НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ БІОРЕСУРСІВ І
ПРИРОДОКОРИСТУВАННЯ УКРАЇНИ
ФАКУЛЬТЕТ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

ДОПУСКАЄТЬСЯ ДО ЗАХИСТУ

Завідувач кафедри
комп'ютерних наук

/ Голуб Б.Л., доц., к.т.н. /
підпис ПІБ, вчене звання і ступінь

« 2 » червня 2025 р

БАКАЛАВРСЬКА КВАЛІФІКАЦІЙНА РОБОТА

на тему:

«Програмне забезпечення фінансових розрахунків і щоденних справ»

Спеціальність 121 «Інженерія програмного забезпечення»

ОПП «Інженерія програмного забезпечення»

Гарант освітньої програми
к.т.н., доцент / Вайганг Г.О./
Науковий ступень та вчене звання підпис ПІБ

Керівник бакалаврської кваліфікаційної роботи : / Голуб Б.Л./
підпис ПІБ

Виконав: / Цибань Б.В./
підпис ПІБ

КИЇВ-2025

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ БІОРЕСУРСІВ
І ПРИРОДОКОРИСТУВАННЯ УКРАЇНИ
ФАКУЛЬТЕТ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

ЗАТВЕРДЖУЮ

Завідувач кафедри
комп'ютерних наук

_____ / Голуб Б.Л., доцент, к.т.н /
підпис

"16" грудня 2024 р.

ЗАВДАННЯ

на виконання бакалаврської кваліфікаційної роботи

студенту Цибань Богдан Васильович

Спеціальність 121 «Інженерія програмного забезпечення»

ОПП «Інженерія програмного забезпечення»

1. Тема роботи: Програмне забезпечення фінансових розрахунків і щоденних справ
Затверджена наказом ректора НУБіП України № 2249 "С" від 16.12.2024

2. Термін подання завершеної роботи на кафедру 2025 . 05 . 25
рік, місяць, число

3. Вихідні дані до роботи: опис програмного забезпечення

4. Перелік питань що розглядаються:

1. Аналіз фінансових розрахунків та щоденних справ
2. Розробка структури та реалізація бази даних
3. Проєктування та реалізація інформаційної системи
4. Впровадження та перевірка роботи інформаційної системи

Керівник бакалаврської кваліфікаційної роботи_____ / Голуб Б.Л. /
підпис ініціали та прізвище

Завдання прийняв до виконання _____ / Цибань Б.В. /
підпис ініціали та прізвище

Дата отримання завдання 2024 . 12 . 16
рік, місяць, число

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

БД — база даних

СУБД — система управління базами даних

API — інтерфейс програмування застосунків

UML — уніфікована мова моделювання

ER — сутність-зв’язок

ID — ідентифікатор

JWT — JSON Web Token

SSMS — SQL Server Management Studio

REST API — архітектурний стиль API

HTTP — протокол передачі гіпертексту

HTTPS — захищений HTTP

JSON — формат обміну даними

CDN — мережа доставки контенту

SSL — протокол безпеки передачі даних

DDoS — розподілена атака відмови в обслуговуванні

CI/CD — безперервна інтеграція / безперервне доставляння

ВСТУП

Актуальність. Ми живемо у світі, де щоденні справи, фінанси, завдання і навіть покупки поступово переходять в цифровий простір. Попри це, багато людей й досі ведуть свої бюджети у блокнотах, нотатках або в таблицях Excel. Схожа ситуація і з планування – хтось записує завдання на листку, хтось ставить будильник замість нормального нагадування. Причини бувають різні: хтось просто звик так робити, хтось не довіряє новим технологіям, а для когось цифрові інструменти здаються надто складними і непотрібними[1].

Утім, ми вже давно живемо в інформаційному суспільстві. І з кожним роком наше життя тільки пришвидшується: більше інформації, більше задач, більше відповідальності. У такому темпі важко тримати все в голові або покладатись лише на старі методи. Саме тому виникає потреба в простих, але ефективних цифрових рішеннях, які дозволяють в одному місці вести як фінансовий облік, так і організовувати особисті справи.

Однак, багато хто все ще не готовий до цього переходу. Є певна недовіра до технологій, особливо коли мова йде про гроші чи особисті справи. Люди бояться втратити дані або що хтось отримає до них доступ. Але сучасні інструменти вже мають потрібний захист, резервні копії, а ще – можливість автоматизувати рутинні речі. Завдяки цьому замість того щоб щодня витрачати час на ручні підрахунки чи переписування справ у блокнот, можна це делегувати цифровій системі.

Це не просто питання зручності, а й питання самоорганізації, ефективності, навіть психологічного спокою. Коли є чіткий план, зрозумілий облік витрат і справ, – менше стресу, більше контролю.

Мета. Метою розробки є створення зручного, надійного та доступного інструменту, який дозволяє користувачам керувати особистими фінансами та організовувати повсякденні справи в єдиному цифровому середовищі. Система має забезпечити простоту користування, швидкий доступ до інформації, а також безпеку зберігання персональних даних.

Розроблювана система має на меті подолати проблеми, пов’язані з

використанням застарілих методів – таких як записи на листочку, локальні таблиці або розрізнені сервіси, що не взаємодіють між собою. Головною перевагою є наявність під рукою єдиного місця для обліку витрат, планування завдань, та ведення нотаток.

Методи і технології. Для реалізації даного проєкту було використано сучасні технології, які забезпечують надійність, масштабованість та зручність у використанні. Клієнтська частина реалізована з використанням бібліотеки React, що дозволяє створити швидкий і динамічний інтерфейс. Серверна логіка побудована на Node.js, що забезпечує ефективну обробку запитів і взаємодію з базою даних. Як система управління базами даних застосовується Microsoft SQL Server – чудове рішення для зберігання структурованої інформації. У результаті обрані методи й технології дають змогу створити ефективний інструмент для щоденного обліку фінансів і організації справ. Це сприятиме підвищенню рівня цифрової грамотності, покращенню самоорганізації та загальному підвищенню якості повсякденного життя.

Апробація. Цибань Богдан Васильович «Застосування цифрових технологій для ведення фінансів і управління щоденними справами в умовах розвитку інформаційних технологій» II Міжнародна науково-практична конференція «Актуальні питання розвитку науки та техніки в умовах глобалізації».

Пояснювальна записка складається з кількох розділів. У першому розділі аналізується проблема, досліджується предметна область і надаються діаграми, які показують логіку системи. Другий розділ присвячений інформаційному забезпеченню: описуються структура бази даних, взаємозв’язки між таблицями та принципи організації даних. У третьому розділі акцент зроблено на виборі технологій та інструментів для реалізації застосунку, а також на структурі створеної системи. Четвертий розділ описує процес впровадження системи, приклади використання та результати перевірки працездатності. Пояснювальна записка складається з 73 сторінок і 3 додатків. В Додатку А – 3 сторінки, в Додатку Б – 1, а в Додатку В – 6.

1 АНАЛІЗ ФІНАНСОВИХ РОЗРАХУНКІВ І ЩОДЕННИХ СПРАВ

1.1 Опис процесів предметної області

Фінансовий облік часто розпочинається з реєстрації доходів і витрат. Люди отримують прибутки з найрізноманітніших джерел: заробітної плати, підробітку, пасивного доходу тощо. Водночас їхні витрати поділяються на категорії, такі як харчування, житло, транспорт, розваги та навчання[2].

Щоб контролювати свій фінансовий стан, зазвичай ведуть облік транзакцій: записують суму, тип операції (дохід чи витрата), дату й мету платежу. Ця інформація дозволяє аналізувати бюджет, оцінювати баланс, виявляти зайві витрати та напрацьовувати корисні фінансові звички.

Організація щоденних справ передбачає створення списків завдань із визначенням пріоритетів, термінів виконання й статусу. Такі завдання можуть стосуватися як побутових справ, так і роботи або навчання.

Крім завдань, нерідко плануються різноманітні події: зустрічі, нагадування, дати сплати рахунків чи виконання певних справ. Події можуть мати конкретний час і дату, а подекуди й повторюваний характер.

У повсякденному житті фінанси й завдання часто тісно переплітаються. Наприклад, оплата комунальних послуг – це водночас завдання і фінансова операція. Або ж поїздка на відпочинок може включати і витрати і підготовчі справи.

У зв’язку з цим важливо забезпечити взаємодію між обліком фінансів і управлінням завданнями. Такий підхід допоможе створити цілісну картину щоденного життя та ефективніше поєднувати різні аспекти організації рутинних процесів.

1.2 Огляд існуючих рішень

Розроблювана система є унікальним рішенням, яке поєднує два важливі функціонали: фінансовий облік та управління завданнями в одному інтерфейсі.

Згідно з проведеним аналізом, наразі на ринку відсутні інструменти, здатні інтегрувати ці два напрями в межах одного інструменту. Існують лише окремі програми та сервіси, котрі виконують кожну з цих функцій окремо:

- Фінансовий облік: Аналоги на ринку – сервіси, як-от Monefy або PocketGuard, допомагають відстежувати витрати, формувати бюджети та аналізувати фінанси.
- Управління завданнями: Такі сервіси, як Todoist або Microsoft To Do надають можливість користувачам складати списки завдань, розподіляти їх за категоріями, встановлювати дедлайни та отримувати нагадування.

Monefy – мобільний додаток для контролю персональних фінансів, який дає змогу вручну додавати транзакції, сортувати їх за категоріями, аналізувати бюджет і переглядати зручні графіки. Вирізняється простим і приємним для користувача інтерфейсом.

На рис. 1 наведено скріншот інструменту Monefy[3]:

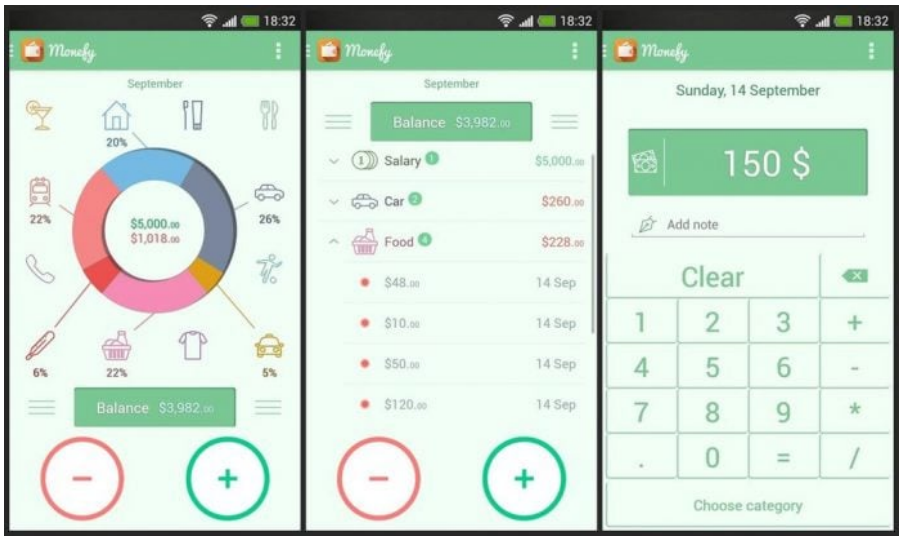


Рис. 1 Застосунок Monefy

Недоліки:

- У безкоштовній версії доступний обмежений набір функцій для аналізу витрат.
- Не передбачено управління завданнями чи подіями — лише фінансовий облік.
- Відсутня інтеграція з іншими системами чи планувальниками.

PocketGuard автоматично синхронізується з банківськими рахунками, надаючи користувачам інформацію про доступний бюджет з урахуванням фінансових зобов'язань і регулярних витрат.

На рис. 2 наведено скріншот інструменту PocketGuard[4]:

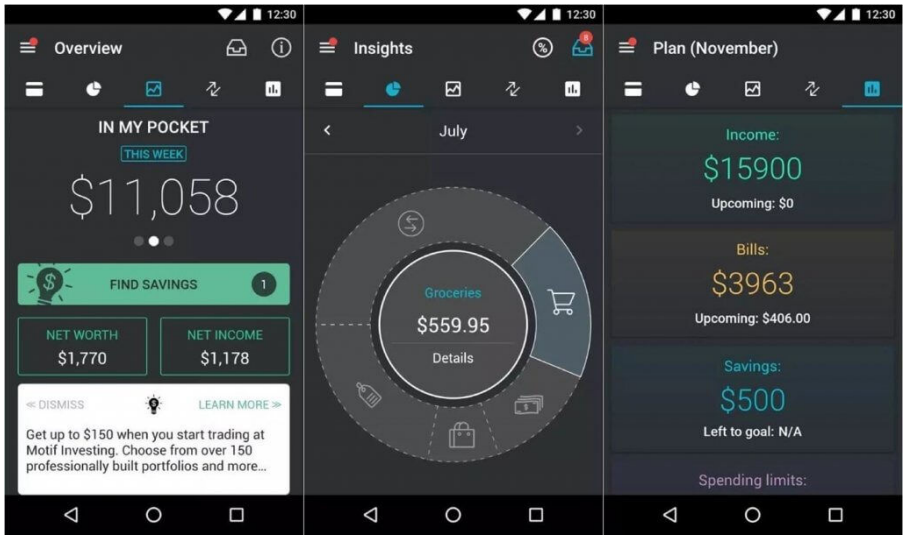


Рис. 2 Застосунок PocketGuard

Недоліки:

- Не підтримує управління завданнями або організацію подій.
- Для неангломовних користувачів інтерфейс може становити певні труднощі.
- Потребує підключення до банківських установ, що обмежує використання у певних країнах, наприклад, в Україні.
- Функціонал не виходить за межі фінансового планування.

Todoist – популярний сервіс для управління завданнями, який допомагає створювати списки справ, визначати дедлайни, встановлювати пріоритети та додавати теги. Призначений для ефективного щоденного планування.

На рис. 3 наведено скріншот інструменту Todoist[5]:

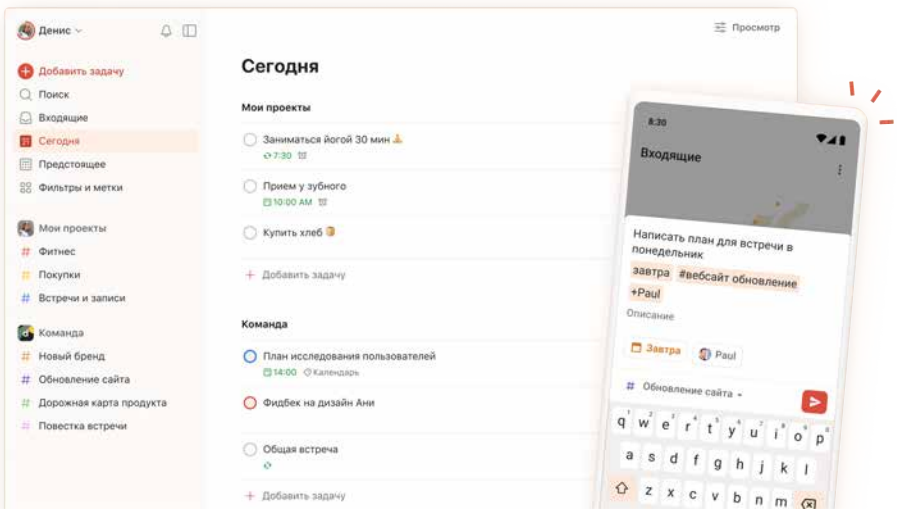


Рис. 3 Інструмент Todoist

Недоліки:

- Не передбачає ведення фінансового обліку чи аналізу бюджету.
- Інтеграція з фінансовими сервісами можлива лише через сторонні застосунки або API.
- Доступ до розширених функцій можливий лише в платній версії.

Microsoft To Do – зручний інструмент для управління завданнями, який дозволяє створювати списки справ, підзадачі та встановлювати нагадування. Інтегрується з екосистемою сервісів Microsoft і ідеально підходить для особистого планування.

На рис. 4 наведено скріншот інструменту Microsoft To Do[6]:

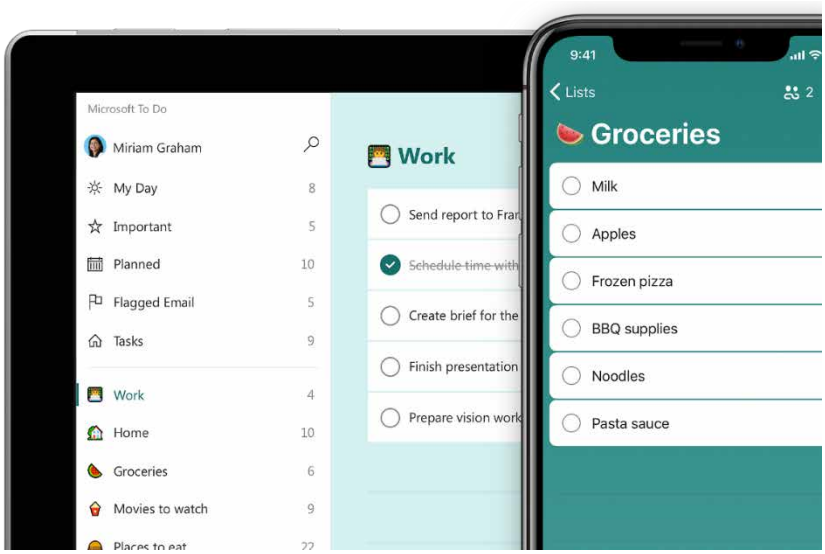


Рис. 4 Інструмент Microsoft To Do

Недоліки:

- Відсутність функціоналу для фінансового обліку.
- Новачкам може здатися складним через перевантажений інтерфейс.
- Недостатня гнучкість у візуалізації завдань (наприклад, відсутність календаря чи діаграм).
- Немає аналітики виконання задач чи ефективного розподілу часу.

Отже, жоден із наведених застосунків не забезпечує універсального підходу до управління фінансами та завданнями в одній системі. Це змушує користувачів поєднувати кілька різних сервісів, дублювати інформацію та втрачати цілісне уявлення про свій розклад і бюджет.

1.3 Постановка завдання

Огляд інструментів, що вже існують, підкреслює актуальну потребу у розробці єдиної цифрової платформи, яка дозволить:

- одночасно вести облік витрат і доходів;
- планувати справи та події;
- бачити взаємозв’язок між фінансами та завданнями;
- мати всю необхідну інформацію в одному місці.

Система для обліку особистих фінансів та організації повсякденних справ користувача повинна охоплювати наступні функції:

Фінансовий облік:

- Додавання категорій витрат і доходів (наприклад, їжа, комунальні послуги, зарплата, подарунки тощо).
- Запис транзакцій із зазначенням суми, категорії та дати (наприклад, витрати або надходження).
- Аналіз фінансових даних за обраний період (місяць, рік тощо) з можливістю перегляду загальної картини.
- Можливість редагувати або видаляти вже занесені транзакції.

Планування завдань:

- Складання списків завдань (наприклад, здійснення покупок, оплата рахунків, приготування їжі тощо).
- Установка дедлайнів для виконання завдань.
- Призначення пріоритетів або статусів завданням (наприклад, важливі, завершені, у процесі).
- Повідомлення чи нагадування про завдання в межах встановленого терміну.

Інтерфейс користувача:

- Інтуїтивний та зручний дизайн для введення та перегляду інформації.
- Функція перегляду історії транзакцій і завдань за вибрані проміжки часу.
- Гнучкі фільтри для аналізу даних за категоріями витрат, типами доходів і статусами завдань.

Безпека:

- Авторизація через логін та пароль для захисту персональних даних.
- Функціонал для резервного копіювання даних користувача.
- Використання сучасних засобів безпеки для захисту інформації.

Технічні вимоги:

- Система повинна бути доступною через сучасні веб-браузери, підтримуючи всі типи пристроїв.
- Для зберігання даних використовується SQL Server[7].
- Інтерфейс має бути розроблений за допомогою React[8], забезпечуючи динамічність і швидкість роботи.
- Безпосереднє опрацювання серверних запитів базується на Node.js[9].

Результатом стане система з функціоналом, що дозволяє тримати під контролем особисті фінанси, ефективно планувати завдання та автоматизувати

шаблонні дії. Усебічна безпека та кросплатформенність зроблять цей інструмент надійним та доступним для повсякденного використання.

1.4 Діаграма прецедентів

Діаграма прецедентів (Use Case Diagram) є ефективним засобом візуалізації функціональних можливостей системи з точки зору її користувачів або зовнішніх учасників. Вона дозволяє наочно представити взаємодію між системою та зовнішніми суб’єктами, а також визначити список дій, доступних для виконання[10].

Основна мета створення такої діаграми полягає у формуванні чіткого розуміння та формалізації вимог до системи на ранніх етапах проєктування.

Основні елементи діаграми включають:

- Актори – зовнішні суб’єкти, які взаємодіють із системою, наприклад, користувачі або адміністратори.
- Прецеденти (випадки використання) — дії або сценарії, які актори виконують за допомогою системи, такі як "Додавання записів" чи "Перегляд звітів".
- Зв’язки – відображають процеси взаємодії акторів із функціональними можливостями системи.
- Система – чітко окреслена область, у рамках якої здійснюються зазначені операції.

У контексті розроблюваного проєкту діаграма прецедентів слугує важливим інструментом для відображення функціоналу системи. Вона демонструє можливості, серед яких ведення обліку витрат, планування завдань, редагування даних, реєстрація та авторизація користувача тощо. Завдяки такій структурі стає зрозумілою логіка взаємодії користувачів із системою, що сприяє більш чіткому плануванню її реалізації.

Актори:

Власник фінансів.

Основний користувач системи відповідає за управління фінансами,

створення завдань, записів і подій, а також аналіз статистичних даних. Він фіксує інформацію про свої доходи та витрати, формує списки справ і переглядає аналітику.

Прецеденти власника фінансів:

- Додати витрати/дохід – додає запис про отриманий дохід або витрату.
- Керувати витратами/доходами – змінює/видаляє існуючі запис про отриманий дохід або витрату.
- Налаштувати категорії, рахунки, валюти – змінює налаштування фінансового обліку під свої потреби.
- Додати категорію – створює нову категорію витрат/доходів.
- Переглянути фінансовий баланс – отримує дані про стан фінансів.
- Переглянути статистику фінансів – переглядає графіки та аналітику операцій.
- Додати заплановану подію – додає заплановану подію.
- Переглянути список подій – отримує список подій.
- Позначити подію виконаною – відмічає виконання події.
- Керувати запланованими подіями – змінює/видаляє заплановану подію.
- Додати запис/завдання – створює новий запис чи завдання.
- Керувати записами/завданнями – змінює/видаляє дані у записах чи завданнях.
- Переглянути список записів/завдань – отримує список створених записів і завдань.
- Позначити завдання виконаним – відмічає завдання як виконане.

Конфігуратор.

Управляє налаштуваннями фінансової системи.

Прецеденти конфігуратора:

- Керувати категоріями доходів – створює або редагує категорії для доходів.

- Керувати категоріями витрат – створює або редагує категорії для витрат.
- Керувати фінансовим аналізом – налаштовує правила для аналізу та генерації фінансових звітів.
- Керувати типами записів – налаштовує типи записів для завдань, подій та фінансів.

Діаграма прецедентів представлена у Додатку А (сторінка 1).

1.5 Діаграма послідовності

Діаграма послідовності (Sequence Diagram) слугує для відображення порядку взаємодії між об'єктами або компонентами системи в часовій послідовності. Вона показує, як інформація переміщується між різними елементами під час виконання певної операції[11].

Основна мета діаграми – створити деталізоване відображення конкретного сценарію роботи системи, показуючи задіяні об'єкти та послідовність їх взаємодії через обмін запитами та відповідями.

Основні компоненти:

- Об’єкти/учасники – користувачі, сервіси, чи інші частини системи, наприклад, сервер або база даних.
- Життєві лінії – вертикальні лінії, що відображають існування об’єкта протягом певного часу.
- Повідомлення – горизонтальні стрілки, що символізують запити, відповіді чи виклики методів.
- Активність – прямокутники на життєвій лінії, які позначають виконання конкретної операції.

У межах проєкту ця діаграма використовується для відображення логіки внутрішніх процесів, таких як додавання фінансового запису, користувача завантаження списку завдань, перегляд звітної інформації тощо. Завдяки цьому чітко визначається, яка частина системи виконує певну функцію та які дії відбуваються в конкретній послідовності.

Діаграма послідовності представлена у Додатку А (сторінка 2).

1.6 Діаграма активності.

Діаграма активності (Activity Diagram) призначена для моделювання алгоритмів, робочих процесів або послідовностей дій у системі. Вона відображає поведінку як системи, так і користувачів, демонструючи етапи виконання дій, альтернативні шляхи, розгалуження та цикли[12].

Основним призначенням такої діаграми є наочне представлення логіки процесів. Вона часто використовується для опису сценаріїв застосування або внутрішніх бізнес-процесах, дозволяючи краще розуміти їхній фукнціонал.

Основні елементи:

1) початковий вузол – старт процесу, позначений чорним колом;
2) дія – виконання певної операції, відображається прямокутником із заокругленими кутами;
3) потік керування – стрілка, яка вказує напрямок руху процесу;
4) вузол прийняття рішення (розгалуження) – ромб, що містить умови або ні, із вибором одного з можливих шляхів на виході;
5) злиття (Merge) – ромб, який об'єднує альтернативні шляхи в один спільний;
6) паралелізм (Fork/Join) – товста горизонтальна чи вертикальна лінія, що використовується для розділення або об’єднання паралельних дій;
7) кінцевий вузол – завершення процесу, позначається чорним колом із білим контуром або двома концентричними колами.

У рамках проєкту діаграма активності використовується для опису процесів, таких як облік витрат, створення завдань, редагування записів, перегляд звітності тощо. Вона допомагає зрозуміти не лише функціональність системи, а й детально показує, як саме ці функції реалізуються, враховуючи всі умови та етапи, включно з повторюваними.

Діаграма активності представлена у Додатку А (сторінка 3).

2 РОЗРОБКА СТРУКТУРИ ТА РЕАЛІЗАЦІЯ БАЗИ ДАНИХ

2.1. Відомості про ER-діаграму

При розробці інформаційних систем структура даних має вирішальне значення, визначаючи способи зберігання, організації та обробки інформації. Для візуального представлення цих взаємозв’язків застосовується ER-діаграмах[13] (діаграма «сутність–зв’язок»), яка є інструментом моделювання, що допомагає визначити ключові об'єкти системи та логіку їхньої взаємодії.

Сучасні системи, особливо ті, що працюють із персональними або фінансовими даними, управляють численними сутностями. У фінансовій системі основними сутностями є Власник фінансів, Транзакції, Категорії, Щоденник та Рахунок. Кожна з цих сутностей має свої атрибути: для користувача це ім'я, електронна пошта та пароль, а для транзакції – назва, сума та дата.

Важливим є також моделювання взаємозв’язків між об'єктами. Жоден об'єкт не існує окремо: транзакції пов'язані з певними користувачами, події зв’язані з датами, а витрати – з категоріями. ER-діаграма відображає ці логічні асоціації, вказуючи не лише наявність зв’язків, але й також їхню природу, як-от «один до багатьох» чи «багато до багатьох», що є важливим для побудови реляційної моделі бази даних.

ER-діаграма виступає своєрідним містком між аналізом бізнесових процесів і технічною реалізацією системи. Вона дозволяє зрозуміти, як дані структуровані в таблицях, які ключі використовуються та як дані взаємопов'язані. Це сприяє виявленню надлишковості даних та їх дублюванню на етапі проектування, забезпечуючи таким чином цілісність і ефективність інформаційної структури.

Такий підхід особливо важливий для динамічних і взаємопов'язаних систем, таких як фінансове управління та керування операціями. У схожих системах сутності тісно переплітаються: наприклад, одна категорія одночасно стосується кількох транзакцій. ER-діаграма враховує всі ці аспекти на ранніх етапах розробки, що сприяє створенню гнучкої та масштабованої структури даних.

Під час проєктування структури також важливо забезпечити дотримання третьої нормальної форми (ЗНФ). Це допомагає усунути транзитивні залежності, мінімізувати дублювання даних і підвищити логічну цілісність бази.

2.2. Побудова ER-діаграми

Перед початком впровадження інформаційної системи необхідно ретельно змоделювати її структурні дані. Одним із найважливіших етапів проєктування бази даних є безпосередньо створення ER-діаграми, яка наочно відображає всі сутності предметної області, їхні атрибути та логічні зв’язки між ними[13].

Після аналізу основних сутностей та їх взаємозв’язків було створено візуальну модель – ER-діаграму, що демонструє логічну структуру інформаційної системи. Вона є фундаментом для розробки реляційної бази даних, оскільки чітко структурує ключові об’єкти системи, їхні характеристики та взаємозв’язки.

Головною метою розробки ER-діаграми є формалізувати архітектуру системи управління фінансами та завданнями. Такий підхід дозволяє враховувати унікальні особливості кожної сутності, чітко визначати взаємозв’язки між ними й уникати логічних суперечностей у структурі. Це надзвичайно важливо для забезпечення цілісності, масштабованості системи та ефективності обробки даних.

При побудові структури бази дотримувалися основних принципів третьої нормальної форми (ЗНФ). У таблицях відсутні повторювані групи атрибутів, відсутні часткові та транзитивні залежності, а всі неключові атрибути залежать виключно від первинного ключа. Це свідчить про відповідність структури бази даних ЗНФ. Завдяки цьому кожна сутність має чітко визначену роль, а атрибути зберігаються лише у відповідних таблицях, що суттєво спрощує процес оновлення та обслуговування бази даних, зменшує дублювання інформації та забезпечує логічну узгодженість даних.

Структура бази даних вебзастосунку включає таблицю OwnersFinances, яка містить основну інформацію про користувачів: унікальний ідентифікатор, ім’я, електронну адресу, зашифрований пароль і дату реєстрації. Вона є основним

елементом у встановленні зв’язків із фінансовими та організаційними компонентами системи.

Таблиця AccountType забезпечує класифікацію фінансових рахунків за типами: «Готівка» чи «Картка». Ця категоризація реалізована через зовнішній ключ у таблиці Account, яка зберігає назву рахунку, тип, баланс, валюту та ідентифікатор користувача.

Для групування транзакцій у системі використовується таблиця Category, яка містить інформацію про назву категорії, її тип (дохід чи витрата), іконку і поле OwnerID. Поле OwnerID є ключовим для визначення приналежності категорії: якщо воно заповнене, категорія прив’язана до конкретного користувача; якщо ж значення NULL, категорія є загальнодоступною та доступною для всіх користувачів. Аналогічний підхід застосовується в таблицях Account, Transactions та Operation.

Значення поля OwnerID займає важливе місце в загальній логіці роботи системи, забезпечуючи підтримку багатокористувацького використання. Кожен запис у фінансових та організаційних таблицях може бути пов’язаний із конкретним користувачем. Це дозволяє гарантувати конфіденційність персональних даних і обмежує доступ до інформації лише авторизованим користувачам. Водночас система підтримує загальні записи завдяки полю OwnerID, яке залишено незаповненим. Такі записи стають доступними для всіх користувачів.

Категорії використовуються для реєстрації транзакцій у таблиці Transactions, яка включає такі дані, як сума, опис, дата, рахунок, категорія та відповідний користувач. Крім фінансових операцій, у системі також передбачена робота із текстовими операціями, що зберігаються у таблиці Operation. Усі ці активності пов’язують дії користувача з конкретними записами, завданнями або подіями.

Саме тому однією з головних особливостей ER-діаграми є її здатність інтегрувати індивідуальні дані користувачів із загальними елементами системи. Це робить структуру універсальною як для індивідуального, так і для спільного

використання декількома користувачами. Така гнучкість особливо важлива у складних системах із численними взаємозалежностями, де необхідно забезпечити баланс між приватністю та спільним доступом до даних.

Всі взаємозв’язки між сутностями були ретельно продумані відповідно до логіки функціонування системи. Зокрема, транзакції пов’язані із рахунками, категоріями та користувачами; операції асоціюють конкретні типи операцій із діями користувачів. Така структура забезпечує інтеграцію всіх компонентів у єдину цілісну інформаційну модель.

ER-діаграма, що представлена на рис. 5, детально демонструє всі сутності системи з основними атрибутами та зовнішніми ключами, які формують логіку зв’язків. Це дозволяє чітко відобразити структуру таблиць для реалізації в реляційній базі даних Microsoft SQL Server.

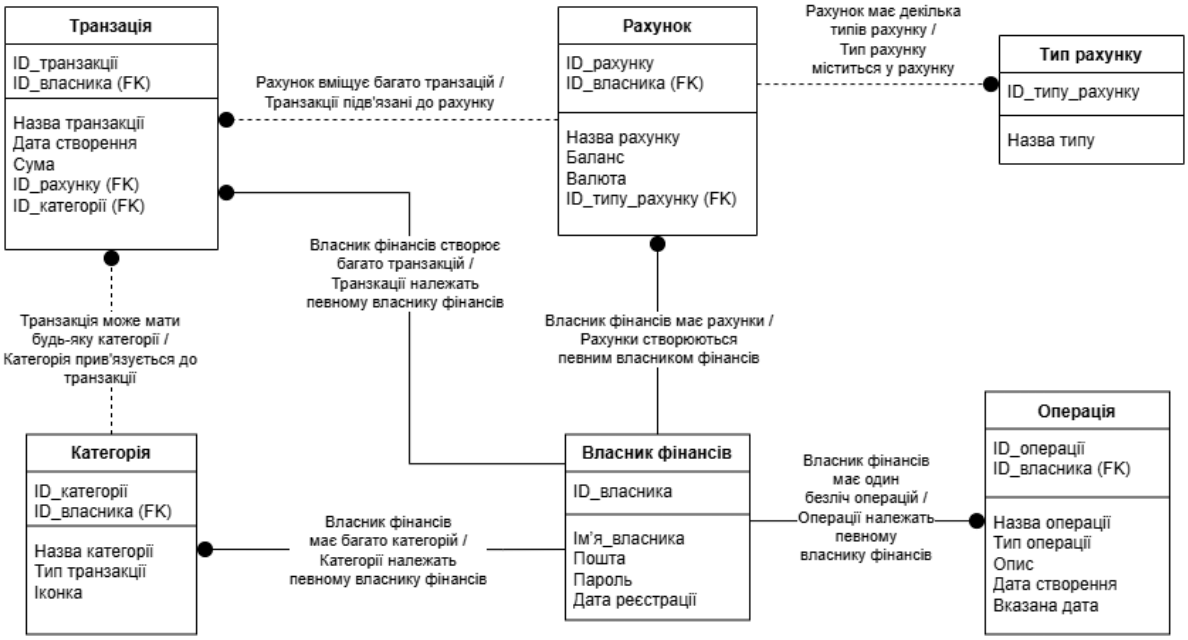


Рис. 5 ER-діаграма інформаційної бази

Опис сутностей та зв'язків.

Власник фінансів:

- Первинний ключ: ID_власника
- Атрибути: Ім’я_власника, Пошта, Пароль, Дата_реєстрації
- Центральна сутність, що зв'язує інші об'єкти з користувачем.

Рахунок:

- Первинний ключ: ID_рахунку
- Зовнішній ключ: ID_власника (зв’язок із Власником), ID_типу_рахунку (зв’язок із Типом рахунку)
- Атрибути: Назва_рахунку, Баланс, Валюта

Тип рахунку:

- Первинний ключ: ID_типу_рахунку
- Атрибути: Назва_типу

Транзакція:

- Первинний ключ: ID_транзакції
- Зовнішні ключі: ID_власника, ID_рахунку, ID_категорії
- Атрибути: Назва_транзакції, Дата_створення, Сума

Категорія:

- Первинний ключ: ID_категорії
- Зовнішній ключ: ID_власника
- Атрибути: Назва_категорії, Тип_транзакції, Іконка

Операція:

- Первинний ключ: ID_операції
- Зовнішній ключ: ID_власника
- Атрибути: Назва_операції, Тип_операції, Опис, Дата_створення, Вказана_дата

Опис зв'язків між сутностями.

Власник фінансів – Рахунок:

- Тип зв’язку: один-до-багатьох (1:N)
- Один власник може мати кілька рахунків.
- Зв’язок ідентифікуючий.

Власник фінансів – Категорія:

- Тип зв’язку: один-до-багатьох (1:N)
- Кожен користувач має свої категорії доходів та витрат.

- Зв’язок ідентифікуючий.

Власник фінансів – Операція:

- Тип зв’язку: один-до-багатьох (1:N)
- Кожен користувач створює власні події, завдання тощо.
- Зв’язок ідентифікуючий.

Тип рахунку – Рахунок:

- Тип зв’язку: один-до-багатьох (1:N)
- Один тип рахунку може відповідати багатьом рахункам.
- Зв’язок неідентифікуючий.

Рахунок – Транзакція:

- Тип зв’язку: один-до-багатьох (1:N)
- На один рахунок може припадати багато транзакцій.
- Зв’язок неідентифікуючий.

Категорія – Транзакція:

- Тип зв’язку: один-до-багатьох (1:N)
- Кожна транзакція належить до певної категорії.
- Зв’язок неідентифікуючий.

Усі зв’язки в ER-діаграмі мають тип один-до-багатьох (1:N), що відповідає реальній логіці функціонування системи, де один користувач взаємодіє з багатьма об’єктами в межах своєї фінансової активності. Така структура забезпечує цілісність даних, уніфіковане управління записами та масштабованість системи.

2.3. Вибір та обґрунтування СУБД

Одним із ключових етапів розробки інформаційної системи є визначення системи управління базами даних (СУБД), яка відповідатиме за зберігання, обробку та доступ до даних. Вибір СУБД базується на архітектурних особливостях системи, функціональних завданнях, очікуваних навантаженнях, а також характеристиках надійності, продуктивності, безпеки та сумісності з іншими компонентами.

Проведений аналіз ER-діаграми підтвердив, що база даних повинна бути організована за принципами реляційної моделі. Сутності з визначеними атрибутами та чіткі логічні зв’язки між ними реалізуються за допомогою первинних і зовнішніх ключів. Така структура дозволяє зберігати інформацію у вигляді таблиць, забезпечуючи узгодженість, цілісність даних та мінімізацію дублювання.

Основні особливості реляційної моделі включають:

- використання таблиць як базових одиниць зберігання, де кожен рядок є записом, а кожен стовпець — окремим атрибутом;
- первинний ключ (PK) як унікальний ідентифікатор запису;
- зовнішній ключ (FK), який встановлює зв’язки між таблицями.

Типи зв’язків у реляційній моделі:

- один до одного (1:1) — відповідність користувача його персональному профілю;
- один до багатьох (1:N) — один користувач має безліч транзакцій.

Для реалізації бази даних обрано Microsoft SQL Server[7] — функціональну реляційну СУБД, широко використовувану в корпоративному та промисловому середовищах.

Аргументація вибору:

- практичний досвід з СУБД скоротив час впровадження і налаштування системи завдяки ранішньому ознайомлені з SQL Server;
- сумісність із Windows 11 спрощує процес установки, адміністрування й подальшого обслуговування;
- інтуїтивний інтерфейс SQL Server Management Studio дозволяє адмініструвати базу навіть із базовими знаннями SQL;
- забезпечення високої стабільності й продуктивності;
- розширені можливості безпеки через функції шифрування, аудит змін, контроль доступу та гнучке управління ролями відповідно до сучасних стандартів захисту інформації.

Додаткові переваги включають:

- використання механізмів процедур, тригерів, представлень і індексів для підвищення ефективності;
- інтеграція з іншими продуктами Microsoft, такими як Visual Studio, Power BI та Excel, полегшує аналіз даних і розробку;
- можливість масштабування завдяки технологіям реплікації, кластеризації та розподільчої обробки даних;
- доступ до великої кількості документації та активної спільноти для отримання підтримки і вирішення питань.

Таким чином, використання Microsoft SQL Server[7] для створення інформаційної системи є обґрунтованим рішенням. Цей варіант поєднує продуктивність, надійність, зручність у розробці та можливості масштабування, що забезпечує стабільну роботу системи, безпечне зберігання даних і готовність до подальшого розвитку проєкту.

2.4. Створення БД

Для реалізації функціональних можливостей програмного рішення була розроблена та впроваджена реляційна база даних, яка забезпечує надійне збереження, обробку і захист даних користувачів. Як сказано раніше для серверної частини було обрано систему керування базами даних Microsoft SQL Server.

Основна мета створення бази даних:

- забезпечення централізованого й структурованого управління особистими фінансами та завданнями користувачів;
- визначення чітких зв’язків між елементами системи (користувачі, рахунки, категорії, транзакції, операції);
- надання можливості персоналізації — кожен користувач отримує власні дані із доступом до спільних ресурсів, як-от загальні категорії;

Процес створення бази даних включав кілька послідовних етапів: визначення сутностей, проєктування логічної структури, встановлення зв’язків

між таблицями, а також написання SQL-запитів для формування об’єктів бази даних. У результаті було створено шість основних таблиць: OwnersFinances, AccountType, Account, Category, Transactions та Operation.

Першочергово в середовищі Microsoft SQL Server Management Studio була створена нова база даних FinDoDB, яка слугує основою всієї інформаційної системи. Процес створення БД представлено на рис. 6. У її складі всі необхідні таблиці, зв’язки між ними, а також механізми для зберігання, обробки та аналізу фінансових даних користувачів.

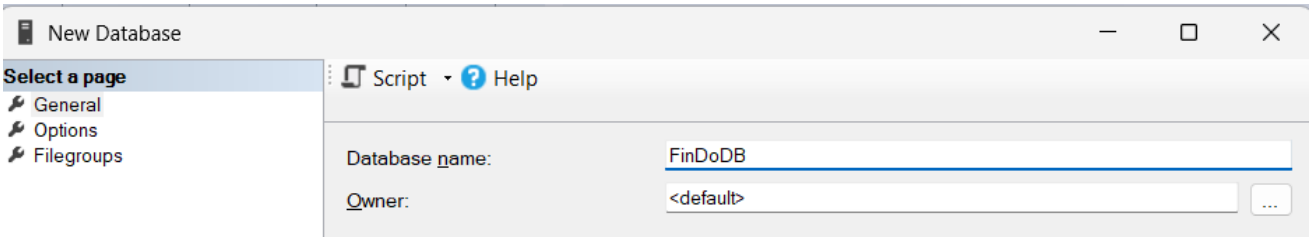


Рис. 6 Створення бази даних FinDoDB

Ключовою складовою бази є таблиця OwnersFinances, яка зберігає інформацію про власників рахунків. Вона містить ідентифікатор, логін у вигляді електронної пошти, ім’я користувача та дату реєстрації. Для забезпечення безпеки зберігання паролів таблиця включає поле Password із достатньою довжиною для хешованих значень.

Користувачі можуть створювати фінансові рахунки, які зберігаються у таблиці Account. Створені рахунки прив’язані до конкретного користувача і мають такі характеристики, як назву, баланс, тип рахунку і валюта. Типи рахунків (Готівка або Банківський рахунок) визначаються у таблиці AccountType, що дає змогу легко оновлювати список типів без перероблення структури системи.

Для класифікації даних створено таблицю Category. Вона містить як загальнодоступні категорії (без прив’язки до конкретного користувача), так і персональні. Кожна категорія має атрибути типу (дохід чи витрата) та візуального відображення.

Фінансові транзакції фіксуються в таблиці Transactions. Тут зберігається інформація про суму, назву й дату транзакції, при цьому кожен запис

пов'язується з існуючими рахунками та категоріями за допомогою зовнішніх ключів.

Окрім фінансових даних, система підтримує текстові операції, такі як звичайні записи, планування завдань чи подій. Для цього слугує таблиця Operation, де відображаються події (Event), завдання (Task) та загальні записи (Record). Завдяки атрибуту OperationType реалізовано єдину структуру для цих трьох типів, що дозволяє уникнути дублювання.

Як було виділено раніше, однією з ключових особливостей бази є розмежування доступу до даних. У кожній основній таблиці використовується атрибут OwnerID для визначення прав доступу до записів і забезпечення ізоляції даних між користувачами для персоналізації даних.

Для зміцнення цілісності даних застосовано зовнішні ключі й перевірки типу CHECK, що означає, що формат електронної пошти перевіряється на рівні бази через умову (Email LIKE '%_@__%.__%'), щоб уникнути збереження некоректної інформації.

Структура бази даних створена за допомогою SQL-запитів у середовищі SQL Server Management Studio. У процесі послідовно формувалися таблиці, задавалися їх атрибути, зв’язки та обмеження. Усі запити виконано з дотриманням вимог нормалізації даних, що гарантує ефективне використання ресурсів пам’яті, мінімізує дублювання даних і забезпечує можливість масштабування системи.

У Додатку Б (сторінка 1) представлено всі SQL-запити для створення кожної таблиці бази даних. Враховано і представлено визначення первинних та зовнішніх ключів, обмежень, типів даних, а також значень, що є базовими.

На рис. 7 представлено назви створених таблиць в базі даних.

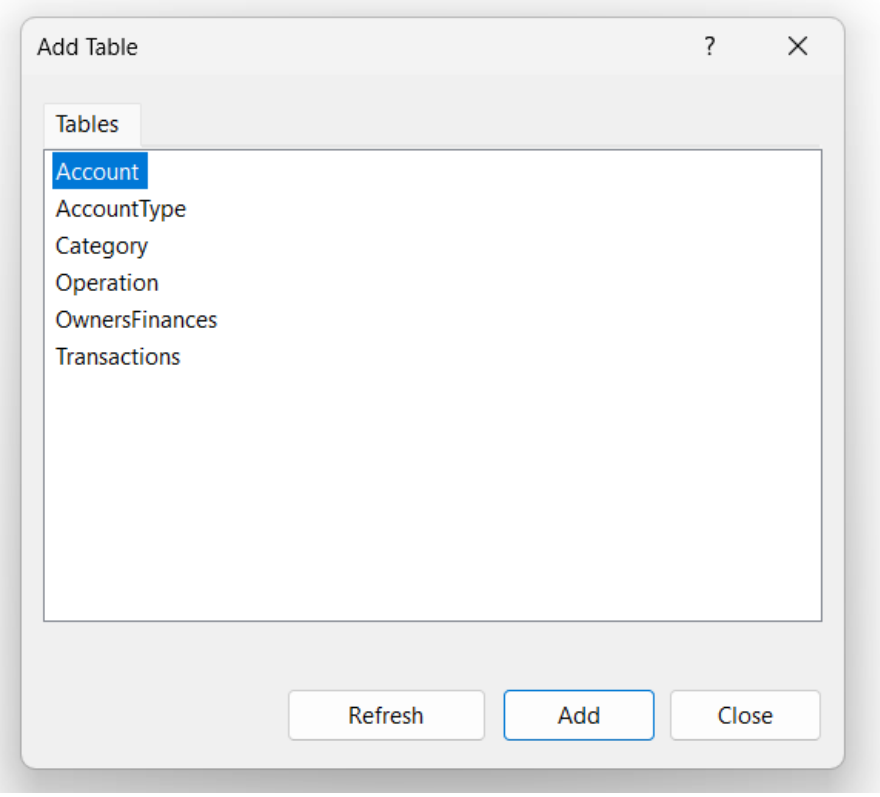


Рис. 7 Створені таблиці

На рис. 8 зображено фізичну реалізацію бази дпних FinDoDB, розроблену у середовищі Microsoft SQL Server Management Studio.

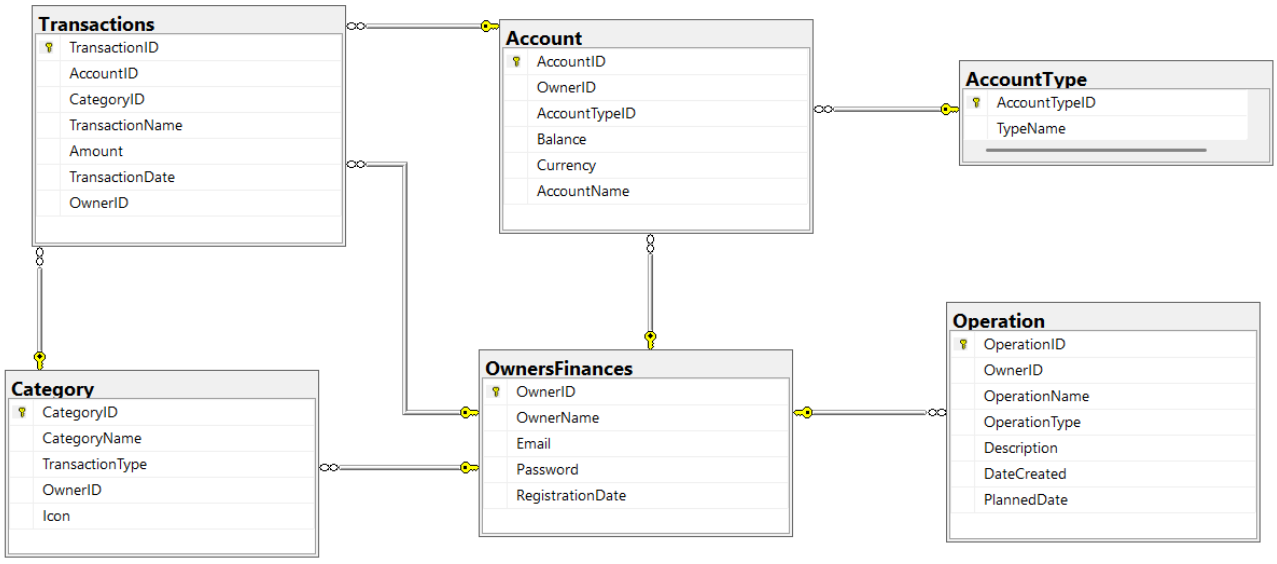


Рис. 8 – Фізична реалізація бази даних

Модель демонструє таблиці, що були створені в бд, які призначені для збереження інформації про користувачів, їх фінансові рахунки, транзакції, операції, категорії та типи рахунків. Крім того, представлено зв’язки між

таблицями через зовнішні ключі, що сприяє забезпеченню цілісності даних і логічної структури системи.

Зв’язки між таблицями:

- Transactions має зв’язки з таблицями Account, Category і OwnersFinances, забезпечуючи комплексний зв'язок між транзакціями та іншими елементами;
- Account пов’язана з таблицями AccountType, Transactions і OwnersFinances, що дозволяє деталізувати структуру рахунків та власників;
- Category, Account, Transactions і Operation мають атрибут OwnerID, яке дає можливість створювати персоналізовані дані кожному користувачу.

Загальний дизайн моделі спрямований на інтеграцію даних та забезпечення їх логічної взаємодії між компонентами системи.

3 ПРОЄКТУВАННЯ ТА РЕАЛІЗАЦІЯ ІНФОРМАЦІЙНОЇ СИСТЕМИ

3.1 Вибір інструментарію для розробки програмного забезпечення

Ефективна розробка програмного забезпечення значною мірою залежить від ретельно підібраного набору інструментів, які повинні відповідати функціональним вимогам проєкту, забезпечувати масштабованість, безпеку, зручність у розробці та подальшій підтримці системи. У рамках реалізації дипломного проєкту було використано сучасний та надійний стек технологій, що охоплює клієнтську й серверну частини, систему управління базами даних, інструменти для тестування та середовище розробки.

Клієнтська частина (Frontend).

Для створення вебінтерфейсу було використано React.js[8] – популярну JavaScript-бібліотеку, яка пропонує високу продуктивність та забезпечує зручність побудови користувацького інтерфейсу. Головними перевагами є компонентний підхід, що дозволяє перевикористовувати код, та широка підтримка спільноти. Використання React забезпечує швидке оновлення окремих елементів сторінки без необхідності її повного перезавантаження, що суттєво покращує зручність взаємодії користувача з вебзастосунком. Ця бібліотека дозволяє легко впроваджувати складні динамічні компоненти й оптимізує процес розробки.

Для оптимізації розробки було обрано Vite — інноваційний інструмент для збору та запуску фронтенд-застосунків. Він забезпечує миттєвий старт локального сервера, гаряче оновлення модулів і значно підвищену швидкість роботи завдяки використанню нативних можливостей браузера під час розробки[14].

Інтеграція React із Vite дозволила:

- зменшити час на налаштування проєкту;

- прискорити процес розробки завдяки гарячому оновленню компонентів;
- забезпечити модульну та добре структуровану архітектуру додатка;
- безперешкодно працювати з сучасними можливостями JavaScript, такими як стрілкові функції, деструктуризація, модулі тощо.

Поєднання React і Vite стало надійною основою для створення динамічного, масштабованого та зручного у використанні вебінтерфейсу.

Для реалізації клієнтської частини вебзастосунку було обрано мову програмування JavaScript, використовуючи сучасний стандарт ES6+. Завдяки таким нововведенням, як стрілкові функції, модулі й деструктуризація об'єктів та масивів, написання, читання і підтримка коду стають значно простішими. Це сприяє створенню чистого, ефективного й масштабованого програмного забезпечення.

Для стилізації сторінок застосовується стандартний CSS без інтеграції додаткових фреймворків. Вибір на користь класичного підходу до стилізації пояснюється прагненням оптимізувати структуру стилів і зменшити залежність від сторонніх інструментів. Це забезпечує більшу гнучкість у процесі розробки і надає можливість детальніше налаштовувати кожен аспект зовнішнього вигляду.

Для відображення фінансових даних і динамічних показників, таких як витрати, доходи чи залишки на рахунках, у вебзастосунках застосовується бібліотека Chart.js. Вона дозволяє створювати різноманітні типи графіків, включаючи лінійні, кругові, стовпчикові та радарні, які без проблем інтегруються у React-компоненти. Бібліотека пропонує великий набір інструментів для налаштування зовнішнього вигляду графіків.

Однією з ключових переваг Chart.js є її адаптивність. Графіки автоматично підлаштовуються під розміри екрана чи контейнера, де вони розміщені[15]. Це дозволяє забезпечити коректне відображення даних на будь-якому пристрої — від настільних моніторів до смартфонів.

Іншою важливою особливістю є реактивність графіків, тобто їх здатність оновлюватися в реальному часі за зміни даних. Використання REST API[16]

разом із механізмами управління станом у React (наприклад, через useState, useEffect) робить цей процес автоматизованим. Наприклад, коли користувач створює нову транзакцію, сервер опрацьовує POST-запит, зберігає отриману інформацію в базі даних, а клієнтська частина через GET-запит оновлює стан компонента графіка.

Цей підхід дозволяє оновлювати графіки динамічно без необхідності перезавантажувати сторінку. Це сприяє:

- Актуальності інформації – користувач завжди бачить найновіші дані.
- Зручності взаємодії – інтерфейс працює швидко та плавно, не потребуючи зайвих дій з боку користувача.
- Оптимізації продуктивності – оновлюється лише конкретна частина інтерфейсу, що зменшує навантаження на клієнтську систему.

Для фінансових діаграм така інтерактивність і зрозумілість є вкрай важливими. Завдяки цьому користувачі можуть в реальному часі аналізувати стан рахунків, структуру витрат і приймати обґрунтовані рішення на основі чіткої та зручної візуалізації.

Серверна частина (Backend).

Node.js[9] – це серверне середовище виконання JavaScript. Він був обраний для реалізації серверної частини веб-додатку. Великою перевагою Node.js є його керована подіями асинхронна модель вводу-виводу, яка дозволяє справді ефективно обробляти велику кількість одночасних клієнтських запитів, що дуже важливо для сучасних веб-додатків, де багато користувачів можуть одночасно взаємодіяти з системою, надсилаючи дані, отримуючи оновлення та, як правило, не стримуватись через затримки. Асинхронність дозволяє серверу починати обробляти нові запити, не чекаючи завершення попередніх, що значно підвищує продуктивність служби та масштабованість.

API сервера було створено з використанням Express.js[17] – мінімалістичної та гнучкої структури для Node.js, яка значно спрощує розробку серверної логіки. Він пропонує маршрутизацію у зручний спосіб, тобто визначення маршрутів, через які запити клієнта обробляються відповідними

функціями. Він також підтримує інтеграцію з базами даних, що дозволяє зберігати та обробляти фінансові дані користувача. Обробка HTTP-запитів і відповідей – із включенням операцій POST, GET, PUT, DELETE тощо; Створення RESTful API — інтерфейсу для стандартизованої взаємодії між клієнтом і сервером; Зв’язування проміжного програмного забезпечення – це програмний код, який знаходиться між виконанням таких завдань, як перевірка автентифікації (через JWT), обробка помилок, журналювання тощо.

Використання Node.js у поєднанні з Express.js забезпечує вебзастосунку надійну, масштабовану та ефективну серверну інфраструктуру. Це дозволяє підтримувати всі необхідні функції бекенду, такі як реєстрація користувачів, обробка транзакцій, додавання категорій тощо.

База даних.

Для зберігання, обробки та аналізу даних у вебзастосунку застосовується Microsoft SQL Server[7] – потужна реляційна система управління базами даних, що забезпечує високу продуктивність, надійність і масштабованість та підтримує складні SQL-запити, тригери, процедурні механізми й індекси.

Базу даних було створено та керовано в SQL Server Management Studio (SSMS), офіційному середовищі для керування SQL Server. SSMS надає графічний інтерфейс, за допомогою якого можна розробляти, тестувати та контролювати запити SQL, окрім керування структурою бази даних та її безпекою.

Дизайн структури бази даних нормалізовано до третьої нормальної форми (3NF). Це дозволяє:

- уникнути надмірного дублювання даних;
- зберегти логічну цілісність зв'язків між таблицями;
- увімкнути розширюваність моделі — тобто можливість додавати нові таблиці або атрибути без зміни базової логіки.

Зв’язки між таблицями створюються за допомогою зовнішніх ключів, що забезпечує логічну послідовність і дозволяє формувати складні запити для

аналітичної інформації, яка використовується для візуалізації в інтерфейсі користувача.

Аутентифікація і безпека.

Аутентифікація та забезпечення безпеки є ключовими складовими у розробці вебзастосунків, що дозволяють захистити дані користувачів та забезпечити правильне функціонування системи.

- JSON Web Tokens[18] (JWT) використовуються для управління авторизацією. Після успішного входу сервер створює токен, підписаний секретним ключем, і відправляє його клієнту. Клієнт додає цей токен до заголовку `Authorization` у кожному запиті до сервера. Такий підхід дозволяє здійснювати перевірку легітимності запиту та ідентифікувати користувача без необхідності зберігати стан сесії на сервері.
- bcrypt використовується для надійного хешування паролів. Перед тим як зберегти пароль в базі даних, він проходить хешування. Це забезпечує додатковий захист, знижуючи ризик зворотного відновлення паролів навіть у разі компрометації бази даних. Таким чином, метод bcrypt є одним із найнадійніших для автентифікації.

Під системою перевірки доступу та обробки помилок передбачено серверне рішення, яке перевіряє кожен запит до захищених маршрутів. Перевіряється коректність JWT-токена та рівень доступу користувача.

Для забезпечення безпеки та зручності розробки також використовуються додатково наступне:

- axios[19] є клієнтською бібліотекою, яка спрощує виконання HTTP-запитів. Вона обробляє помилки, автоматично додає JWT у заголовки запитів та забезпечує асинхронну взаємодію з API, що робить процес оновлення даних більш оперативним і гнучким.
- dotenv[20] використовується для управління конфіденційною інформацією через змінні середовища. У файлах зберігаються важливі дані,

наприклад параметри підключення до бази даних чи порти програми. Це дозволяє мінімізувати ризик витоку критичної інформації у вихідному коді.

- cors[21] виступає серверним проміжним шаром для контролю міждоменного доступу. Він допомагає встановлювати правила для запитів із різних доменів або портів, що особливо актуально, коли фронтенд і бекенд розміщені на окремих платформах. Через нього налаштовуються дозволені джерела, методи і заголовки для запитів.

У комплексі ці технології забезпечують високий рівень безпеки, спрощують розробку та сприяють ефективній взаємодії між клієнтською та серверною частинами вебзастосунку.

Середовище розробки.

Для розробки, тестування та підтримки вебзастосунку було залучено ряд сучасних інструментів, що сприяють оптимізації процесу створення програмного забезпечення, полегшенню налагодження та ефективній організації командної роботи.

Visual Studio Code (VS Code)[22] використовується як основне середовище розробки завдяки своїй легкості, гнучкості та широким можливостям розширення. Ця інтегрована платформа стала одним із найпопулярніших виборів серед веброзробників завдяки ряду переваг:

- підсвічування синтаксису для різних мов програмування, включаючи JavaScript, JSX, CSS, SQL;
- інтеграція з Git для управління версіями;
- підтримка широкого спектра розширень;
- автозаповнення коду та інтелектуальні підказки;
- засоби налагодження як клієнтського, так і серверного коду;
- інтегрований термінал для запуску локальних серверів, виконання команд Node.js або роботи з базою даних.

SQL Server Management Studio (SSMS)[23] використовується для управління та адміністрування баз даних на платформі Microsoft SQL Server. SSMS застосовувався для:

- створення та модифікації таблиць бази даних;
- налагодження зв’язків між таблицями за допомогою зовнішніх ключів;
- написання й виконання SQL-запитів;
- управління користувачами та їхніми правами доступу;
- резервного копіювання і відновлення баз даних;
- відлагодження запитів і аналізу їхньої продуктивності.

Інтерфейс SSMS забезпечує зручний графічний інструментарій для роботи з об’єктами бази даних, що значно спрощує проєктування й підтримку структури даних під час розробки інформаційної системи.

Postman[24] використовується як зручний графічний інструмент для тестування RESTful API. У процесі розробки він допомагав вирішувати завдання, що пов’язані з надсиланням HTTP-запитів (GET, POST, PUT, DELETE) до серверного API; перевірка механізмів авторизації через JWT-токени; емуляцією різних сценаріїв комунікації з бекендом без запуску інтерфейсу клієнта.

Git[25] виконує роль системи контролю версій, що дозволяє:

- зберігати історію змін у коді;
- створювати відокремлені гілки для розробки нових функцій чи виправлення помилок;
- зливати зміни між гілками;
- слідкувати за змінами та відновлювати попередній стан у разі помилок.

GitHub[26] є хмарною платформою для зберігання репозиторіїв Git, яка надає: централізоване сховище для зберігання коду, відкритий або закритий доступ до проєктів, можливість створювати pull-запити для обговорення змін

перед їх інтеграцією в основну гілку, перегляд історії змін, коментарів і запуску автоматизованих перевірок (за потреби CI/CD).

Усе це разом формує комплексне середовище розробки, яке охоплює всі етапи створення вебзастосунку – від написання коду та проєктування баз даних до тестування API і впровадження змін. Використання таких інструментів сприяє підвищенню якості програмного продукту та прискорює розробку.

3.2 Архітектурне моделювання проєктування ПЗ

Абстракції в інформаційній системі.

Створюючи інформаційну систему для управління особистими фінансами, необхідно окреслити основні абстракції – сутності, які представляють ключові концепти предметної області. Кожна з цих абстракцій володіє набором характеристик (атрибутів) і виконує певні обов’язки (функції). Це дозволяє сформувати зрозумілу та модульну структуру системи.

Абстракція: Транзакція

Властивості:

- Назва – назва конкретної транзакції, що дозволяє відрізняти її від інших.
- Категорія – вказує тип транзакції (дохід чи витрата) і належність до певної категорії (їжа, транспорт, зарплата тощо).
- Сума – значення, яке позначає обсяг коштів, що витратились або отримались.
- Опис – текстовий опис транзакції.
- Рахунок – посилання на конкретний рахунок користувача, з якого чи на який відбулася транзакція.

Обов’язки:

- Зберігати всю необхідну інформацію про конкретну транзакцію.
- Забезпечувати функціонал створення, редагування та видалення транзакцій.
- Підтримувати збережені валюти і створені рахунки.

Абстракція: Рахунок

Властивості:

- Назва – назва рахунку, зазвичай згідно типу («Картка», «Готівка»).
- Баланс – поточний залишок коштів на рахунку.
- Валюта – валюта, у якій ведеться облік рахунку.

Обов’язки:

- Зберігати інформацію про стан рахунку.
- Оновлювати баланс при здійсненні транзакцій.
- Забезпечувати вибір рахунку під час додавання транзакції.

Абстракція: Тип рахунку

Властивості:

- Назва – визначає назву типу рахунку («Картка», «Готівка»).
- Тип рахунку – тип рахунку згідно встановлених.

Обов’язки:

- Зберігати інформацію про класифікацію типів рахунків.

Абстракція: Категорія

Властивості:

- Назва – назва категорії, яка вказує на приналежність до класифікації.
- Тип транзакції – вказує, чи категорія призначена для доходів чи витрат.
- Іконка – призначена для візуальної демонстрації приналежності до певної категорії.

Обов’язки:

- Зберігати інформацію про категорії.
- Керувати категоріями.
- Дозволяти перегрупування та управління всіма категоріями.

Абстракція: Власник фінансів

Властивості:

- Ім’я – ім’я користувача.

- Пошта – контактна інформація для ідентифікації, заміняє логін.

Обов’язки:

- Керувати транзакціями, рахунками, категоріями та операціями.
- Переглядати фінансові баланси.

Абстракція: Операції

Властивості:

- Назва – назва-визначення операції
- Тип операції – вказує текстову дію (подія, запис, завдання).
- Опис – опис-вказівки текстової операції.
- Дата — часова позначка виконання операції.

Обов’язки:

- Зберігати інформацію про виконані операції.
- Додавати, змінювати і видаляти операції.

Повний перелік абстракцій наведено в Додатку В (сторінка 1).

Діаграма класів.

Діаграма класів є важливим елементом об'єктно-орієнтованого проєктування, який відображає структуру системи через класи, їх властивості, методи та взаємозв'язки між ними. У рамках розробленої системи для фінансового менеджменту вона виконує ключову функцію моделювання логіки предметної області, забезпечуючи формалізований опис взаємодії основних об'єктів[27].

Ця діаграма дозволяє наочно відобразити класи, що представляють головні абстракції системи. Вона демонструє не лише внутрішню структуру кожного класу (включаючи властивості та методи), але й типи зв'язків між ними.

Такий підхід полегшує проєктування програмної логіки на основі чітко сформульованої моделі. Це сприяє легшому масштабуванню системи, інтеграції нових функцій і підвищенню стабільності підтримки поточного функціоналу.

Діаграма класів представлена в Додатку В (сторінка 2).

Класи діаграми:

Клас «Транзакція»

- Властивості: назва, категорія, сума, опис, рахунок
- Обов’язки: зберігання даних про транзакцію, взаємодія з рахунками і валютами і безпосередньо керування транзакціями.

Клас «Рахунок»

- Властивості: назва, баланс, валюта
- Обов’язки: зберігання даних про рахунок, оновлення балансу при змінах, зв’язок з транзакціями, вибір типу рахунку

Клас «Тип рахунку»

- Властивості: назва, тип валюти
- Обов’язки: визначення і зберігання типу рахунку

Клас «Категорія»

- Властивості: назва, категорія, тип транзакції, іконка
- Обов’язки: зберігання даних про категорію, управління категоріями, перегляд наявних категорій

Клас «Власник фінансів»

- Властивості: ім’я, пошта
- Обов’язки: управління власними транзакціями і операціями, перегляд статистики.

Клас «Операції»

- Властивості: назва, тип операції, опис, дата
- Обов’язки: зберігання даних про операції, управління операціями.

Зв’язки між класами:

- Транзакція – Рахунок: один рахунок містить багато транзакцій (1...*)
- Рахунок – Тип рахунку: один до одного (1...1)
- Транзакція – Категорія: кожна транзакція належить до однієї категорії (1...1)
- Власник фінансів – Рахунок: один власник має багато рахунків (1...*)

- Власник фінансів – Категорія: один власник керує багатьма категоріями (1...*)
- Власник фінансів – Операції: один власник створює багато операцій (1...*)
- Власник фінансів – Транзакція: один власник створює багато транзакцій (1…*)

Побудована діаграма класів слугує основою для реалізації логіки програмної системи, що відповідає заданим концептуальним абстракціям. Завдяки такій архітектурі досягається ефективне управління фінансовими процесами, забезпечується гнучкість у застосуванні та створюються умови для поступового вдосконалення й розширення функціональних можливостей у перспективі.

Діаграма пакетів.

Діаграма пакетів використовується для зображення загальної архітектури програмного забезпечення, що демонструє, як окремі частини системи, що логічно структуровані у вигляді пакетів. Вона наочно відображає модульність проєкту, показуючи взаємодію компонентів користувацького інтерфейсу й адміністратора через чітко визначені залежності[27].

Діаграма пакетів пропонує візуальне представлення архітектури системи, акцентуючи увагу на розподілі відповідальностей, рівнях доступу, можливостях для масштабування і спрощення підтримки. Кожен пакет об’єднує класи чи компоненти із суміжною функціональністю, що сприяє кращій ізоляції функцій і полегшує управління системою.

Діаграма пакетів представлена в Додатку В (сторінка 3).

Опис пакетів діаграми:

- UserInterface (Інтерфейс користувача)

Цей пакет є ключовою частиною взаємодії кінцевого користувача із системою. Він включає кілька функціональних компонентів, які забезпечують реалізацію основних можливостей системи.

- AdminInterface (Інтерфейс адміністратора)

Цей пакет є ключовим для адміністратора. Він включає в себе два компонента, які забезпечують основі можливості адміністратора.

Залежності між пакетами.

- **Взаємодія у межах UserInterface:**

Transaction і Settings – використання налаштувань для персоналізації і використання транзакцій.

Categories і Settings – використання для управління назвами, типом і іконками категорій.

Accounts і Settings – використання для управління рахунками і їх валютами.

Operation і Settings – використання налаштувань для управління операціями (запис, завдання і подія).

- **Взаємозв'язок між UserInterface та AdminInterface:**

Settings і Configuring the system – налаштування базових даних для загального користування.

UserInterface і Configuring user – встановлення обмежень доступу до даних.

У представленій архітектурі системи поєднуються гнучкість, безпека та легкість у розширенні й підтримці. Модульна структура спрощує процес розробки, оновлення та масштабування. Інтеграція всіх компонентів через єдиний підхід забезпечує узгоджений та гармонійний робочий процес.

Прості кооперації.

Під час впровадження інформаційної системи управління фінансами головним завданням стає моделювання взаємодії її компонентів для забезпечення необхідного рівня функціональності. У системі кооперації описують порядок і логіку співпраці між елементами при виконанні дій користувачів. У межах проєкту були визначені ключові сценарії кооперацій, спрямовані на персоналізацію, облік, управління та аналіз фінансових даних.

Кооперація «Персоналізація користувача»:

Ця кооперація спрямована на налаштування роботи системи під індивідуальні потреби кожного користувача. Система дозволяє налаштовувати

облікові записи шляхом створення та управління рахунками, вибору їх типів і реєстрації операцій, пов’язаних із фінансовою активністю. Це сприяє зручності користування та адаптації інтерфейсу до конкретного користувача.

Прості кооперації представлено в Додатку В (сторінка 4 і 5).

Ключові компоненти:

- Рахунок (Account): Користувач може створювати декілька рахунків для ефективнішого управління фінансами, наприклад, «Гаманець», «Зарплатна карта», «Кредит» тощо.

Приклад: Створення рахунку «Картка ПриватБанк» із валютою «UAH» та початковим балансом у 5000 грн.

- Тип рахунку (AccountType): Рахунки класифікуються за категоріями, такими як «Готівка», «Банківська карта».

Приклад: Для рахунку «Картка ПриватБанк» при створені встановлено тип «Банківська карта».

- Власник фінансів (OwnersFinances): Основна сутність у взаємодії з іншими класами системи. Користувач має доступ до базових рахунків і категорій і повний доступ над власними рахунками, операціями, категоріями, транзакціями та налаштуваннями системи.

Приклад: Користувач додає три рахунки, налаштовує категорії доходів і витрат, а також створює різні операції і транзакції.

- Операції (Operation): Використовуються для фіксації подій, таких як регулярні витрати, задачі чи нагадування, завдань, таких як зробити домашню роботу, записів, таких як список справ на день.

Приклад: Запланована зустріч як подія на перше число наступного місяця.

Взаємодія компонентів:

- Тип рахунку – Рахунок

Тип зв’язку: Композиція

Кожен рахунок прив’язаний до певного типу, який визначає його функціональні особливості. Тип рахунку є невіддільною складовою самого рахунку, адже його існування без визначеного типу неможливе.

- Власник фінансів – Рахунок

Тип зв’язку: Агрегація

Рахунки створюються та керуються користувачем залежно від його потреб. Вони можуть бути незалежними від конкретного користувача в межах загальної системи, але логічно пов’язані з ним як із власником.

- Власник фінансів – Категорія

Тип зв’язку: Агрегація

Категорію створюються і змінюються користувачем залежно від його способу життя. Вони логічно належать користувачу, однак на технічному рівні можуть існувати як спільні елементи структури.

- Власник фінансів – Операція

Тип зв’язку: Композиція

Всі операції контролюються користувачем, який володіє відповідними рахунками. Кожна операція нерозривно пов’язана з діяльністю конкретного користувача.

Кооперація «Керування фінансами»:

Ця кооперація є ключовою для всієї системи, оскільки виконує функції обліку й аналізу фінансових подій. Вона дозволяє користувачам додавати транзакції, класифікувати їх за категоріями, переглядати баланси і статистику.

Ключові компоненти:

- Рахунок (Account): Містить дані про загальний фінансовий стан користувача. Баланс рахунку має автоматично оновлюватись під час додавання нової транзакції.

Приклад: Якщо створити транзакцію «Комунальний платіж» для рахунку «Картка ПриватБанк» із сумою 1500 грн, баланс рахунку коригується одразу після списання коштів.

- Транзакція (Transaction): Основний елемент обліку, що включає всі ключові дані, такі як категорія, сума, тип (дохід чи витрата) і опис.

Приклад: Транзакція «Покупка продуктів» має категорію "Продукти", суму 750 грн і прив’язаний рахунок "Готівка".

- Категорія (Category): Дозволяє впорядковувати транзакції. Категорії можуть бути загальними (доступними всім користувачам) або персональними (створеними конкретним користувачем).

Приклад: Для структуризації витрат користувач може створити категорії на зразок «Зарплата», «Їжа» чи «Транспорт».

- Власник фінансів (OwnersFinances): Користувач є головним суб'єктом у системі, маючи доступ до своєї фінансової інформації.

Приклад: Користувач створює категорію «Освіта», додає рахунок «Готівка» та реєструє транзакцію як витрату на курси.

Взаємодія компонентів:

- Транзакція – Рахунок

Тип зв'язку: Композиція

Транзакція не може існувати без рахунку. Вона невід’ємно пов’язана з ним та впливає на його баланс.

- Транзакція – Категорія

Тип зв'язку: Композиція

Категорія є обов’язковою частиною транзакції, оскільки визначає її зміст та мету.

- Власник фінансів – Категорія

Тип зв'язку: Агрегація

Користувач створює й редагує категорії. Вони логічно належать йому, проте можуть існувати окремо як загальні чи персоналізовані.

- Власник фінансів – Рахунок

Тип зв'язку: Агрегація

Користувач управляє своїми рахунками та володіє ними. Незважаючи на те, що рахунки створені користувачем, вони можуть зберігатися технічно незалежно від нього.

Спільні компоненти кооперацій:

Рахунок виступає як центральний елемент персоналізації й джерело або одержувач коштів.

Користувач є ініціатором усіх дій, володіє об’єктами системи (рахунками, транзакціями, категоріями, операціями).

Діаграма компонентів.

Діаграма компонентів представляє логічну структуру програмного забезпечення у формі незалежних функціональних модулів — компонентів, які взаємодіють між собою та з базами даних. Основним завданням такого підходу є візуалізація зв’язків між модулями для користувачів, адміністративного інтерфейсу та спеціалізованих підсистем управління даними. Це сприяє зрозумілому розподіленню функціональності, спрощенню технічного обслуговування, а також покращенню масштабованості системи та її захищеності[27].

Діаграма компонентів представлено в Додатку В (сторінка 6).

Компоненти головного (користувацького) інтерфейсу:

Компонент Транзакції:

1) виконує створення, перегляд, редагування та видалення транзакцій;
2) отримує дані з модуля Transaction BD та відображає їх у зручному вигляді;
3) дозволяє фільтрувати інформацію за такими параметрами, як тип, сума, категорія та дата;
4) інтегрується із категоріями і рахунками для забезпечення узгодженої роботи.

Компонент Налаштування:

1) забезпечує управління персональними параметрами (наприклад, мова платформи, тема інтерфейсу);

2) впливає на функціонування інших модулів, наприклад вибір валюти для рахунку;

3) зберігає параметри реєстрації і авторизації в Users BD через централізовану логіку.

Компонент Авторизація/реєстрація:

1) здійснює вхід користувачем для використання;

2) зберігає дані реєстрації і авторизації.

Компонент Операції:

1) відповідальний за роботу з різними записами, подіями та завданнями;

2) отримує дані з Operations BD для обробки операцій;

Компоненти адміністративного інтерфейсу:

Компонент Шаблони категорій:

1) відповідає за управління загальними шаблонами фінансових категорій;

2) можливість створення загальнодоступних шаблонів;

3) дані зберігаються в модулі Categories BD.

Компонент Шаблони рахунків:

1) дає змогу створювати різні типи, які будуть загальнодоступними;

2) взаємодіє з Accounts BD для збереження відповідних параметрів.

Компонент Адміністрування:

1) основний модуль управління користувачами;

2) виконує видалення облікових записів;

3) має доступ із Users BD для обробки даних користувачів.

Підсистеми бази даних

Система баз даних складається з окремих підсистем, кожна з яких має чіткі завдання, пов'язані з управлінням даними. Компоненти системи працюють у тісній взаємодії через запити, що забезпечує ефективну обробку інформації та збереження її цілісності.

Основні компоненти бази даних:

- Transaction BD: містить інформацію про всі фінансові транзакції.
- Categories BD: підтримує загальні та персоналізовані категорії.
- Operations BD: відображає всі дії користувачів, пов'язані з операціями (запити, події, завдання).
- Accounts BD: зберігає інформацію про рахунки: назва, тип, баланс, валюта тощо.
- Users BD: виступає основним сховищем даних про користувачів і забезпечує автентифікацію та авторизацію.

Взаємодія між елементами системи організована через користувацький та адміністративний інтерфейси, які підключаються до бази даних за допомогою серверної логіки та REST API-запитів. Адміністративний інтерфейс надає повний функціонал для управління інформацією (створення, редагування, видалення) і доступний виключно авторизованим адміністраторам із належними правами.

4 ВПРОВАДЖЕННЯ ТА ПЕРЕВІРКА РОБОТИ ІНФОРМАЦІЙНОЇ СИСТЕМИ

4.1 Тестування функціональності системи

Після завершення розробки головних частин вебзастосунку було проведено тестування функціональних можливостей REST API – інтерфейсу для забезпечення взаємодії між клієнтом і сервером за допомогою HTTP-запитів. Завдяки REST API здійснюється управління обліковими записами, транзакціями, категоріями, подіями та іншими системними ресурсами в режимі реального часу.

Для тестування використовувався інструмент Postman[24]. Він забезпечує зручність перевірки різних типів запитів (POST, GET, PUT, DELETE), роботу з токенами авторизації, аналіз відповідей сервера, перевірку статус-кодів і обробку помилок[16].

POST використовується для додавання нових елементів до бази даних сервера.

Основні характеристики:

- Обов’язкове тіло запиту у форматі JSON
- У відповідь сервер зазвичай надсилає статус 201 Created із новоствореним об'єктом

Далі будуть наведені приклади, які є тестами REST API, що виконувались для оцінки роботи основних маршрутів і перевірки функціональності системи.

Приклади (тести) використання POST-запитів:

- Реєстрація користувача: POST /api/auth/register

Виконання тесту створення користувача представлено на рис. 9.

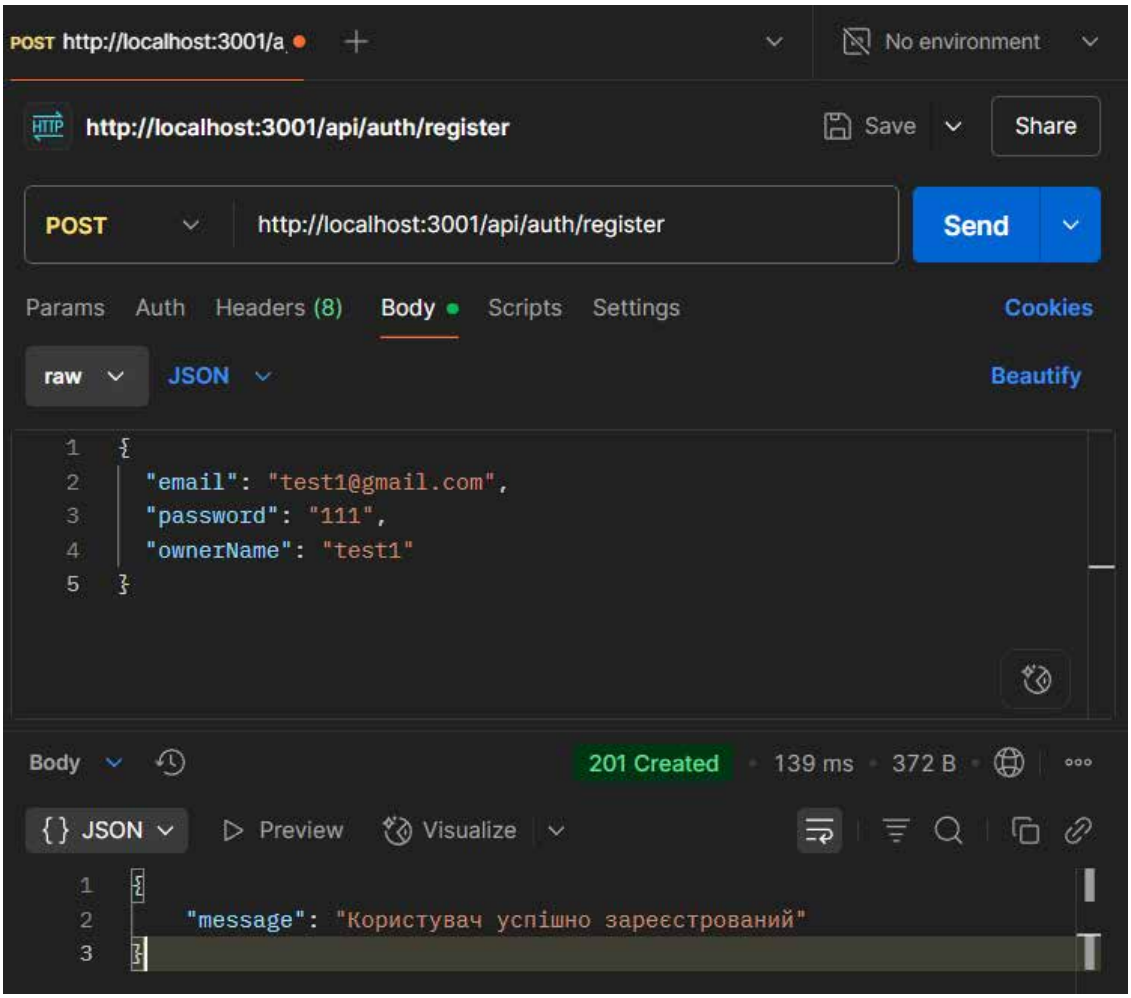


Рис. 9 Тест реєстрації користувача

- Створення категорії: POST /api/categories

Щоб виконати POST-запит на створення нової категорії, необхідно передати параметри відповідного користувача, оскільки доступ до ресурсу захищено. Для цього в заголовку запиту (Header) слід вказати токен авторизації у форматі Authorization: Bearer «jwt_token». Для цього, необхідно виконати POST-запит на отримання токена користувача.

Виконання тесту отримання користувача представлено на рис. 10.

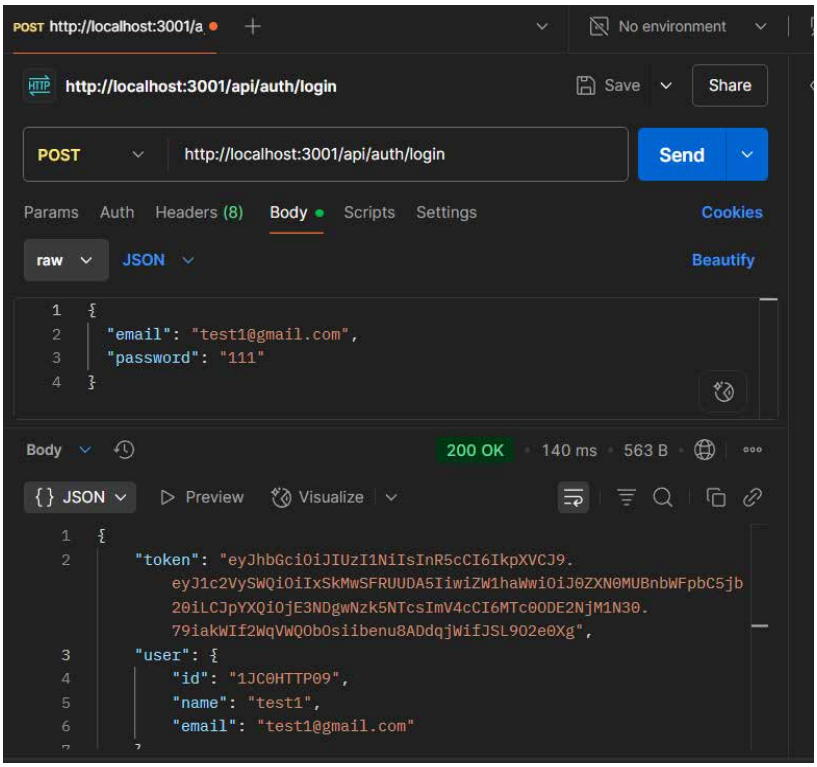


Рис. 10 Тест авторизації користувача користувача

Тепер виконується POST-запит на створення категорії.

Виконання тесту створення категорії представлено на рис. 11.

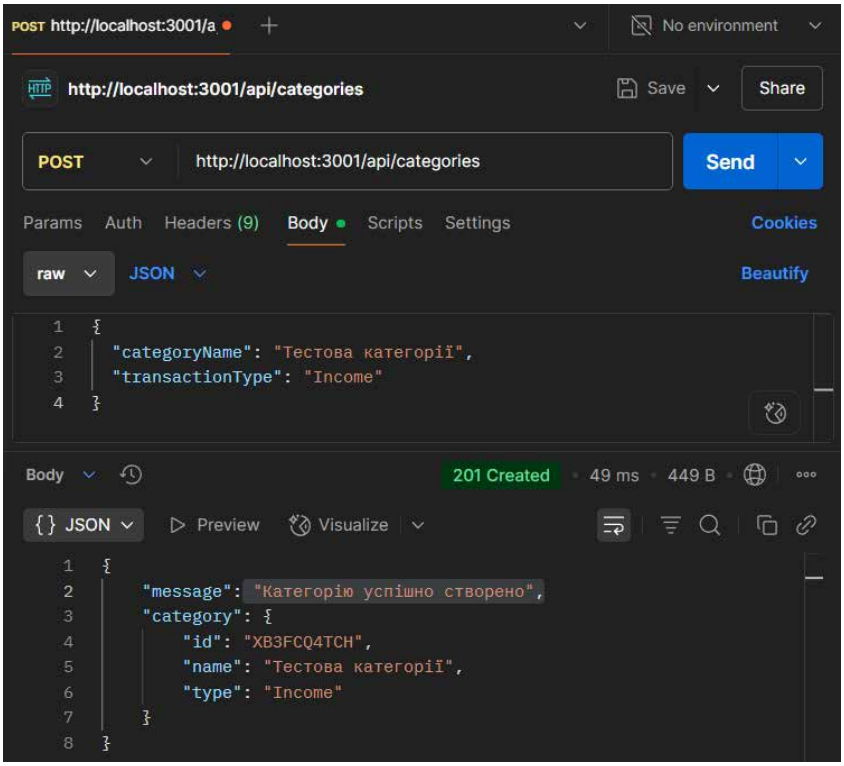


Рис. 11 Тест створення категоорії користувача

GET застосовується для запиту інформації з сервера без будь-яких змін у самих даних.

Основні характеристики:

- Не впливає на стан сервера
- Параметри запиту передаються через URL
- Не потребує тіла запиту

Далі будуть наведені приклади, які є тестами REST API, що виконувались для оцінки роботи основних маршрутів і перевірки функціональності системи як це було з POST-запитами.

Приклади (тести) використання GET-запитів:

- Отримання списку рахунків: GET /api/accounts

Тут так само для виконати GET-запиту на отримання списків, необхідно вказати Authorization: Bearer «jwt_token» в заголовку запиту (Header), що вказує на певного користувача, список якого треба отримати.

Виконання тесту отримання списку рахунків представлено на рис. 12.

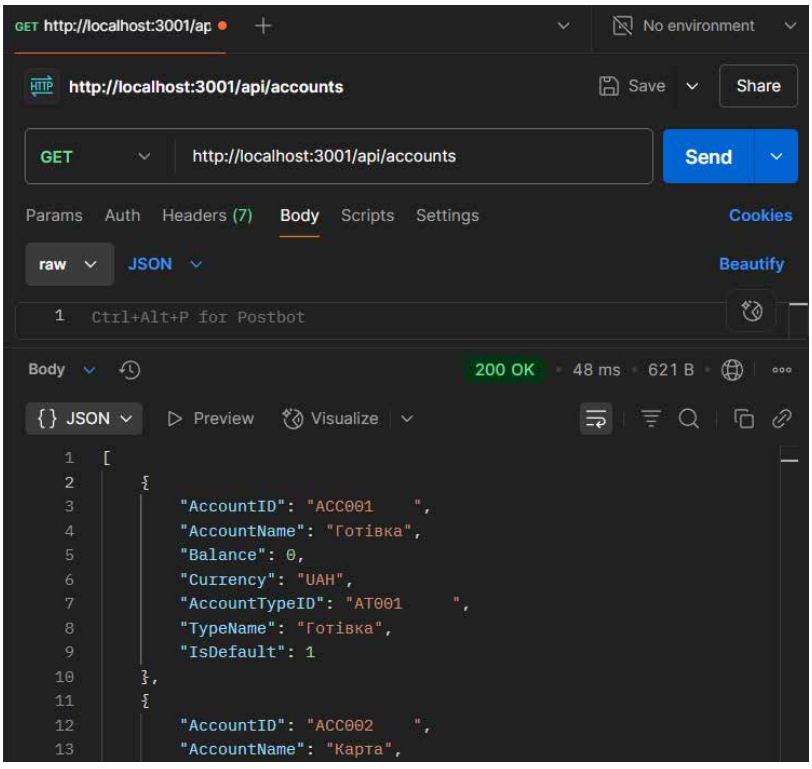


Рис. 12 Тест отримання списку рахунків

- Отримання всіх категорій: GET /api/categories

Виконання тесту отримання всіх категорій представлено на рис. 13.

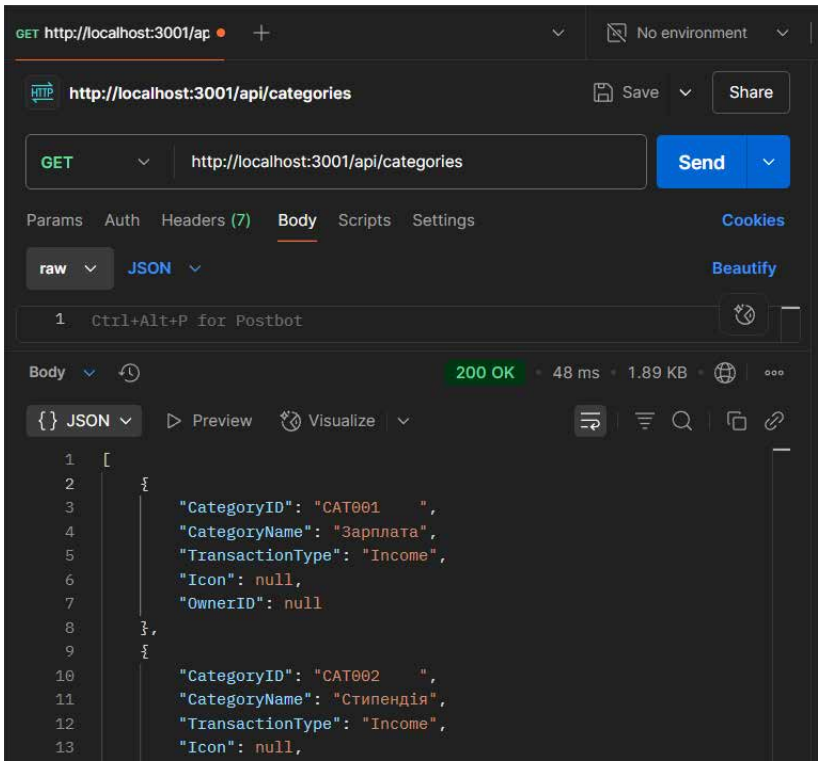


Рис. 13 Тест отримання списку категорій

PUT застосовується для повної заміни ресурсу за його ідентифікатором.

Основні характеристики:

- Вимагає повного тіла запиту з усіма необхідними даними.
- Якщо деякі поля пропущені, їхні значення можуть бути обнулені.
- Відповідь зазвичай має статус 200 OK і містить оновлений ресурс.

Далі будуть наведені приклади, які є тестами REST API, що виконувались для оцінки роботи основних маршрутів і перевірки функціональності системи як це було з запитами до цього.

Приклади (тести) використання PUT -запитів:

- Редагування наявного рахунку: PUT /api/accounts/

Також вказується, як і на GET-запити, Authorization: Bearer «jwt_token» в заголовку запиту (Header), що вказує на певного користувача, рахунок якого треба оновити.

Виконання тесту змінення рахунку представлено на рис. 14.

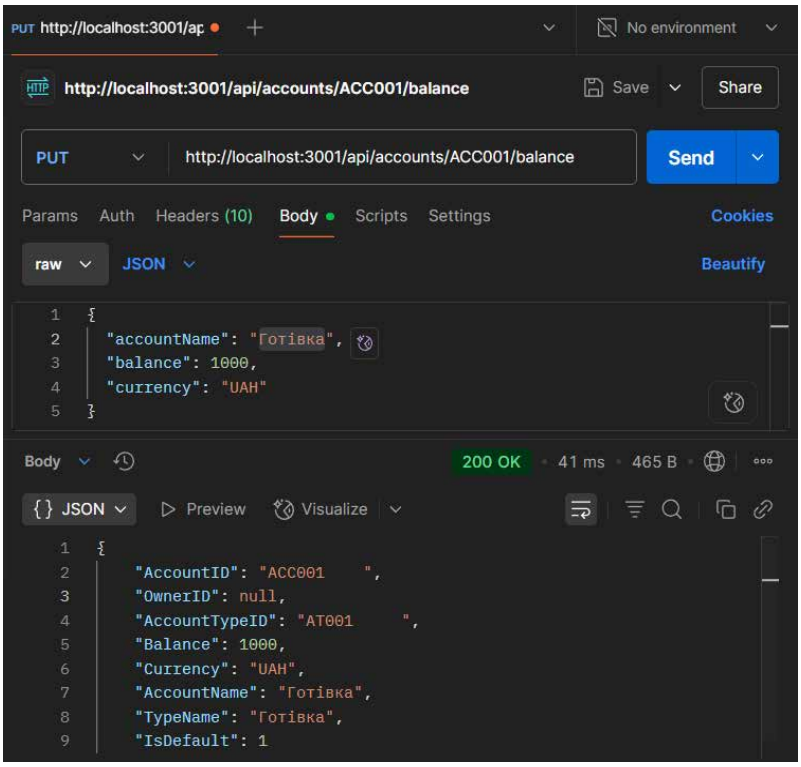


Рис. 14 Тест редагування рахунку

- Оновлення категорії: PUT /api/categories/

Виконання тесту змінення категорії представлено на рис. 15.

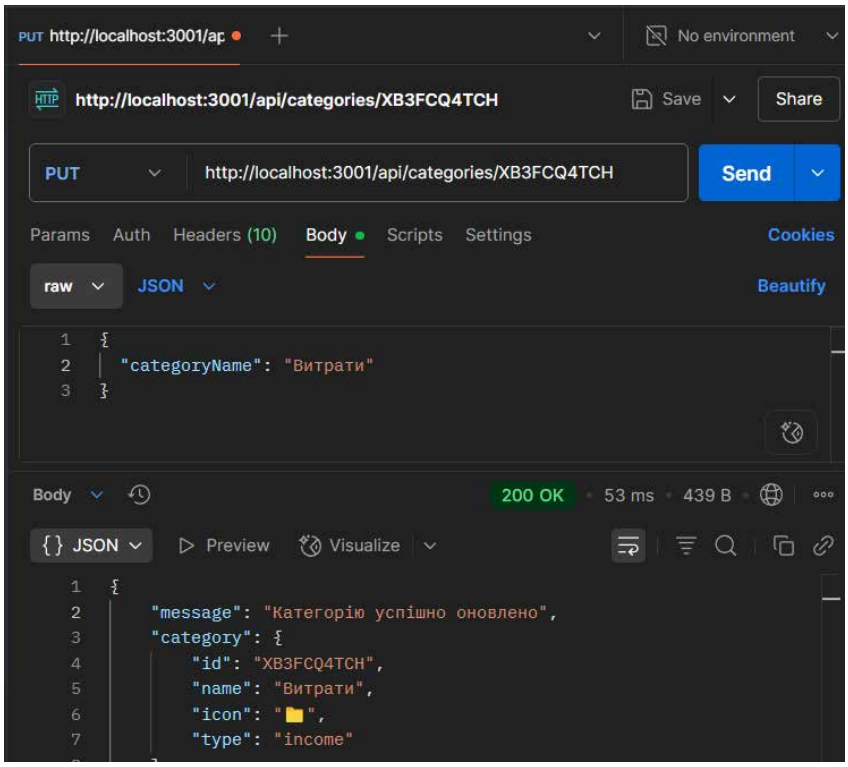


Рис. 15 Тест редагування категорії

DELETE використовується для видалення інформації із сервера.

Основні характеристики:

- Не потребує тіла запиту
- Повертає статус 200 OK або 204 No Content як підтвердження успішного виконання операції

Далі будуть наведені приклади, які є тестами REST API, що виконувались для оцінки роботи основних маршрутів і перевірки функціональності системи як це було з запитами до цього.

Приклади (тести) використання DELETE-запитів:

- Видалення рахунку: DELETE /api/accounts/

Також вказується, як і на інших запитах, Authorization: Bearer «jwt_token» в заголовку запиту (Header), що вказує на певного користувача, рахунок якого треба оновити.

Виконання тесту видалення рахунку представлено на рис. 16.

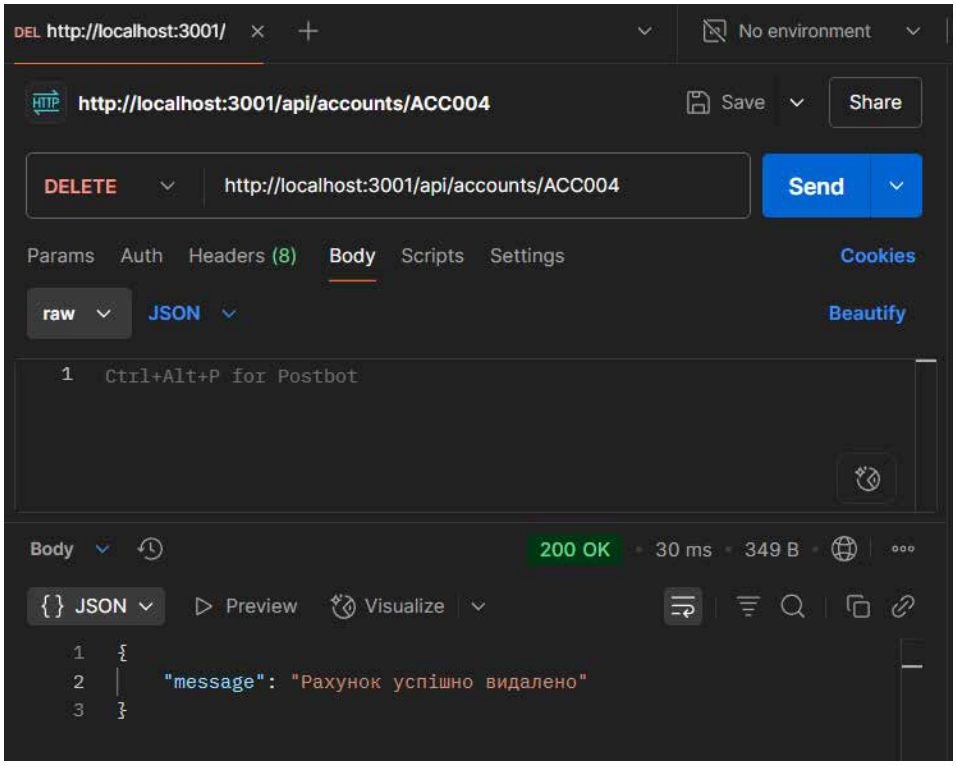


Рис. 16 Тест видалення рахунку

- Видалення категорії: DELETE /api/categories/

Виконання тесту видалення категорії представлено на рис. 17.

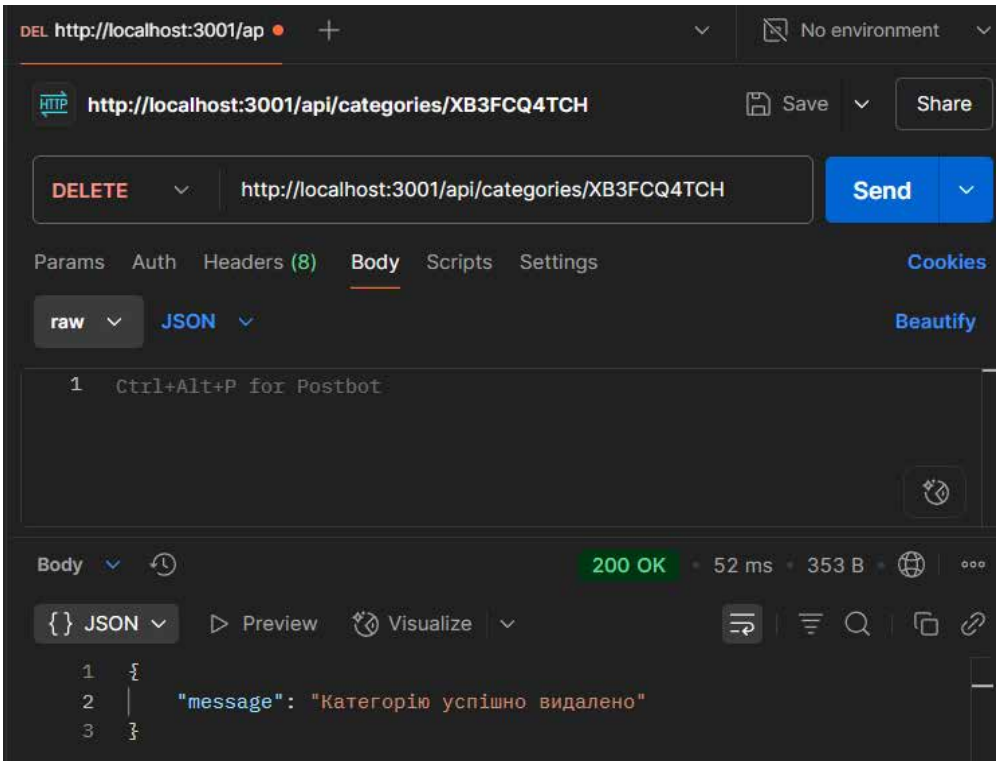


Рис. 17 Тест видалення категорії

Використання методів POST, GET, PUT і DELETE разом із правильним налаштуванням заголовків запиту, таких як Authorization для автентифікації, створює основу для ефективної та безпечної взаємодії між клієнтом і сервером.

HTTP-методи надають структурований та ефективний механізм взаємодії між клієнтом і сервером. Вони дозволяють здійснювати ключові операції з ресурсами, охоплюючи роботу з користувачами, рахунками, категоріями й іншими об'єктами. Це робить REST API універсальним інструментом для розробки масштабованих і гнучких вебсервісів.

4.2 Апаратні та технічні засоби

Для ефективної роботи вебзастосунку використовується архітектура типу «клієнт-сервер», яка забезпечує чіткий розподіл завдань між різними елементами системи. Діаграма розгортання показує як логічні компоненти застосунку розміщені на фізичних пристроях. Основними складовими є вузли — це пристрої або середовища виконання, що містять програмні артефакти[27]. Взаємини між вузлами підтримуються через мережеві протоколи, такі як HTTP/HTTPS або SQL Server.

Діаграма розгортання.

Вузли та компоненти діаграми:

Пристрій (PC, mobile)

Цей вузол відповідає за взаємодію кінцевого користувача із системою через вебінтерфейс.

- Артефакт: веббраузер, що завантажує й відображає клієнтську частину застосунку.
- Функції: забезпечення прийому HTML-сторінок, виконання JavaScript-коду, а також взаємодія з елементами користувацького інтерфейсу.
- З’єднання: взаємодія з вебсервером здійснюється через протокол HTTP/HTTPS.

Вебсервер (Web-server)

Цей вузол призначений для реалізації логіки застосунку, опрацюванням запитів і формуванням відповідей.

- Артефакти:
  - HTML, CSS, JS — клієнтська частина, розроблена за допомогою React + Vite.
  - Node.js — серверна частина, реалізована з використанням фреймворка Express.
- Функції: підтримка REST-запитів, автентифікація користувачів, перевірка даних, обмін інформацією з базою даних.
- З’єднання: обмін даними з клієнтом через HTTP/HTTPS та взаємодія з базою даних через SQL Server протокол.

Сервер бази даних (DB)

Окремий стрижневий вузол, що виконує роль централізованого сховища даних.

- Артефакт: реляційна база даних Microsoft SQL Server (SQL Server DB).

- Функції: обробка та збереження інформації про транзакції, рахунки, категорії, події й користувачів системи.
- З’єднання: прийняття SQL-запитів від серверної частини Node.js із подальшим поверненням оброблених результатів.

Діаграма розгортання представлена у Додатку В (сторінка 6).

4.3 Вимоги до апаратного та програмного забезпечення

Для забезпечення ефективної роботи вебзастосунку необхідно враховувати як апаратні, так і програмні вимоги, що визначають мінімальні характеристики для запуску, стабільності та масштабованості системи.

Апаратні вимоги:

- Операційна система: Windows 10 або 11 — надійна платформа з широкою підтримкою серверного програмного забезпечення.
- Процесор: 4-ядерний або продуктивніший, наприклад, Intel Core i5 чи AMD Ryzen 5.
- Оперативна пам’яті: Від 8 ГБ — достатньо для одночасної роботи клієнтської частини, серверу та СУБД.
- Накопичувач: SSD від 100 ГБ — забезпечує швидкий доступ до логів, баз даних та резервних копій.
- Мережевий інтерфейс: Стабільне підключення з підтримкою протоколу HTTPS (TLS).

Програмні вимоги:

- Серверна частина (Backend):
- Node.js: середовище виконання JavaScript на сервері.
- Express.js: мікрофреймворк для створення REST API.
- npm: менеджер пакунків для встановлення залежностей (dotenv, mssql, cors тощо).
- Безпека: `bcrypt` для хешування паролів, `helmet` для захисту заголовків.

Клієнтська частина (Frontend):

- React.js: бібліотека для створення інтерфейсу користувача.
- Vite: швидкий інструмент для збірки й розгортання.

База даних:

- Microsoft SQL Server 2019 або новіша версія.

Вимоги до системи з точки зору користувача

Система має бути інтуїтивно зрозумілою, легкою у використанні та не вимагати технічних знань.

Функціональні вимоги:

- доступ через браузер без додаткового встановлення ПЗ;
- адаптивний дизайн для різних пристроїв: ПК, планшетів і смартфонів;
- авторизація з перевіркою доступу;
- простий інтерфейс навігації та зручне меню.

Нефункціональні вимоги:

- надійна робота навіть при високих навантаженнях;
- мінімальний час відгуку (до 1 секунди);
- безпечне збереження даних;
- забезпечення конфіденційності інформації користувача.

Вимоги до інтерфейсу:

- мінімалістичний і зрозумілий дизайн;
- висока контрастність і хороша читабельність;
- швидкість завантаження сторінок без затримок.

4.4 Розгортання проєкту

У проєкті створено вебзастосунок, що включає клієнтську частину (інтерфейс користувача) та серверну частину (логіка обробки запитів і доступу до бази даних). Для забезпечення доступу до системи через Інтернет проєкт розгортається на хмарній платформі. Розгортання означає публікацію

програмного забезпечення на зовнішньому сервері, щоб користувачі могли користуватись розробленим функціоналом[28].

Клієнтська частина (React + Vite).

Клієнтська частина використовує React і збирається з допомогою Vite у статичний набір файлів. Після кожного оновлення у головній гілці репозиторію виконується автоматичний сценарій, який створює оптимізований каталог з ресурсами додатку. Цей каталог публікується як статичний сайт з підключеним CDN і власним доменом, захищений SSL-сертифікатом, що забезпечує швидку доставку HTML, JavaScript і CSS-файлів користувачам.

Серверна частина (Node.js + Express).

Серверна частина працює як окремий веб-сервіс із підтримкою протоколів HTTP/2 і TLS 1.2+. Після завантаження коду з репозиторію виконується «npm install», і головний скрипт запускається через «npm start». Завдяки горизонтальному масштабуванню автоматично додаються додаткові екземпляри сервера при зростанні навантаження. Безпека забезпечується механізмами Express (Helmet, rate-limiting) та захистом від DDoS-атак. Конфіденційні дані зберігаються у змінних середовища, відокремлених від репозиторію.

База даних (Microsoft SQL Server).

База даних відповідає за моделювання та зберігання структурованих даних, функціонуючи на окремому сервері з приватним мережевим доступом. Ініціалізація таблиць здійснюється SQL-скриптами під час першого запуску чи через систему міграцій. Сервер підтримує регулярне автоматичне резервне копіювання та моніторинг ресурсів без простою.

Узагальнений процес розгортання включає підготовку коду з розподілом репозиторію на каталоги /frontend та /backend, де налаштовані скрипти для розробки та продакшен-збірки, а також конфігураційні файли.

Налаштування хмарного сервісу передбачає:

- для клієнта: підключення каталогу зі статичними файлами до CDN із власним доменом і SSL;

- для сервера: створення веб-сервісу з командою запуску та змінними середовища через веб-інтерфейс платформи;
- автоматизація CI/CD із GitHub забезпечує автоматичний запуск збірки, тестування та деплой при кожному push до головної гілки, розгортаючи фронтенд на CDN, а бекенд на платформу;
- моніторинг і підтримка використовують можливості платформи для збору логів, метрик продуктивності та сповіщень про критичні події. У разі потреби здійснюється динамічне масштабування або відновлення з резервних копій.

Цей підхід гарантує безперервну доступність проєкту для кінцевих користувачів, гнучкість у масштабуванні та високий рівень безпеки даних.

4.5 Опис роботи програми

Метою опису програми є демонстрація сторінок проєкту як логічно взаємопов’язаних компонентів, що забезпечують повноцінну функціональність системи відповідно потребам користувачів. Кожна сторінка відповідає за реалізацію окремого аспекту функціоналу, а їхня сукупність створює єдину структуру. Вона дозволяє ефективно керувати особистими фінансами, працювати з базою даних, створювати записи, аналізувати інформацію та інше. Представлено послідовний опис ключових сторінок інтерфейсу користувача: від форми авторизації та головної панелі до сторінок налаштування й облікового запису, доступного лише для користувачів із певними правами. Розбір включає логіку переходів між сторінками, огляд основних елементів інтерфейсу, особливості роботи з даними, а також деякі аспекти реалізації окремих функцій.

- Сторінка авторизації і сторінка реєстрації.

Сторінка авторизації системи автоматично відкривається під час завантаження вебзастосунку. На ній реалізовано всі необхідні функції для авторизації користувача, при умові його реєстрації до цього та на сторінці реєстрації реалізовано попередню реєстрацію нового користувача для входу.

Реєстрація

Для створення нового облікового запису користувачу необхідно перейти на вкладку реєстрації, де потрібно заповнити наступні обов’язкові поля:

- ім’я;
- адреса електронної пошти;
- пароль.

Після введення даних і їх підтвердження відбувається надсилання запиту на сервер. У разі успішної реєстрації користувача автоматично переадресовують на сторінку входу до системи.

Сторінку реєстрації представлено на рис. 18.

Авторизація

Форма авторизації дозволяє увійти до системи за умови попередньої реєстрації, ввівши:

- електронну пошту;
- пароль.

Якщо дані введено правильно, система підтверджує їх коректність і перенаправляє на головну сторінку. У разі помилки система виводить модальне повідомлення, яке інформує про некоректність введених даних.

Сторінку авторизації представлено на рис. 19.

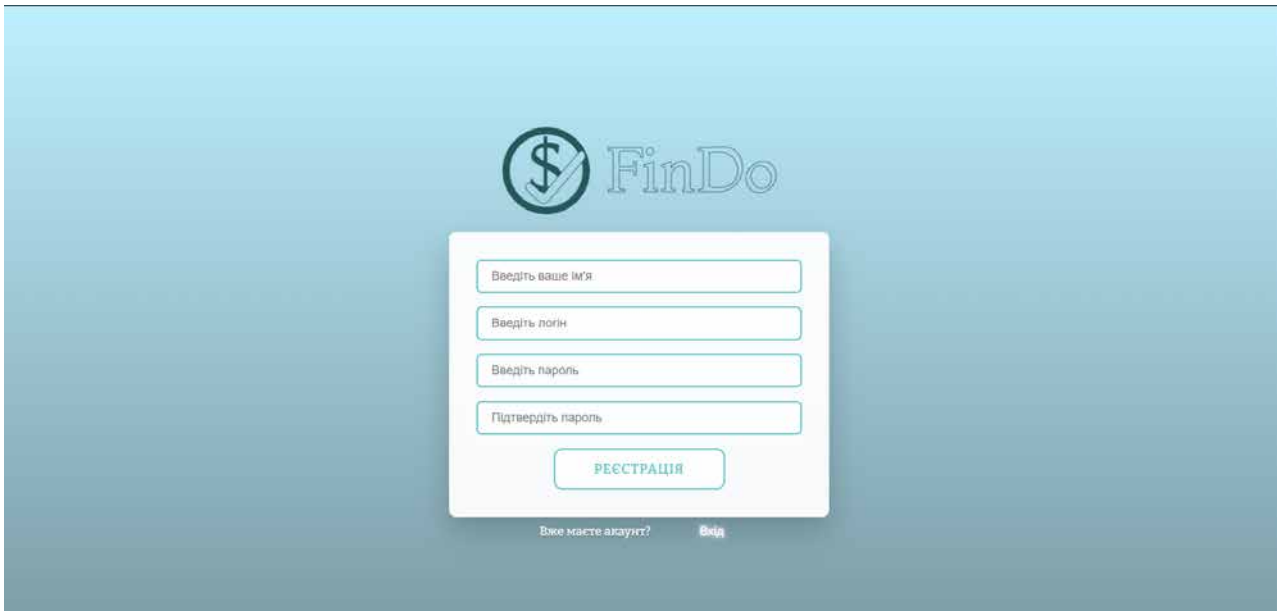


Рис. 18 Сторінка реєстрації

Рис. 19 Сторінка авторизації

Якщо під час входу введено облікові дані адміністратора, система перенаправляє на сторінку адмін-панелі замість головної. На цій сторінці відкривається список усіх зареєстрованих користувачів. Адміністратор має доступ до інформації про кожного користувача, зокрема імені, електронної пошти та дати реєстрації, а також можливість видаляти облікові записи за необхідності. Сторінка адмін-панелі представлено на рис. 20.

- **Адмін-панелі.**

Доступ тільки адміністратору за даними адміністратора. Функціонал дозволяє переглядати повний список зареєстрованих користувачів із зазначенням їхніх імен, електронних адрес та дати реєстрації. Окрім цього, передбачена опція видалення облікових записів.

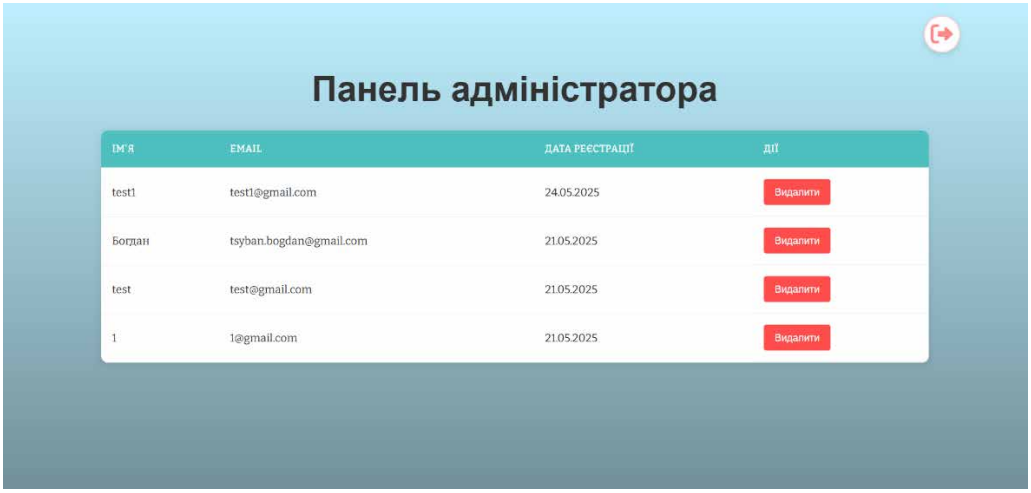


Рис. 20 Сторінка адмін-панелі

- **Головна сторінка**

Після перевірки заповнених полів на сторінці авторизації і успішному вході, користувач попадає на головну сторінку програми, призначену для використання головного функціоналу. Надається можливість переглядати загальний баланс, останні транзакції, події на день та інтегрований календар. Додатково пресдтавлена кругова діаграма для аналізу витрат за категоріями.

Головна сторінка представлена на рис. 21.

Рис. 21 Головна сторінка

З будь-якої частини програми, можна додати нову транзакцію або текстову операцію. Для цього необхідно натиснути на «Додати транзакцію» або ж «Додати текстовий…» відповідно. Після цього в залежності від потрібної дії, демонструється вікно додавання.

- **Вікно додавання транзакцій.**

Вікно для створення нових транзакцій. Користувач може вибрати категорію, тип транзакції (дохід або витрата), дату, рахунок і суму. Для кращої зручності типи транзакцій відображаються візуально: доходи позначаються синім кольором, а витрати — червоним.

Процес додавання нової транзакції представлено на рис. 22-24.

- **Вікно додавання текстових операцій.**

Вікно додавання текстової операції забезпечує можливість створення звичайних записів, подій чи завдань у системі. Це вікно дозволяє користувачам реєструвати текстову активність, яка відіграє важливу роль у плануванні справ або веденні щоденника.

Вікно додавання нової текстової операції представлено на рис. 25.

Рис. 22 Вибір типу транзакції

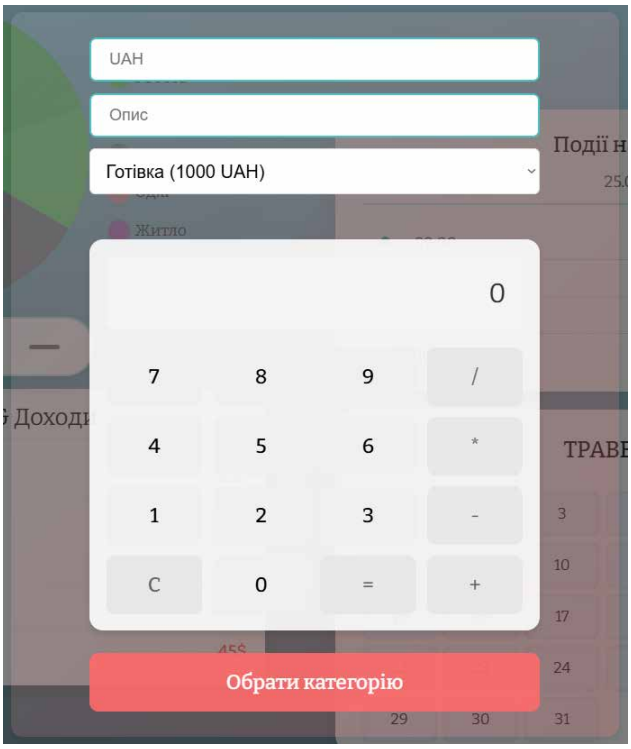


Рис. 23 Поля введення інформації про транзакції

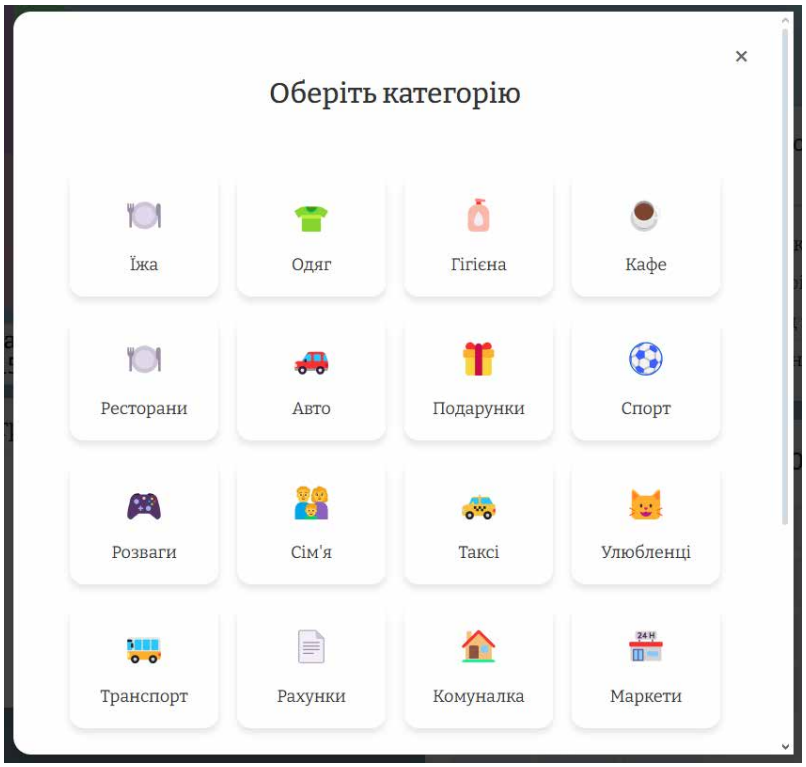


Рис. 24 Вибір категорії транзакції

Рис. 25 Додавання нової текстової операції

- **Сторінка категорій**

Ця сторінка дозволяє управляти категоріями транзакцій, вибираючи витрати чи доходи. Категорії представлені у вигляді карток із іконками. Користувачі можуть створювати нові категорії, редагувати або видаляти існуючі. Окрім цього, реалізовано розподіл категорій на загальні для всіх та

персоналізовані для кожного користувача. Для додавання нової категорії змодельоване вікно з вибором типу транзакції і вводом інформації.

Сторінка категорій представлено на рис. 26.

Рис. 26 Сторінка категорій

- **Сторінка рахунків**

На сторінці рахунків створюють, переглядають, редагують або видаляють фінансові рахунки. Кожен рахунок містить назву та поточний баланс, а також функцію встановлення рахунку за замовчуванням. Інтерфейс побудований у вигляді карток зі списком, а для керування рахунками використовуються модальні вікна. Для додавання нового рахунку є вікно з вводом інформації.

Сторінка рахунків представлено на рис. 27.

Рис. 27 Сторінка рахунків

- **Сторінка валют**

Ця сторінка надає доступні валюти разом із умовними позначеннями.

Сторінка валют представлено на рис. 28.

Рис. 28 Сторінка валют

- **Сторінка статистики**

Сторінка статистики забезпечує графічне представлення даних про транзакції користувача. Інформація подається у формі діаграми, що дає змогу аналізувати витрати за категоріями, рахунками та часовими інтервалами.

Сторінка статистики представлено на рис. 29.

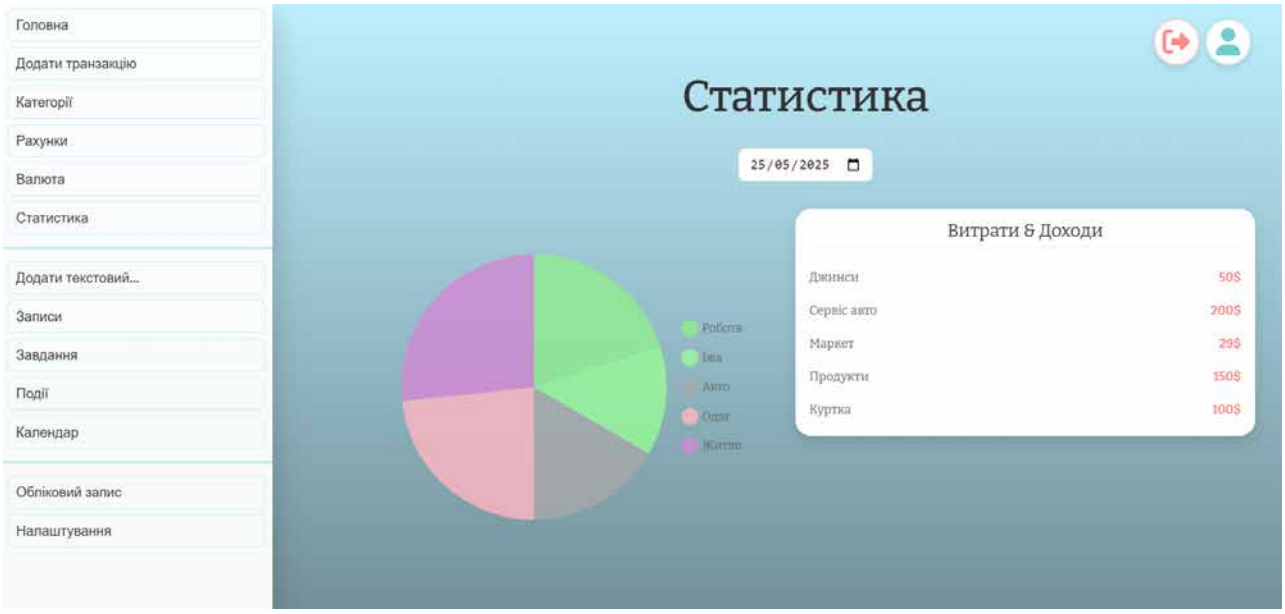


Рис. 29 Сторінка статистики

- **Сторінка записів**

Ця сторінка призначена для перегляду всіх записів користувача. Дані представлені у вигляді списку, де можна передивитись всі зроблені записи.

Сторінка записів представлено на рис. 30.

Рис. 30 Сторінка записів

- **Сторінка завдань**

Ця сторінка дозволяє створювати завдання. Для кожного завдання можна вказати опис і статус виконання, що корисно для позначення виконаного завдання.

Сторінка завдань представлено на рис. 31.

Рис. 31 Сторінка завдань

- **Сторінка подій**

На сторінці подій користувач може створювати або переглядати особисті події. Події прив’язуються до дати і супроводжуються коротким описом, відображаючись у хронологічному порядку.

Сторінка подій представлено на рис. 32.

Рис. 32 Сторінка подій

- **Сторінка календаря**

Календар показує поточний місяць. Вибір конкретного дня дає змогу переглянути події й перейти до відповідної вкладки, полегшуючи навігацію.

Сторінка календаря представлено на рис. 33.

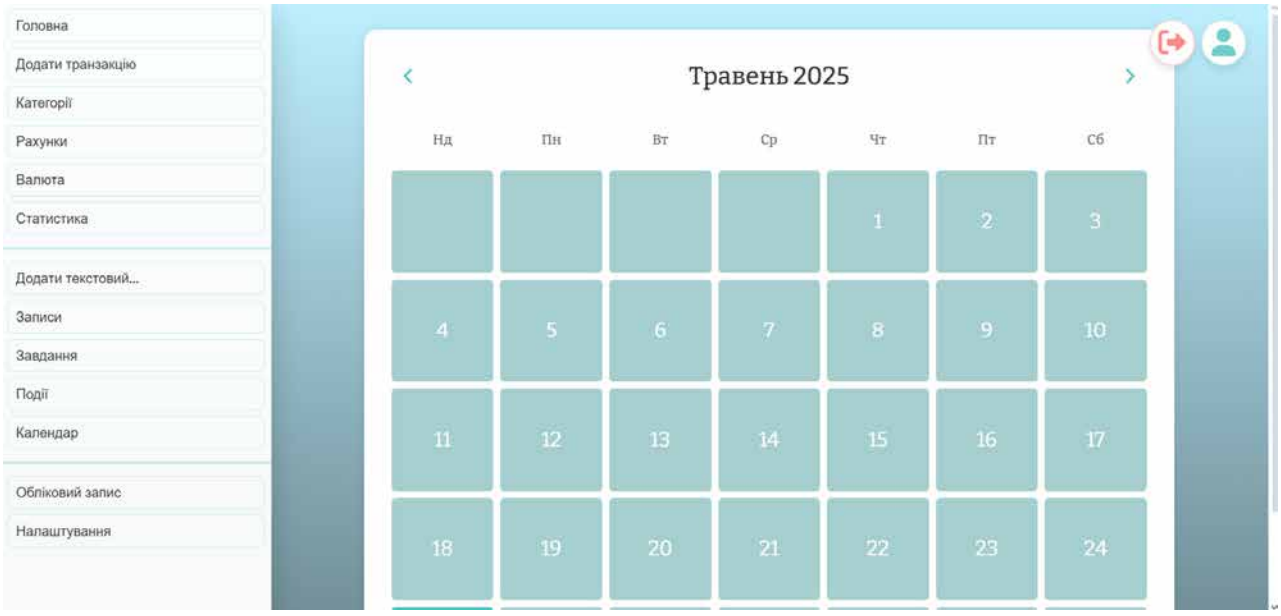


Рис. 33 Сторінка календаря

- **Сторінка облікового запису**

Ця сторінка містить персональну інформацію користувача: ім’я, email та інші дані при уточненні користувачем.

Сторінка облікового запису представлено на рис. 34.

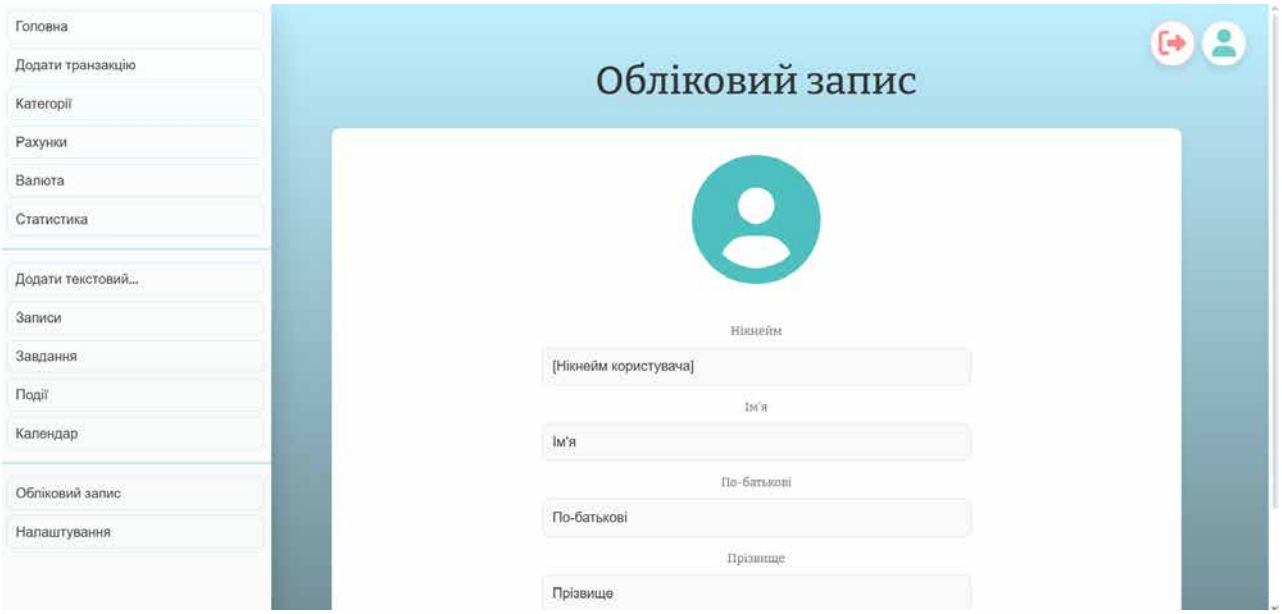


Рис. 34 Сторінка облікового запису

- **Сторінка налаштувань**

На сторінці налаштувань користувач може змінювати персоналізувати інтерфейс, формат дати, валюту за замовчуванням та встановлювати інші параметри персоналізації системи.

Сторінка налаштувань представлено на рис. 35.

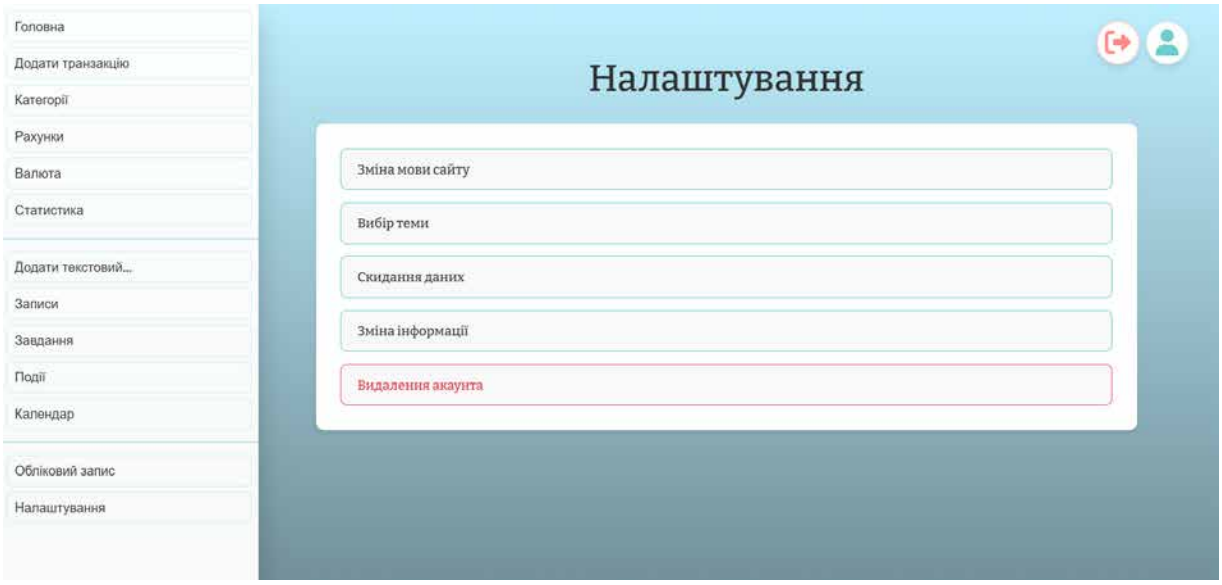


Рис. 35 Сторінка налаштувань

ВИСНОВКИ

У процесі розробки проєкту було успішно створено інформаційну систему, призначену для управління особистими фінансами, записами, подіями та завданнями. Робота охоплювала аналіз предметної області, проєктування, реалізацію, тестування та розгортання програмного забезпечення.

На етапі аналізу, тобто першого розділу, досліджено існуючі аналоги систем управління фінансами й організації справ, визначено ключові вимоги до системи. Для моделювання процесів створено діаграми прецедентів, послідовності та активності.

У другому розділі виконано розробку структури бази даних: створено ER-діаграму та налаштовано зв’язки між ключовими сутностями. Для збереження даних було обрано Microsoft SQL Server, що забезпечує високу продуктивність і надійність.

Третій розділ зосереджувався на архітектурному проєктуванні та розробці клієнт-серверної архітектури. Клієнтська частина створена за допомогою React.js і Vite, а серверна – з використанням Node.js/Express. Реалізовано інтеграцію з базою даних і впроваджено механізми автентифікації, керування транзакціями, категоріями, рахунками, подіями, завданнями, записами та додатково користувачами.

На етапі тестування, у четвертому розділі, перевірено функціональність системи, за допомогою тестів, на відповідність встановленим вимогам. Підготовлено впровадження проєкту з використанням хмарної інфраструктури. Описано процедури адміністрування й особливості роботи системи для користувачів.

Розроблена система являє собою багатофункціональний та адаптивний інструмент, який підходить як для використання окремими користувачами, так і для масштабування у ширших контекстах. Створена система вже забезпечила основу для майбутнього функціонального розширення та масштабування.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Головань М. С., Паламарчук В. Ю. Управління особистими фінансами: сучасні підходи та інструменти. Економіка та суспільство. – 2021.

2. Денисюк С. В. Цифрові рішення для організації повсякденних завдань. Інформаційні технології і засоби навчання. – 2020.

3. Monefy – [Електронний ресурс]. – Режим доступу: https://monefy.me (дата звернення 10.04.2025)

4. PocketGuard – [Електронний ресурс]. – Режим доступу: https://pocketguard.com (дата звернення 10.04.2025)

5. Todoist – [Електронний ресурс]. – Режим доступу: https://todoist.com (дата звернення 10.04.2025)

6. Microsoft To Do – [Електронний ресурс]. – Режим доступу: https://www.microsoft.com/uk-ua/microsoft-365/microsoft-to-do-list-app (дата звернення 10.04.2025)

7. Офіційна документація React – [Електронний ресурс]. – Режим доступу: https://reactjs.org (дата звернення 08.04.2025)

8. Офіційна документація Node.js. – [Електронний ресурс]. – Режим доступу: https://nodejs.org (дата звернення 09.04.2025)

9. Microsoft SQL Server documentation – [Електронний ресурс]. – Режим доступу: https://learn.microsoft.com/sql (дата звернення 11.04.2025)

10. Lucidchart: UML Use Case Diagram – [Електронний ресурс]. – Режим доступу: https://www.lucidchart.com/pages/uml-use-case-diagram (дата звернення 15.04.2025)

11. Lucidchart: UML Sequence Diagram – [Електронний ресурс]. – Режим доступу: https://www.lucidchart.com/pages/uml-sequence-diagram (дата звернення 16.04.2025)

12. Lucidchart: UML Activity Diagram – [Електронний ресурс]. – Режим доступу: https://www.lucidchart.com/pages/uml-activity-diagram (дата звернення 16.04.2025)

13. ER Diagram Tutorial – [Електронний ресурс]. – Режим доступу: https://www.guru99.com/uk/er-diagram-tutorial-dbms.html (дата звернення 11.04.2025)

14. Vite – [Електронний ресурс]. – Режим доступу: https://vite.dev/guide/ (дата звернення 09.04.2025)

15. Chart.js – [Електронний ресурс]. – Режим доступу: https://www.chartjs.org (дата звернення 16.04.2025)

16. Повний огляд REST – [Електронний ресурс]. – Режим доступу: https://dou.ua/forums/topic/50364/ (дата звернення 18.04.2025)

17. Express.js documentation – [Електронний ресурс]. – Режим доступу: https://expressjs.com/uk/ (дата звернення 20.04.2025)

18. DevZone. Як використовувати JSON Web Tokens (JWT) для автентифікації – [Електронний ресурс]. – Режим доступу: https://devzone.org.ua/post/iak-vykorystovuvaty-json-web-tokens-jwt-dlia-avtentyfikatsiyi (дата звернення 19.04.2025)

19. Axios documentation – [Електронний ресурс]. – Режим доступу: https://axios-http.com/uk/docs/intro (дата звернення 18.04.2025)

20. dotenv– [Електронний ресурс]. – Режим доступу: https://www.npmjs.com/package/dotenv (дата звернення 19.04.2025)

21. HTTP CORS guide – [Електронний ресурс]. – Режим доступу: https://developer.mozilla.org/en-US/docs/Web/HTTP/Guides/CORS (дата звернення 20.04.2025)

22. Visual Studio Code documentation – [Електронний ресурс]. – Режим доступу: https://code.visualstudio.com/docs (дата звернення 05.04.2025)

23. Microsoft SQL Server Management Studio (SSMS) – [Електронний ресурс]. – Режим доступу: https://learn.microsoft.com/en-us/ssms/sql-server-management-studio-ssms (дата звернення 05.04.2025)

24. Postman – [Електронний ресурс]. – Режим доступу: https://learning.postman.com/docs/introduction/overview/ (дата звернення 24.04.2025)

25. Інструкція Git для новачків – [Електронний ресурс]. – Режим доступу: https://dan-it.com.ua/uk/blog/instrukcija-git-dlja-novachkiv-shho-ce-take-jak-vin-pracjuie-ta-jaki-ie-osnovni-komandi/ (дата звернення 16.04.2025)

26. GitHub – [Електронний ресурс]. – Режим доступу: https://github.com (дата звернення 13.04.2025)

27. UML diagrams overview – [Електронний ресурс]. – Режим доступу: https://evergreens.com.ua/ua/articles/uml-diagrams.html (дата звернення 09.04.2025)

28. Що таке deploy і навіщо він потрібен – [Електронний ресурс]. – Режим доступу: https://foxminded.ua/deploi-tse/ (дата звернення 26.04.2025)

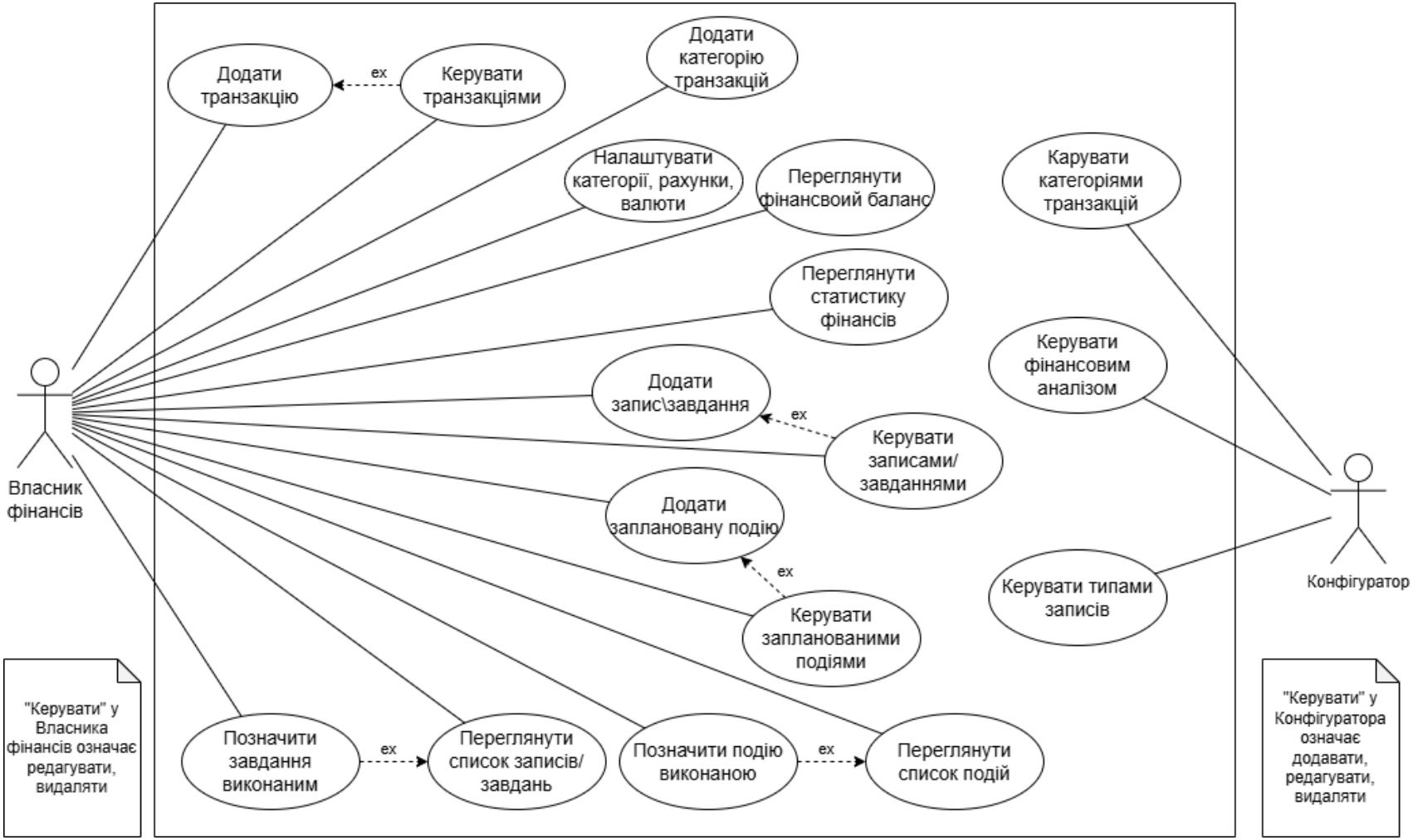
ДОДАТОК А

Діаграми предметної області

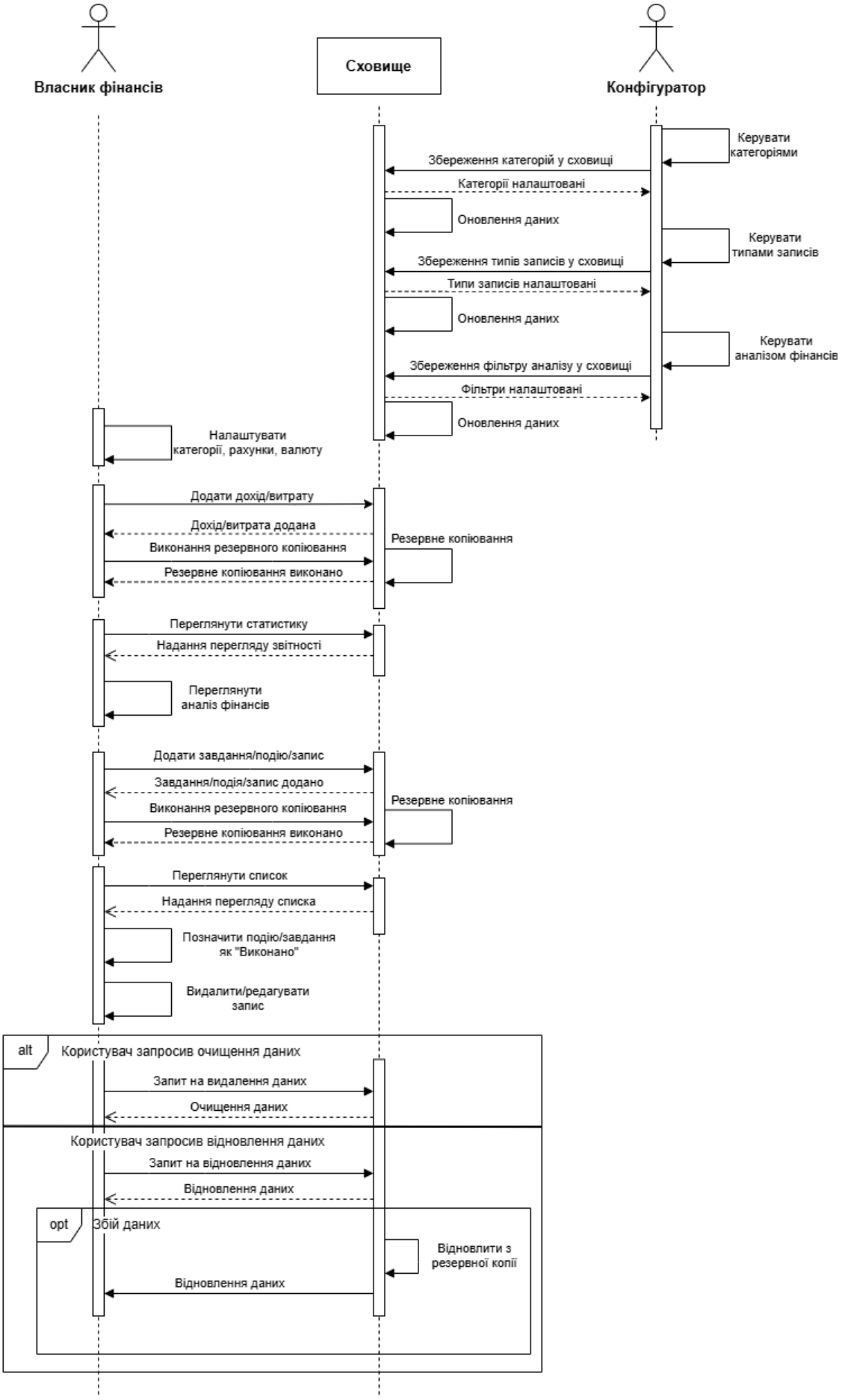
Сторінок 3

Київ-2025

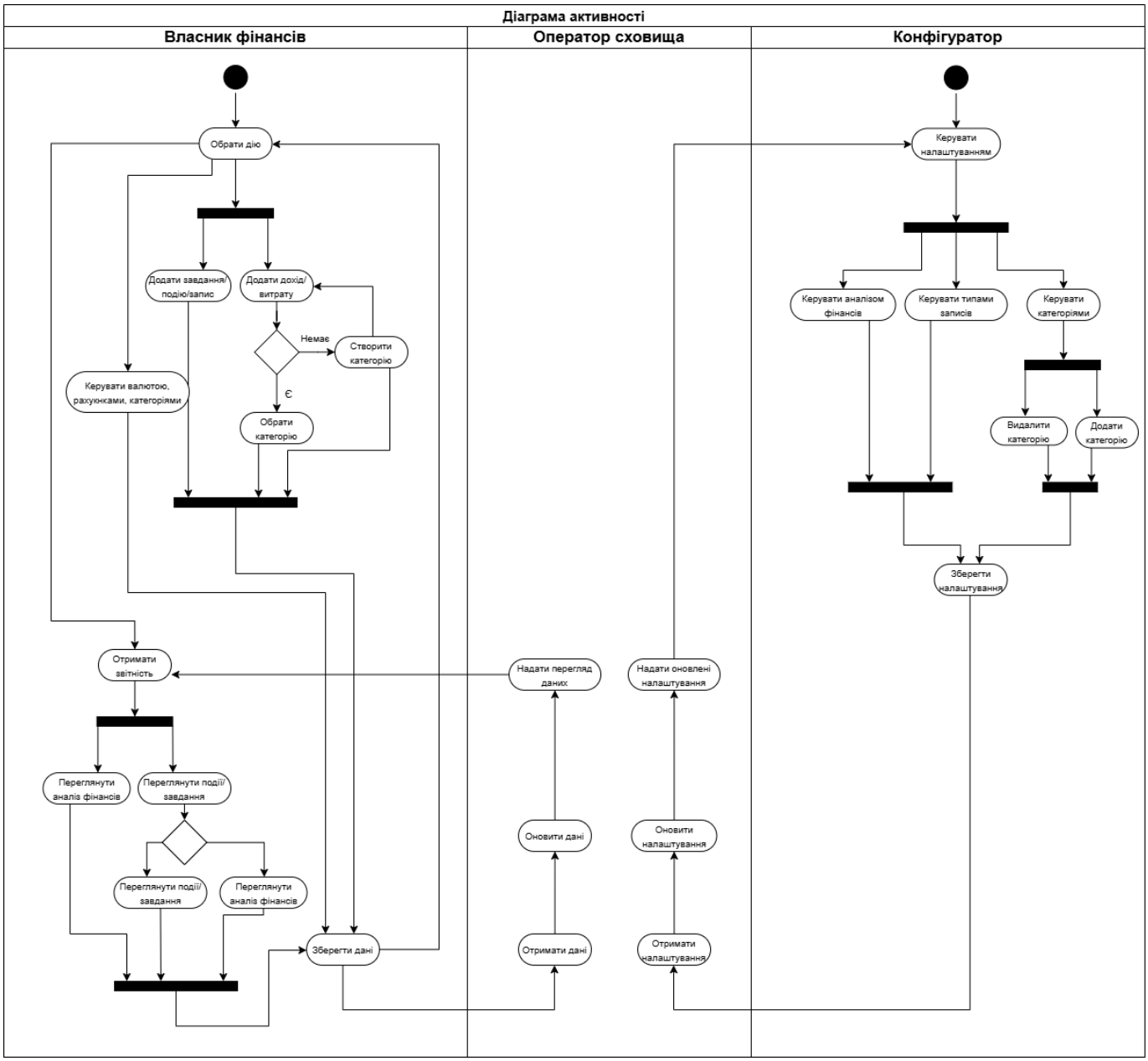
Діаграма прецедентів



Діаграма послідовності


Власник фінансів
Сховище
Конфігуратор
Керувати категоріями
Збереження категорій у сховищі
Категорії налаштовані
Оновлення даних
Керувати типами записів
Збереження типів записів у сховищі
Типи записів налаштовані
Оновлення даних
Керувати аналізом фінансів
Збереження фільтру аналізу у сховищі
Фільтри налаштовані
Оновлення даних
Налаштувати категорії, рахунки, валюту
Додати дохід/витрату
Дохід/витрата додана
Виконання резервного копіювання
Резервне копіювання виконано
Резервне копіювання
Переглянути статистику
Надання перегляду звітності
Переглянути аналіз фінансів
Додати завдання/подію/запис
Завдання/подія/запис додано
Виконання резервного копіювання
Резервне копіювання виконано
Резервне копіювання
Переглянути список
Надання перегляду списка
Позначити подію/завдання як "Виконано"
Видалити/редагувати запис
alt
Користувач запросив очищення даних
Запит на видалення даних
Очищення даних
Користувач запросив відновлення даних
Запит на відновлення даних
Відновлення даних
opt
Збій даних
Відновлити з резервної копії
Відновлення даних

Діаграма активності



ДОДАТОК Б

SQL-запити створення таблиць

```
-- 1. OwnersFinances
CREATE TABLE OwnersFinances (
    OwnerID CHAR(10) PRIMARY KEY,
    OwnerName NVARCHAR(25) NOT NULL,
    Email NVARCHAR(50) NOT NULL UNIQUE,
    Password NVARCHAR(255) NOT NULL,
    RegistrationDate DATETIME NOT NULL,
    CONSTRAINT CHK_Email CHECK (Email LIKE '%_@__%.__%')
);
-- 2. AccountType
CREATE TABLE AccountType (
    AccountTypeID CHAR(10) PRIMARY KEY,
    TypeName NVARCHAR(25) NOT NULL
);
-- 3. Account
CREATE TABLE Account (
    AccountID CHAR(10) PRIMARY KEY,
       AccountName NVARCHAR(25) NOT NULL,
    OwnerID CHAR(10) NOT NULL,
    AccountTypeID CHAR(10) NOT NULL,
    Balance DECIMAL(19, 2) NOT NULL DEFAULT 0,
    Currency CHAR(3) NOT NULL,
    FOREIGN KEY (OwnerID) REFERENCES OwnersFinances(OwnerID),
    FOREIGN KEY (AccountTypeID) REFERENCES AccountType(AccountTypeID)
);
-- 4. Category
CREATE TABLE Category (
    CategoryID CHAR(10) PRIMARY KEY,
    CategoryName NVARCHAR(30) NOT NULL,
    TransactionType NVARCHAR(15) NOT NULL CHECK (TransactionType IN ('Income', 'Expense')),
       Icon NVARCHAR(10) NULL,
    OwnerID CHAR(10) NULL,
    FOREIGN KEY (OwnerID) REFERENCES OwnersFinances(OwnerID)
);
-- 5. Transactions
CREATE TABLE Transactions (
    TransactionID CHAR(10) PRIMARY KEY,
    AccountID CHAR(10) NOT NULL,
    CategoryID CHAR(10) NOT NULL,
    TransactionName NVARCHAR(50) NOT NULL,
    Amount DECIMAL(19,2) NOT NULL,
    TransactionDate DATETIME NOT NULL DEFAULT GETDATE(),
    OwnerID CHAR(10) NOT NULL,
    FOREIGN KEY (AccountID) REFERENCES Account(AccountID),
    FOREIGN KEY (CategoryID) REFERENCES Category(CategoryID),
    FOREIGN KEY (OwnerID) REFERENCES OwnersFinances(OwnerID)
);
-- 6. Operation
CREATE TABLE Operation (
    OperationID CHAR(10) PRIMARY KEY,
    OwnerID CHAR(10) NOT NULL,
    OperationName NVARCHAR(50) NOT NULL,
    OperationType NVARCHAR(25) NOT NULL CHECK (OperationType IN ('Event', 'Task',
'Record')),
    Description NVARCHAR(255),
    DateCreated DATETIME NOT NULL DEFAULT GETDATE(),
    PlannedDate DATETIME NULL,
    FOREIGN KEY (OwnerID) REFERENCES OwnersFinances(OwnerID)
);
```

ДОДАТОК В

Діаграми програмної системи

Сторінок 6

Київ-2025

Абстракції

Абстракція: Транзакція
Властивості: Назва Категорія Сума Опис Рахунок
Обоов'язки: Зберігати інформацію про транзакцію. Керувати транзакціями. Підтримувати валюти і рахунки.

Абстракція: Рахунок
Властивості: Назва Баланс Валюта
Обоов'язки: Зберігати інформацію про рахунок. Оновлювати баланс рахунку при внесенні доходу чи витрати. Обирати тип рахунку при здійснені транзакцій.

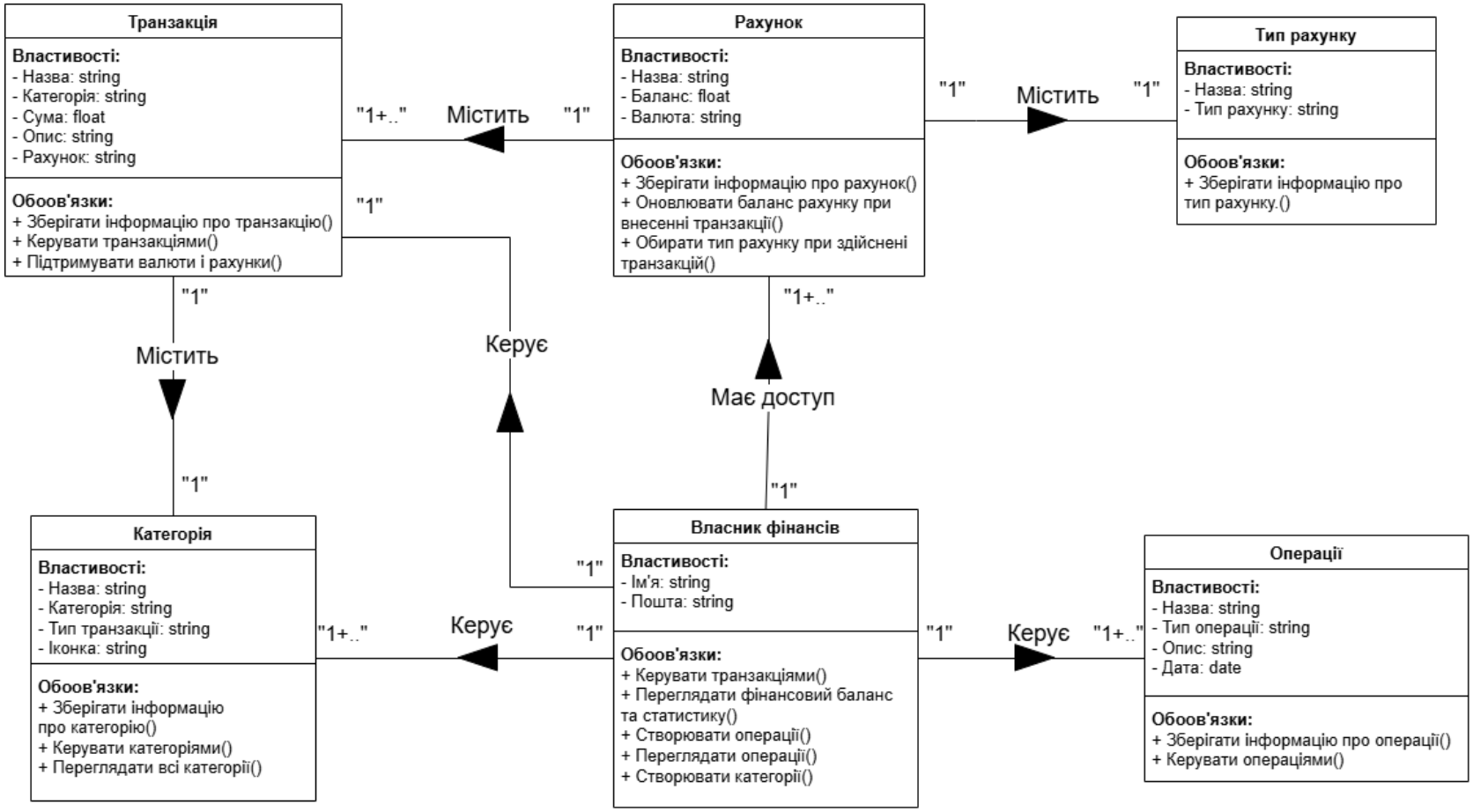
Абстракція: Тип рахунку
Властивості: Назва Тип рахунку
Обоов'язки: Зберігати інформацію про тип рахунку.

Абстракція: Категорія
Властивості: Назва Категорія Тип транзакції Іконка
Обоов'язки: Зберігати інформацію про категорію. Керувати категоріями. Переглядати всі категорії.

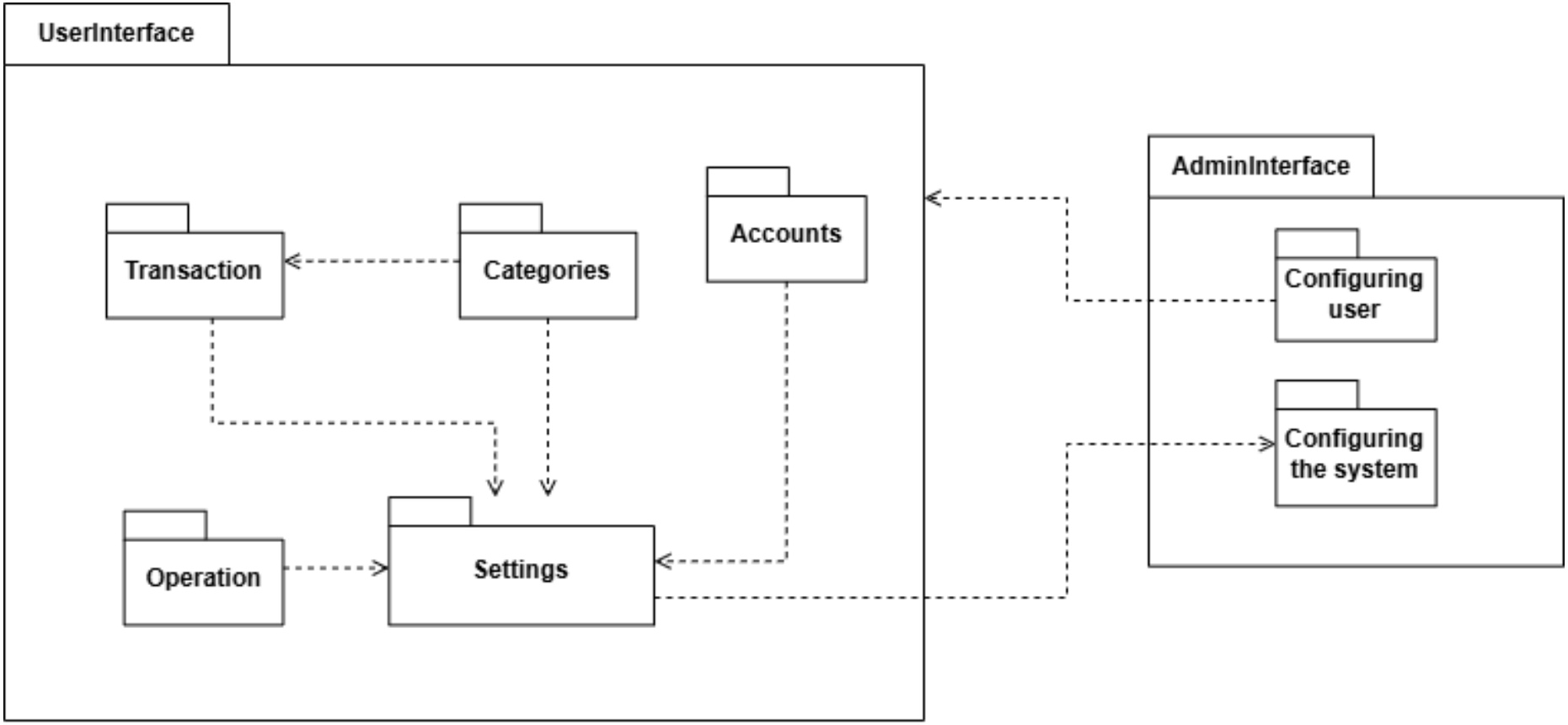
Абстракція: Власник фінансів
Властивості: Ім'я Пошта
Обоов'язки: Керувати транзакціями. Переглядати фінансовий баланс та статистику. Створювати операції. Переглядати операції. Створювати категорії.

Абстракція: Операції
Властивості: Назва Тип операції Опис Дата
Обоов'язки: Зберігати інформацію про операції. Керувати операціями.

Діаграма класів



Діаграма пакетів

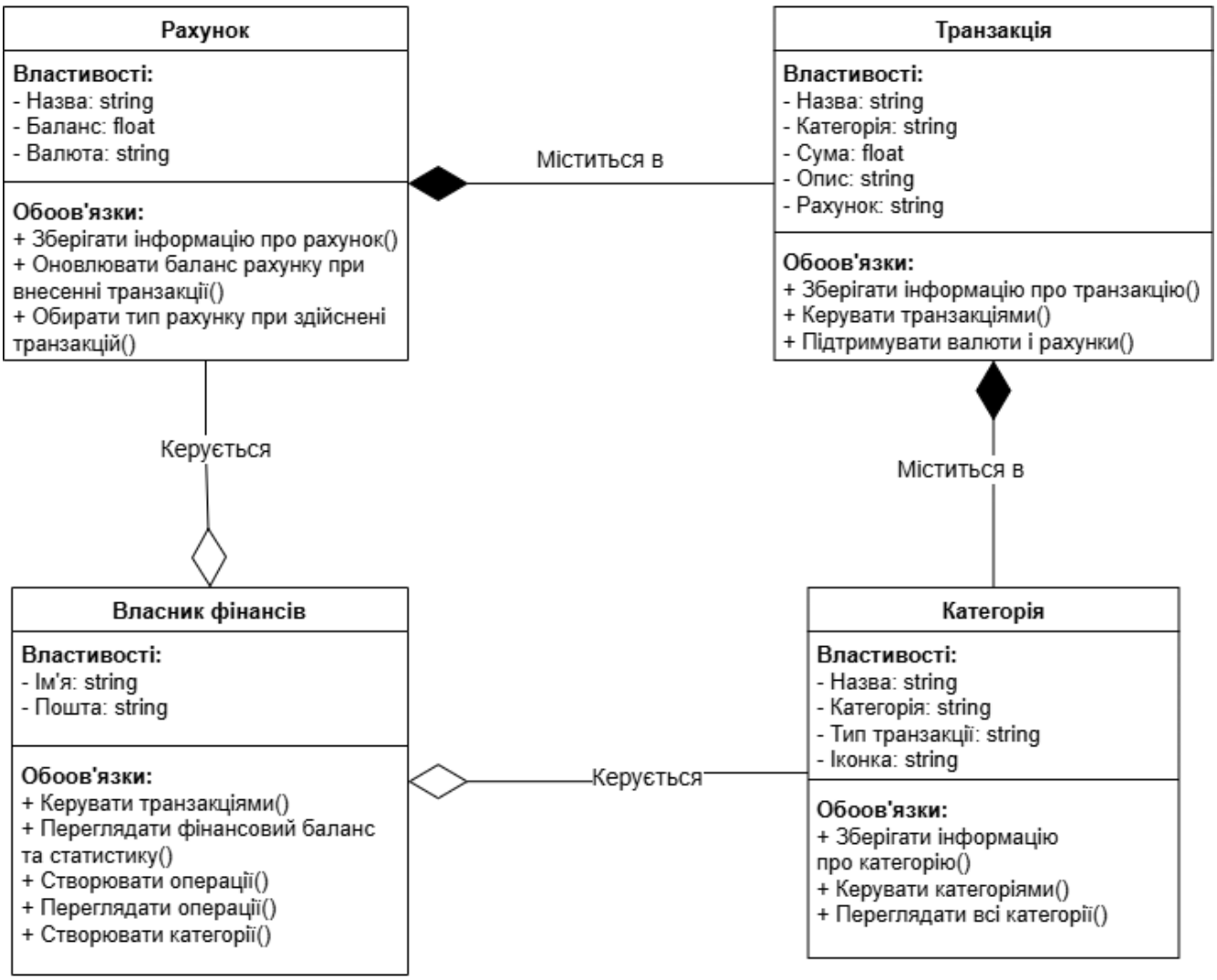


Прості кооперації

Сторінка 5

Прості кооперації

Керування фінансами



Діаграма розгортання