note);

        $this->em->persist($transaction);
        $this->em->flush();

        if ($exchangeRate !== 1.0 && $account !== null) {
            $this->exchangeRateService->saveIfMissing($currency, $account->getCurrency(),
$date, $exchangeRate);
        }

        return $transaction;
    }

    public function update(Transaction $transaction, UpdateTransactionRequest $dto):
Transaction
    {
        if ($dto->amount !== null) {
            $transaction->setAmount(number_format($dto->amount, 2, '.', ''));
        }
        if ($dto->type !== null) {
            $transaction->setType(TransactionType::from($dto->type));
        }
        if ($dto->categoryId !== null) {
            $category = $this->categoryRepository->find($dto->categoryId);
            if ($category === null || $category->isDeleted()) {
                throw new NotFoundException("Category #{$dto->categoryId} not found.");
            }
            $transaction->setCategory($category);
        }
        if ($dto->transactedAt !== null) {
            $transaction->setTransactedAt(new DateTimeImmutable($dto->transactedAt));
        }
        if (array_key_exists('note', (array) $dto)) {
            $transaction->setNote($dto->note);
        }
        if ($dto->currency !== null) {
```

```
            $transaction->setCurrency($dto->currency);
        }

        if ($dto->accountId !== null) {
            $newAccount = $this->accountRepository->find($dto->accountId);
            if ($newAccount === null || $newAccount->isDeleted() || $newAccount-
>getUser()->getId() !== $transaction->getUser()->getId()) {
                throw new NotFoundException("Account #{$dto->accountId} not found.");
            }
            $transaction->setAccount($newAccount);
        }

        $account = $transaction->getAccount();
        if ($account !== null && ($dto->amount !== null || $dto->currency !== null ||
$dto->accountAmount !== null || $dto->exchangeRate !== null || $dto->transactedAt !== null
|| $dto->accountId !== null)) {
            [$accountAmount, $exchangeRate] = $this->resolveAmounts(
                amount:           (float) $transaction->getAmount(),
                currency:         $transaction->getCurrency(),
                accountCurrency:  $account->getCurrency(),
                date:             $transaction->getTransactedAt(),
                dtoAccountAmount: $dto->accountAmount,
                dtoExchangeRate:  $dto->exchangeRate,
                currentRate:      (float) $transaction->getExchangeRate(),
            );
            $transaction->setAccountAmount(number_format($accountAmount, 2, '.', ''));
            $transaction->setExchangeRate(number_format($exchangeRate, 6, '.', ''));
        }

        $this->em->flush();

        if ($account !== null && (float) $transaction->getExchangeRate() !== 1.0) {
            $this->exchangeRateService->saveIfMissing(
                $transaction->getCurrency(),
                $account->getCurrency(),
                $transaction->getTransactedAt(),
                (float) $transaction->getExchangeRate(),
            );
        }

        return $transaction;
    }

    public function delete(Transaction $transaction): void
    {
        $transaction->softDelete();
        $this->em->flush();
    }

    private function resolveAmounts(
        float $amount,
        string $currency,
        string $accountCurrency,
        DateTimeImmutable $date,
        ?float $dtoAccountAmount,
        ?float $dtoExchangeRate,
        float $currentRate = 1.0,
    ): array {
        if ($currency === $accountCurrency) {
```

```
            return [$amount, 1.0];
        }

        if ($dtoAccountAmount !== null) {
            return [$dtoAccountAmount, $amount > 0 ? round($dtoAccountAmount / $amount, 6)
: 1.0];
        }

        if ($dtoExchangeRate !== null) {
            return [round($amount * $dtoExchangeRate, 2), $dtoExchangeRate];
        }

        $rate = $this->exchangeRateService->getRate($currency, $accountCurrency, $date);

        if ($rate !== null) {
            return [round($amount * $rate, 2), $rate];
        }

        if ($currentRate !== 1.0) {
            return [round($amount * $currentRate, 2), $currentRate];
        }

        throw new UnprocessableEntityHttpException(
            "No exchange rate found for {$currency}→{$accountCurrency} on {$date-
>format('Y-m-d')}. " .
            'Provide account_amount or exchange_rate manually.'
        );
    }
}
```

"src/Entity/Transaction.php"

```
<?php

declare(strict_types=1);

namespace App\Entity;

use App\Entity\Trait\SoftDeleteTrait;
use App\Entity\Trait\TimestampableTrait;
use App\Enum\TransactionType;
use App\Repository\TransactionRepository;
use DateTimeImmutable;
use Doctrine\ORM\Mapping as ORM;

#[ORM\Entity(repositoryClass: TransactionRepository::class)]
#[ORM\Table(name: 'transactions')]
#[ORM\HasLifecycleCallbacks]
#[ORM\Index(columns: ['user_id'])]
#[ORM\Index(columns: ['group_id'])]
#[ORM\Index(columns: ['category_id'])]
#[ORM\Index(columns: ['type'])]
#[ORM\Index(columns: ['transacted_at'])]
#[ORM\Index(columns: ['user_id', 'transacted_at'])]
#[ORM\Index(columns: ['group_id', 'transacted_at'])]
class Transaction
{
    use TimestampableTrait;
    use SoftDeleteTrait;
```

```
    #[ORM\Id]
    #[ORM\GeneratedValue]
    #[ORM\Column(options: ['unsigned' => true])]
    private ?int $id = null;

    #[ORM\ManyToOne(targetEntity: User::class, inversedBy: 'transactions')]
    #[ORM\JoinColumn(nullable: false, onDelete: 'CASCADE')]
    private User $user;

    #[ORM\ManyToOne(targetEntity: Group::class, inversedBy: 'transactions')]
    #[ORM\JoinColumn(nullable: true, onDelete: 'SET NULL')]
    private ?Group $group = null;

    #[ORM\ManyToOne(targetEntity: Category::class, inversedBy: 'transactions')]
    #[ORM\JoinColumn(nullable: false)]
    private Category $category;

    #[ORM\ManyToOne(targetEntity: Account::class)]
    #[ORM\JoinColumn(nullable: true, onDelete: 'SET NULL')]
    private ?Account $account = null;

    #[ORM\Column(enumType: TransactionType::class)]
    private TransactionType $type;

    #[ORM\Column(type: 'decimal', precision: 15, scale: 2)]
    private string $amount;

    #[ORM\Column(length: 3)]
    private string $currency;

    #[ORM\Column(type: 'decimal', precision: 15, scale: 2, options: ['default' =>
'0.00'])]
    private string $accountAmount = '0.00';

    #[ORM\Column(type: 'decimal', precision: 18, scale: 6, options: ['default' =>
'1.000000'])]
    private string $exchangeRate = '1.000000';

    #[ORM\Column(type: 'date_immutable')]
    private DateTimeImmutable $transactedAt;

    #[ORM\Column(type: 'text', nullable: true)]
    private ?string $note = null;

    #[ORM\Column(options: ['default' => false])]
    private bool $isPlanned = false;

    public function getId(): ?int
    {
        return $this->id;
    }

    public function getUser(): User
    {
        return $this->user;
    }

    public function setUser(User $user): static
```

```
{
    $this->user = $user;
    return $this;
}

public function getGroup(): ?Group
{
    return $this->group;
}

public function setGroup(?Group $group): static
{
    $this->group = $group;
    return $this;
}

public function getCategory(): Category
{
    return $this->category;
}

public function setCategory(Category $category): static
{
    $this->category = $category;
    return $this;
}

public function getType(): TransactionType
{
    return $this->type;
}

public function setType(TransactionType $type): static
{
    $this->type = $type;
    return $this;
}

public function getAmount(): string
{
    return $this->amount;
}

public function setAmount(string $amount): static
{
    $this->amount = $amount;
    return $this;
}

public function getCurrency(): string
{
    return $this->currency;
}

public function setCurrency(string $currency): static
{
    $this->currency = $currency;
    return $this;
}
```

```
public function getTransactedAt(): DateTimeImmutable
{
    return $this->transactedAt;
}

public function setTransactedAt(DateTimeImmutable $transactedAt): static
{
    $this->transactedAt = $transactedAt;
    return $this;
}

public function getNote(): ?string
{
    return $this->note;
}

public function setNote(?string $note): static
{
    $this->note = $note;
    return $this;
}

public function getAccount(): ?Account
{
    return $this->account;
}

public function setAccount(?Account $account): static
{
    $this->account = $account;
    return $this;
}

public function getAccountAmount(): string
{
    return $this->accountAmount;
}

public function setAccountAmount(string $accountAmount): static
{
    $this->accountAmount = $accountAmount;
    return $this;
}

public function getExchangeRate(): string
{
    return $this->exchangeRate;
}

public function setExchangeRate(string $exchangeRate): static
{
    $this->exchangeRate = $exchangeRate;
    return $this;
}

public function isPlanned(): bool
{
    return $this->isPlanned;
```

```
    }

    public function setIsPlanned(bool $isPlanned): static
    {
        $this->isPlanned = $isPlanned;
        return $this;
    }
}
```

"migrations/Version20260508000005.php"

```
<?php

declare(strict_types=1);

namespace DoctrineMigrations;

use Doctrine\DBAL\Schema\Schema;
use Doctrine\Migrations\AbstractMigration;

final class Version20260508000005 extends AbstractMigration
{
    public function getDescription(): string
    {
        return 'Plans: add is_uniform flag for proportional scaling of rolling plans';
    }

    public function up(Schema $schema): void
    {
        $this->addSql('ALTER TABLE plans ADD is_uniform TINYINT(1) NOT NULL DEFAULT 0
AFTER is_rolling');
    }

    public function down(Schema $schema): void
    {
        $this->addSql('ALTER TABLE plans DROP COLUMN is_uniform');
    }
}
```

ДОДАТОК Б

РЕАЛІЗАЦІЯ КЛІЄНТСЬКОГО ІНТЕРФЕЙСУ

"src/app/app.routes.ts"

```
import { Routes } from '@angular/router';
import { authGuard, adminGuard, guestGuard } from './core/guards/auth.guard';
import { AuthLayoutComponent } from './layout/auth-layout/auth-layout.component';
import { MainLayoutComponent } from './layout/main-layout/main-layout.component';

export const routes: Routes = [
  {
    path: 'auth',
    component: AuthLayoutComponent,
    canActivate: [guestGuard],
    loadChildren: () =>
      import('./features/auth/auth.routes').then((m) => m.AUTH_ROUTES),
  },
  {
    path: 'pending',
    loadComponent: () =>
      import('./features/auth/pending/pending.component').then((m) => m.PendingComponent),
  },
  {
    path: '',
    component: MainLayoutComponent,
    canActivate: [authGuard],
    children: [
      {
        path: 'dashboard',
        loadComponent: () =>
          import('./features/dashboard/dashboard.component').then(
            (m) => m.DashboardComponent
          ),
      },
      {
        path: 'transactions',
        loadComponent: () =>
          import('./features/transactions/transactions.component').then(
            (m) => m.TransactionsComponent
          ),
      },
      {
        path: 'accounts',
        loadComponent: () =>
          import('./features/accounts/accounts.component').then(
            (m) => m.AccountsComponent
          ),
      },
      {
        path: 'categories',
        loadComponent: () =>
          import('./features/categories/categories.component').then(
            (m) => m.CategoriesComponent
          ),
      },
      {
```

```
        path: 'groups',
        loadComponent: () =>
          import('./features/groups/groups.component').then(
            (m) => m.GroupsComponent
          ),
      },
      {
        path: 'plans',
        loadComponent: () =>
          import('./features/plans/plans.component').then(
            (m) => m.PlansComponent
          ),
      },
      {
        path: 'reports',
        loadComponent: () =>
          import('./features/reports/reports.component').then(
            (m) => m.ReportsComponent
          ),
      },
      {
        path: 'profile',
        loadComponent: () =>
          import('./features/profile/profile.component').then(
            (m) => m.ProfileComponent
          ),
      },
      {
        path: 'admin/users',
        canActivate: [adminGuard],
        loadComponent: () =>
          import('./features/admin/admin-users.component').then(
            (m) => m.AdminUsersComponent
          ),
      },
      { path: '', redirectTo: 'dashboard', pathMatch: 'full' },
    ],
  },
  { path: '**', redirectTo: 'auth/login' },
];
```

"src/app/core/services/auth.service.ts"

```
import { Injectable, inject } from '@angular/core';
import { HttpClient } from '@angular/common/http';
import { Router } from '@angular/router';
import { BehaviorSubject, Observable, map, tap } from 'rxjs';
import { environment } from '../../../environments/environment';
import { User } from '../models/user.model';
import {
  AuthResponse,
  LoginRequest,
  RefreshTokenRequest,
  RegisterRequest,
  RegisterResponse,
} from '../models/auth.model';

const ACCESS_TOKEN_KEY = 'access_token';
const REFRESH_TOKEN_KEY = 'refresh_token';
```

```
@Injectable({ providedIn: 'root' })
export class AuthService {
  private readonly http = inject(HttpClient);
  private readonly router = inject(Router);
  private readonly apiUrl = environment.apiUrl;

  private currentUserSubject = new BehaviorSubject<User | null>(this.loadUser());
  currentUser$ = this.currentUserSubject.asObservable();

  private loadUser(): User | null {
    try {
      const raw = localStorage.getItem('current_user');
      return raw ? (JSON.parse(raw) as User) : null;
    } catch {
      return null;
    }
  }

  login(credentials: LoginRequest): Observable<AuthResponse> {
    return this.http
      .post<AuthResponse>(`${this.apiUrl}/auth/login`, credentials)
      .pipe(tap((res) => this.storeSession(res)));
  }

  register(data: RegisterRequest): Observable<RegisterResponse> {
    return this.http.post<RegisterResponse>(`${this.apiUrl}/auth/register`, data);
  }

  refreshToken(): Observable<string> {
    const token = localStorage.getItem(REFRESH_TOKEN_KEY);
    const body: RefreshTokenRequest = { refresh_token: token ?? '' };
    return this.http
      .post<AuthResponse>(`${this.apiUrl}/auth/refresh`, body)
      .pipe(
        tap((res) => this.storeSession(res)),
        map((res) => res.access_token)
      );
  }

  logout(): void {
    localStorage.removeItem(ACCESS_TOKEN_KEY);
    localStorage.removeItem(REFRESH_TOKEN_KEY);
    localStorage.removeItem('current_user');
    this.clearTokenCookie();
    this.currentUserSubject.next(null);
    this.router.navigate(['/auth/login']);
  }

  getCurrentUser(): User | null {
    return this.currentUserSubject.getValue();
  }

  isAuthenticated(): boolean {
    return !!localStorage.getItem(ACCESS_TOKEN_KEY);
  }

  getAccessToken(): string | null {
    return localStorage.getItem(ACCESS_TOKEN_KEY);
```

```
  }

  updateCurrentUser(user: User): void {
    localStorage.setItem('current_user', JSON.stringify(user));
    this.currentUserSubject.next(user);
  }

  private storeSession(res: AuthResponse): void {
    localStorage.setItem(ACCESS_TOKEN_KEY, res.access_token);
    localStorage.setItem(REFRESH_TOKEN_KEY, res.refresh_token);
    this.setTokenCookie(res.access_token);
    if (res.user) {
      localStorage.setItem('current_user', JSON.stringify(res.user));
      this.currentUserSubject.next(res.user);
    }
  }

  private setTokenCookie(token: string): void {
    document.cookie = `BEARER=${token}; path=/; SameSite=Strict; Secure`;
  }

  private clearTokenCookie(): void {
    document.cookie = 'BEARER=; path=/; expires=Thu, 01 Jan 1970 00:00:00 GMT';
  }
}
```

"src/app/features/transactions/transactions.component.ts"

```
import { Component, OnInit, inject } from '@angular/core';
import { CommonModule } from '@angular/common';
import { MatTableModule } from '@angular/material/table';
import { MatPaginatorModule, PageEvent } from '@angular/material/paginator';
import { MatButtonModule } from '@angular/material/button';
import { MatIconModule } from '@angular/material/icon';
import { MatProgressBarModule } from '@angular/material/progress-bar';
import { MatDialog } from '@angular/material/dialog';
import { forkJoin } from 'rxjs';
import { TransactionService } from '../../core/services/transaction.service';
import { CategoryService } from '../../core/services/category.service';
import { GroupService } from '../../core/services/group.service';
import { AccountService } from '../../core/services/account.service';
import { AuthService } from '../../core/services/auth.service';
import { Transaction, TransactionFilter } from '../../core/models/transaction.model';
import { Category } from '../../core/models/category.model';
import { Group } from '../../core/models/group.model';
import { Account } from '../../core/models/account.model';
import { TransactionFilterComponent } from './transaction-filter/transaction-
filter.component';
import { TransactionFormComponent } from './transaction-form/transaction-form.component';
import { ConfirmDialogComponent } from '../../shared/components/confirm-dialog/confirm-
dialog.component';
import { EmptyStateComponent } from '../../shared/components/empty-state/empty-
state.component';
import { AmountDisplayComponent } from '../../shared/components/amount-display/amount-
display.component';

@Component({
  selector: 'app-transactions',
  standalone: true,
```

```
  imports: [
    CommonModule,
    MatTableModule,
    MatPaginatorModule,
    MatButtonModule,
    MatIconModule,
    MatProgressBarModule,
    TransactionFilterComponent,
    EmptyStateComponent,
    AmountDisplayComponent,
  ],
  templateUrl: './transactions.component.html',
  styleUrl: './transactions.component.scss',
})
export class TransactionsComponent implements OnInit {
  private txService = inject(TransactionService);
  private categoryService = inject(CategoryService);
  private groupService = inject(GroupService);
  private accountService = inject(AccountService);
  private authService = inject(AuthService);
  private dialog = inject(MatDialog);

  readonly currentUserId = this.authService.getCurrentUser()?.id ?? 0;

  transactions: Transaction[] = [];
  categories: Category[] = [];
  groups: Group[] = [];
  accounts: Account[] = [];

  total = 0;
  loading = false;
  filter: TransactionFilter = { page: 1, limit: 20 };

  displayedColumns = ['transacted_at', 'account', 'category', 'note', 'type', 'amount',
'actions'];

  ngOnInit(): void {
    forkJoin({
      categories: this.categoryService.getAll(),
      groups:     this.groupService.getAll(),
      accounts:   this.accountService.getAll(),
    }).subscribe(({ categories, groups, accounts }) => {
      this.categories = this.flatten(categories);
      this.groups = groups;
      this.accounts = accounts;
    });
    this.loadTransactions();
  }

  loadTransactions(): void {
    this.loading = true;
    this.txService.getAll(this.filter).subscribe({
      next: (res) => {
        this.transactions = res.data;
        this.total = res.meta.total;
        this.loading = false;
      },
      error: () => { this.loading = false; },
    });
```

```
  }

  onFilterChange(f: TransactionFilter): void {
    this.filter = { ...f, page: 1, limit: 20 };
    this.loadTransactions();
  }

  onPageChange(event: PageEvent): void {
    this.filter = { ...this.filter, page: event.pageIndex + 1, limit: event.pageSize };
    this.loadTransactions();
  }

  openForm(transaction?: Transaction): void {
    const ref = this.dialog.open(TransactionFormComponent, {
      width: '500px',
      data: { transaction, groups: this.groups },
    });
    ref.afterClosed().subscribe((saved) => {
      if (saved) this.loadTransactions();
    });
  }

  confirmDelete(tx: Transaction): void {
    const ref = this.dialog.open(ConfirmDialogComponent, {
      data: { message: `Видалити транзакцію на ${tx.amount} ${tx.currency}?` },
    });
    ref.afterClosed().subscribe((confirmed) => {
      if (confirmed) {
        this.txService.delete(tx.id).subscribe(() => this.loadTransactions());
      }
    });
  }

  private flatten(cats: Category[]): Category[] {
    return cats.reduce((acc, cat) => {
      acc.push(cat);
      if (cat.children?.length) acc.push(...this.flatten(cat.children));
      return acc;
    }, [] as Category[]);
  }
}
```

"src/app/core/interceptors/jwt.interceptor.ts"

```
import { inject } from '@angular/core';
import { HttpInterceptorFn } from '@angular/common/http';
import { catchError, switchMap, throwError } from 'rxjs';
import { AuthService } from '../services/auth.service';

export const jwtInterceptor: HttpInterceptorFn = (req, next) => {
  const authService = inject(AuthService);

  if (req.url.includes('/auth/')) {
    return next(req);
  }

  const token = authService.getAccessToken();
  const authReq = token
    ? req.clone({ setHeaders: { Authorization: `Bearer ${token}` } })
```

```
    : req;

  return next(authReq).pipe(
    catchError((error) => {
      if (error.status === 401 && token) {
        return authService.refreshToken().pipe(
          switchMap((newToken) =>
            next(req.clone({ setHeaders: { Authorization: `Bearer ${newToken}` } }))
          ),
          catchError((refreshError) => {
            authService.logout();
            return throwError(() => refreshError);
          })
        );
      }
      return throwError(() => error);
    })
  );
};
```