НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ БІОРЕСУРСІВ І ПРИРОДОКОРИСТУВАННЯ УКРАЇНИ
ФАКУЛЬТЕТ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

ДОПУСКАЄТЬСЯ ДО ЗАХИСТУ

Завідувач кафедри

Комп'ютерних наук

_____Голуб Б.Л.

" " 2025 р.

БАКАЛАВРСЬКА КВАЛІФІКАЦІЙНА РОБОТА

на тему

«Програмне забезпечення інформаційної системи управління фінансами для фізичної особи-підприємця»

Спеціальність 121 «Інженерія програмного забезпечення»

Гарант освітньої програми

К.т.н., доцент _____ / Вайганг Г.О. /
(Науковий ступень та вчене звання) (підпис) (ПІБ)

Керівник бакалаврської кваліфікаційної роботи

к.т.н., доцент _____ Бородкін Г. О.
(Науковий ступень та вчене звання) (підпис) (ПІБ)

Виконав _____ Маслік М. А.
(підпис) (ПІБ)

КИЇВ-2025

НАЦІОНАЛЬНИЙ УНІВЕРСТИТЕТ БІОРЕСУРСІВ І ПРИРОДОКОРИСТУВАННЯ УКРАЇНИ

Факультет інформаційних технологій

ЗАТВЕРДЖУЮ
Завідувач кафедри
Комп'ютерних наук
_____Голуб Б.Л.
_" "_____20 р._

З А В Д А Н Н Я

на виконання бакалаврської кваліфікаційної роботи студенту

Масліку Микиті Антоновичу

Спеціальність 121 – «Інженерія програмного забезпечення»

Тема бакалаврської кваліфікаційної роботи: «Програмне забезпечення інформаційної системи управління фінансами для фізичної особи-підприємця»

Затверджена наказом ректора НУБіП України від 16.12.2024 № 2249 «С».

Термін подання завершеної роботи на кафедру _____

Вихідні дані до бакалаврської кваліфікаційної роботи: документація бібліотек, нормативні та технічні документи

Перелік питань , які потрібно розробити: проаналізувати предметну область, аналоги, розробити вимоги до системи, спроектувати та розробити систему

Дата видачі завдання "_____"_____20___ р.

Керівник бакалаврської кваліфікаційної роботи

_____ _____ ___Бородкін Г. О. _

(науковий ступінь та вчене звання) (підпис) (ПІБ)

Завдання прийняв до виконання _____ Маслік М. А.

(підпис) (ПІБ студента)

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

ФОП – фізична особа-підприємець

ПЗ – програмне забезпечення

Backend – серверна частина додатку

Frontend – клієнтська частина додатку

PHP – мова програмування серверних додатків

UML – уніфікована мова моделювання

ERD – діаграма сутність-зв’язок

CSV – текстовий формат, що представляє табличні дані

XLSX – формат файлу для електронних таблиць

SQL – структурована мова запитів

NoSQL – сімейство баз даних, що не базуються на використанні не тільки мови SQL

API – програмний інтерфейс додатку

REST – архітектурний стиль взаємодії компонентів розподіленого додатку в мережі

СУБД – система управління базою даних

СКБД – система керування базою даних

ІС – інформаційна система

UI –інтерфейс користувача

MVC – архітектурний паттерн модель-вид-контролер

HTTP – протокол передачі гіпертекстових повідомлень

JSON – текстовий формат обміну даними, що базується на мові JavaScript

ACID – набір властивостей для реляційних баз даних

UUID – стандарт ідентифікації в програмному забезпеченні

ОС – операційна система

CRUD – акронім, що означає чотири базові функції: створення, читання, оновлення, видалення

POST – метод запиту на запис в HTTP протоколі

GET – метод запиту на читання в HTTP протоколі

HTML – мова гіпертекстової розмітки

CSS – каскадна таблиця стилів; мова для описання зовнішнього виду веб-документа

SSL – криптографічний протокол, що забезпечує безпечну передачу даних між клієнтом та сервером

ВСТУП

У сучасних економічних умовах фізичні особи-підприємці (ФОП) займають важливе місце в структурі національної економіки. Вони забезпечують гнучкість ринку, сприяють створенню нових робочих місць та активізують локальну ділову активність. Однак із розвитком підприємницької діяльності зростає й обсяг рутинних завдань, пов’язаних з обліком доходів і витрат, веденням фінансової документації, контролем податкових зобов’язань та формуванням звітності.

Традиційне ведення фінансів у вигляді паперових записів або електронних таблиць не завжди є ефективним, особливо за умови постійних змін у законодавстві та високої динаміки бізнес-процесів. У зв’язку з цим актуальною стає розробка інформаційної системи, що дозволяє фізичним особам-підприємцям автоматизувати управління фінансами, підвищити точність обліку, зменшити вплив людського фактору та заощадити час.

Метою кваліфікаційної роботи є створення програмного забезпечення інформаційної системи, яка дозволяє фізичній особі-підприємцю ефективно керувати фінансовими потоками, вести облік операцій, формувати звіти, контролювати податкові зобов’язання та забезпечувати зручний інтерфейс для роботи як з комп’ютера.

Об’єктом дослідження є процес управління фінансами фізичної особи-підприємця.

Предметом дослідження є методи та інструменти автоматизації фінансового обліку і звітності з використанням сучасних технологій веб-розробки.

Основні завдання дослідження:

1. провести аналіз існуючих інформаційних систем для управління фінансами ФОП;
2. визначити функціональні та нефункціональні вимоги до майбутнього програмного забезпечення;
3. обґрунтувати вибір технологій та архітектурних рішень;

4. розробити структуру і логіку інформаційної системи;

5. реалізувати веб-інтерфейс для зручної взаємодії користувача з системою;

6. забезпечити зберігання та обробку даних за допомогою сучасної бази даних;

7. провести тестування розробленого продукту та оцінити його відповідність поставленим вимогам.

Для реалізації програмного забезпечення використовуються сучасні веб-технології. Серверна (backend) частина розробляється за допомогою фреймворку Laravel 12.x (PHP), що забезпечує стабільність, масштабованість та розширюваність системи. Для клієнтської (frontend) частини використовується фреймворк Vue.js. У якості системи управління базами даних обрано PostgreSQL, яка дозволяє надійно зберігати фінансові записи та швидко обробляти запити.

Результатом роботи є створення функціонального програмного продукту, здатного полегшити управління фінансами для фізичних осіб-підприємців і підвищити ефективність їх діяльності.

1 Системний аналіз предметної області

1.1 Опис предметної області

Інформаційна система управління фінансами фізичної особи-підприємця (ФОП) призначена для автоматизації обліку доходів, витрат та інших фінансових операцій, що здійснюються підприємцем у рамках його господарської діяльності. Така система має на меті допомогти користувачу ефективно контролювати фінансовий стан, планувати бюджет, формувати звітність та аналізувати основні фінансові показники.

Сучасні фізичні особи-підприємці стикаються з необхідністю ведення точного і своєчасного обліку великої кількості фінансових операцій. Це включає прийом платежів, сплату податків, контроль за доходами і витратами, а також обробку різноманітних фінансових документів. Відсутність єдиного інструменту змушує більшість користувачів застосовувати паперові документи або фрагментарні електронні таблиці, що часто призводить до помилок, дублювання інформації та ускладнює процес аналізу даних.

Система покликана автоматизувати основні процеси фінансового обліку ФОП, що значно підвищить якість та оперативність ведення бізнесу.

1.1.1 Основні функції системи

Система забезпечує ведення обліку доходів і витрат підприємця з можливістю деталізації за категоріями. Крім того, вона дозволяє вести інформацію про клієнтів, контрагентів і банківські рахунки.

Для потреб аналітики і контролю система формує фінансові звіти: баланс, звіт про прибутки і збитки, касові операції, а також автоматично розраховує податкові зобов’язання. Інтерфейс системи орієнтований на зручність і швидкість введення та перегляду даних, що суттєво підвищує продуктивність роботи користувачів.

1.1.2 Основні користувачі системи

Основними користувачами системи є:

- фізична особа-підприємець – основний користувач, що веде облік власних фінансів, контролює виконання бюджету і приймає управлінські рішення на основі звітності;
- бухгалтер або консультант – допомагає у веденні фінансової документації, перевіряє звіти і здійснює податкові розрахунки;
- адміністратор системи – відповідає за підтримку працездатності системи, адміністрування користувачів та забезпечення безпеки даних.

1.1.3 Основні проблеми предметної області

Серед головних проблем, які існують у веденні фінансового обліку ФОП, можна виділити:

- відсутність єдиної централізованої системи обліку фінансових операцій;
- використання розрізнених інструментів, таких як паперові документи і електронні таблиці, що спричиняє помилки та втрати даних;
- ускладнення формування аналітичних звітів і прогнозування фінансового стану;
- недостатній рівень автоматизації, що потребує значних часових затрат на обробку інформації.

1.1.4 Потенціал автоматизації

Впровадження автоматизованої інформаційної системи управління фінансами дозволить об’єднати всю фінансову інформацію у єдиному середовищі з централізованим доступом за логіном і паролем. Це спростить введення даних завдяки інтуїтивному інтерфейсу, забезпечить точність і цілісність інформації через системи контролю і валідації даних. Автоматизація суттєво підвищить ефективність прийняття управлінських рішень, надаючи оперативний доступ до аналітики і звітів.

Система також передбачає масштабування у міру зростання бізнесу та змін законодавчих вимог, що гарантує її довгострокову актуальність і адаптивність. Запровадження автоматизації сприятиме не лише зниженню трудових і часових

витрат, а й підвищенню прозорості, надійності та контролю над фінансовими процесами фізичної особи-підприємця.

1.2 Аналіз вимог до програмної системи

Проаналізувавши предметну область розроблюваної інформаційної системи управління фінансами фізичної особи-підприємця, було визначено основні вимоги до програмного забезпечення. Аналіз вимог дозволяє сформувати чіткі очікування користувачів, встановити технічні обмеження, виявити потенційні ризики та обрати оптимальні технології для реалізації системи.

Функціональні вимоги:

- реєстрація та авторизація користувачів – система повинна забезпечувати безпечний вхід для фізичних осіб-підприємців, бухгалтерів та адміністраторів;
- ведення обліку доходів і витрат – можливість додавати, редагувати та видаляти записи про фінансові операції з деталізацією за категоріями (оренда, закупівля, послуги тощо);
- формування фінансових звітів – автоматичне створення звітів: баланс, звіт про прибутки і збитки, касові операції;
- розрахунок податкових зобов’язань – автоматичний підрахунок податків відповідно до чинного законодавства;
- планування бюджету і контроль виконання – інструменти для встановлення бюджетних лімітів і моніторингу їх дотримання;
- зручний інтерфейс для введення та перегляду даних – забезпечення швидкого і простого доступу до інформації;
- історія змін і резервне копіювання – збереження історії редагувань для контролю цілісності даних;

- ролі та права доступу – розмежування повноважень між користувачами для забезпечення безпеки і конфіденційності;
- імпорт/експорт даних – підтримка роботи з поширеними форматами для обміну інформацією (CSV, Excel).

Нефункціональні вимоги:

- продуктивність – система має забезпечувати швидкий відгук інтерфейсу навіть при великих обсягах даних;
- масштабованість – можливість розширення функціоналу без значних змін у базовій архітектурі;
- надійність – захист від втрати даних через регулярне резервне копіювання і відновлення;
- безпека – застосування сучасних методів аутентифікації і шифрування для захисту персональних даних;
- кросплатформність – доступність системи з різних пристроїв і браузерів;
- інтуїтивність інтерфейсу – простота у використанні без необхідності тривалого навчання.

Технічні вимоги:

- використання сучасних веб-технологій для реалізації інтерфейсу (React, Vue.js або Angular);
- серверна частина на базі PHP (Laravel) або іншої зручної серверної платформи;
- база даних реляційного типу (PostgreSQL, MySQL) з можливістю масштабування;
- використання RESTful API або GraphQL для взаємодії між клієнтською і серверною частинами;
- забезпечення захисту даних за допомогою HTTPS, JWT або OAuth для аутентифікації;
- інтеграція з системами резервного копіювання і моніторингу;

- можливість розгортання системи в контейнеризованому середовищі Docker.

Таким чином, сформовані вимоги визначають базову функціональність, якість роботи, безпеку та технологічну основу системи управління фінансами фізичної особи-підприємця. Їх реалізація забезпечить створення надійного, зручного та ефективного інструменту для ведення фінансової діяльності та прийняття управлінських рішень.

1.3 Моделювання предметної області

Моделювання предметної області є фундаментальним етапом у процесі створення інформаційної системи, що дозволяє формалізувати знання про майбутню систему, виявити взаємозв’язки між її компонентами та зрозуміти очікувану поведінку на різних етапах взаємодії з користувачем. У цьому проекті UML (Unified Modeling Language) використовується як основний засіб для візуального представлення логіки, структури та функціональності розроблюваної системи.

Основна мета моделювання полягає у створенні узгодженого бачення між замовником і розробником щодо ключових функцій, ролей користувачів і динаміки роботи системи. Це також забезпечує чітке уявлення про обсяг і складність реалізації функціоналу, дає змогу уникнути логічних помилок ще до початку програмної реалізації.

Для моделювання системи управління фінансами ФОП було обрано такі типи UML-діаграм:

1. діаграма прецедентів (Use Case Diagram) – демонструє, які основні функції система надає користувачам (акторам), і як вони взаємодіють із системою;
2. діаграма послідовності (Sequence Diagram) – описує покроковий сценарій виконання певної дії користувача та взаємодії між об'єктами системи в часі;

3. діаграма активності (Activity Diagram) – відображає логіку виконання бізнес-процесів, включаючи розгалуження, цикли та умови.

1.3.1 Діаграма прецедентів

Діаграма прецедентів відображає функціональну модель системи з точки зору кінцевого користувача і має наступний вигляд:

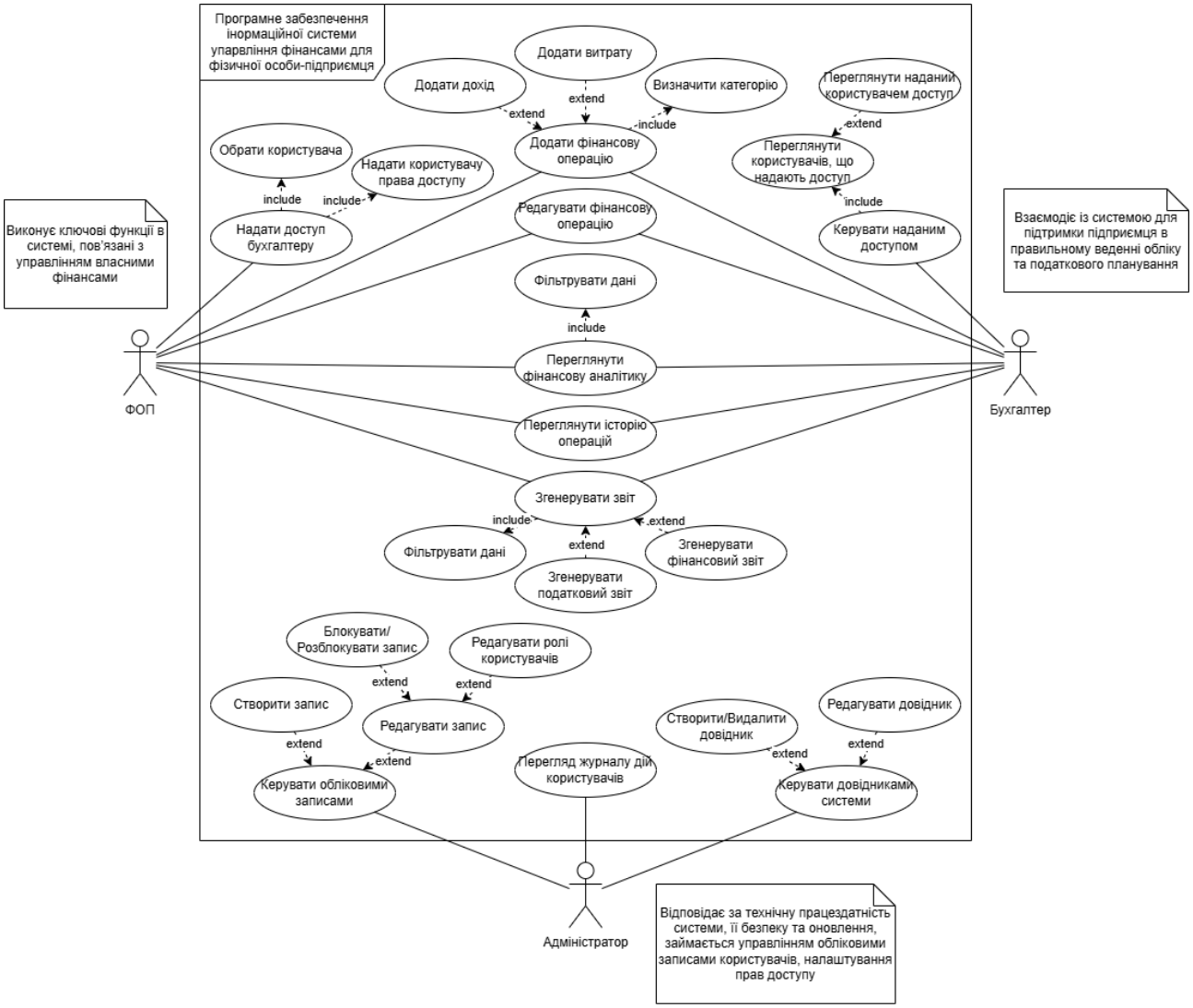

Рисунок 1 – Діаграма прецедентів

Основними акторами системи є:

- фізична особа-підприємець (ФОП) – головний користувач, який здійснює облік доходів і витрат та формує звітність;
- бухгалтер/консультант – допомагає з аналізом фінансових операцій, перевіркою правильності заповнення даних, поданням звітності.

- адміністратор системи – відповідає за управління обліковими записами, контроль доступу, резервне копіювання даних.

Основні прецеденти включають:

- керування доступами;
- додавання фінансових операцій;
- ведення обліку доходів і витрат;
- ведення податкової звітності;
- аналітика фінансової діяльності;
- керування обліковими записами адміністратором;
- перегляд адміністратором журналу дій користувачів;
- керування адміністратором довідниками системи.

1.3.2 Діаграма послідовності

Діаграма послідовності описує типовий сценарій роботи кожного із представлених раніше акторів у системі та має наступний вигляд:

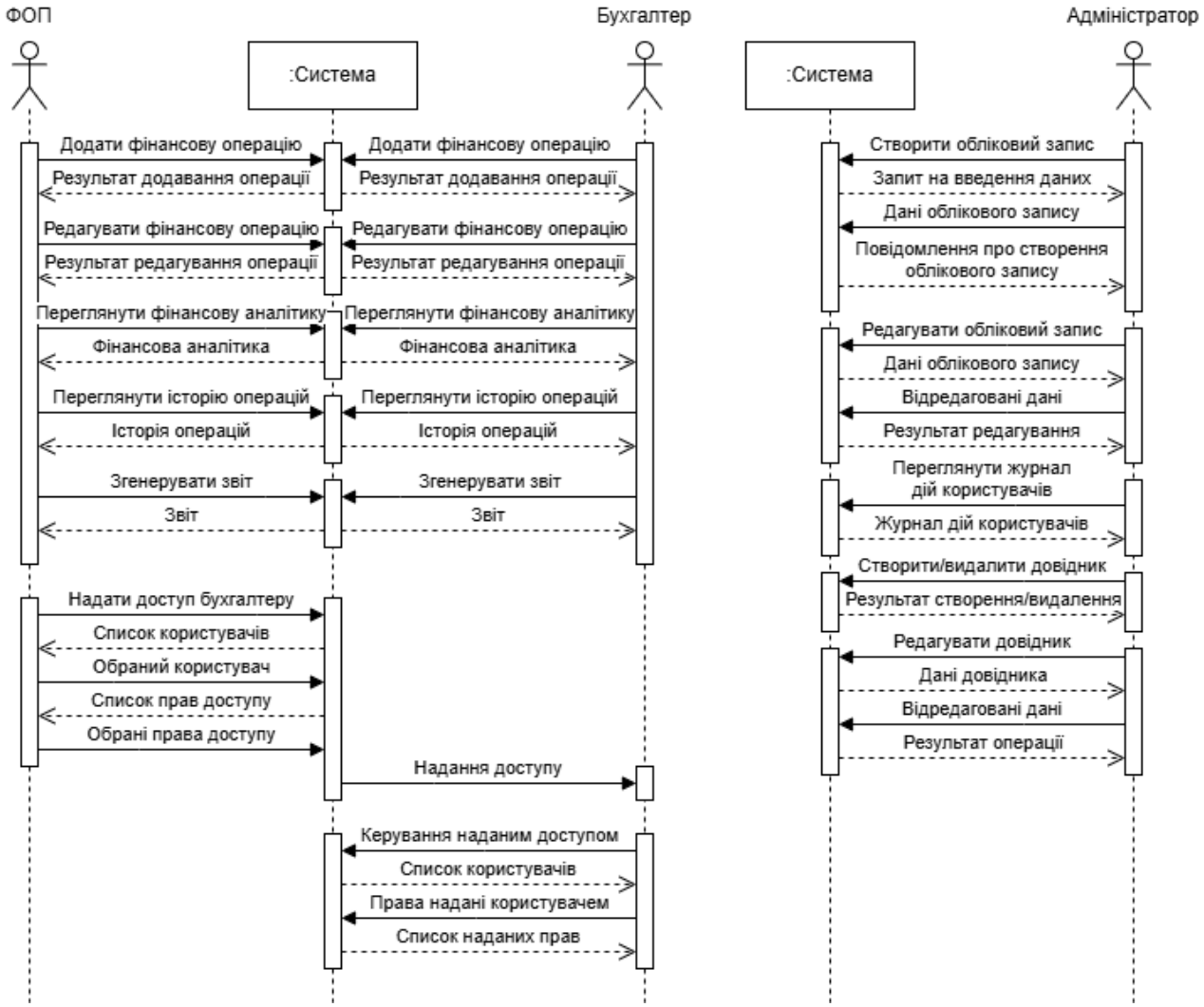

Рисунок 2 – Діаграма послідовності

Розроблюване програмне забезпечення працюватиме за принципом «запит-відповідь», що впливає на взаємодію із системою кожного із акторів.

1.3.3 Діаграма активності

Діаграма активності (див. рис. 3) в UML використовується для моделювання послідовності дій, логіки управління потоками та умов переходів між етапами виконання бізнес-процесу.

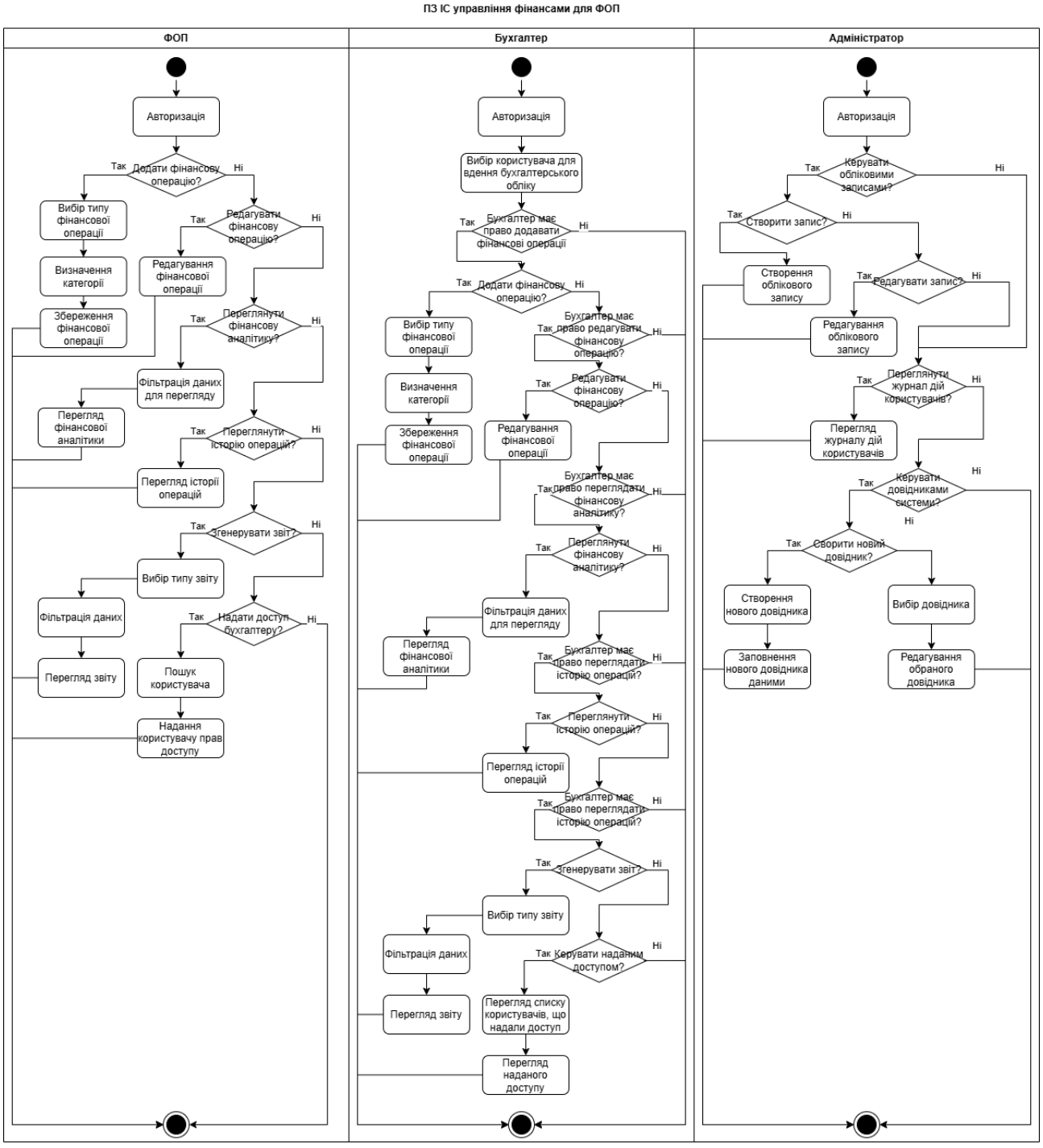

Рисунок 3 – Діаграма активності

Ця діаграма особливо ефективна для візуалізації алгоритмів, складних процесів обробки даних та сценаріїв взаємодії користувача з системою.

Переваги такої діаграми:

- чітко відображає умови прийняття рішень у процесі;

- полегшує розуміння алгоритмів не лише розробникам, а й замовникам;
- дозволяє виявити дублювання логіки або надмірну складність процесу ще на етапі проектування;
- сприяє майбутньому тестуванню – на основі діаграми легко виділити тестові сценарії.

1.4 Огляд інформаційних джерел та існуючих рішень

1.4.1 Огляд інформаційних джерел

У процесі дослідження та проектування інформаційної системи управління фінансами для фізичної особи-підприємця було опрацьовано низку літературних джерел, методичних матеріалів та онлайн-ресурсів. Основними джерелами стали:

- наукова література з тематики фінансового обліку [1][2], автоматизації бізнес-процесів та інформаційних систем [4][5];
- методичні рекомендації українських ЗВО щодо моделювання бізнес-процесів (UML);
- офіційна документація веб-фреймворків і бібліотек, зокрема: Laravel (laravel.com) [6][8][16], Vue.js (vuejs.org) [10], Chart.js (chartjs.org), TailwindCSS (tailwindcss.com);
- відкриті програмні продукти та сервіси, що частково реалізують схожі функції для обліку доходів і витрат.

1.4.2 Аналіз існуючих рішень

На ринку представлено декілька сервісів, які забезпечують часткову або повну автоматизацію фінансового обліку для ФОП. Основна увага приділяється веб-рішенням, які користуються популярністю серед малих підприємців в Україні:

- MOBI Finance – онлайн-сервіс для обліку доходів, витрат і податкових платежів. Підтримує автоматичне заповнення звітності (ЄСВ, декларації ФОП), надсилає нагадування, дозволяє формувати графіки платежів;
- Fairo – мобільний додаток для ФОП, орієнтований на банківську інтеграцію. Має функції автоматичного обліку надходжень на рахунок, генерації рахунків-фактур, обчислення податків. Спрощений інтерфейс, проте обмежений функціонал для аналітики;
- Oblik.Online – веб-платформа з широким функціоналом: планування бюджету, ведення книги обліку доходів, категоризація витрат, візуалізація даних у вигляді діаграм. Підтримка мультивалютності та інтеграція з банківськими сервісами;
- Money Lover – популярний фінансовий трекер, орієнтований на персональне використання. Підтримує імпорт/експорт даних, нагадування, звіти. Не спеціалізується на податкових звітах для ФОП.

Порівняння існуючих аналогів інформаційної системи

Таблиця 1.1

Критерій / Система	MOBI Finance	Fairo	Oblik.Online	Money Lover	Запропонована система
Ведення обліку доходів	Так	Так	Так	Так	Так
Ведення обліку витрат	Так	Обмежено	Так	Так	Так
Формування звітності ФОП	Так	Так	Обмежено	Ні	Так

Критерій / Система	MOBI Finance	Fairo	Oblik.Online	Money Lover	Запропонована система
Підтримка кількох ролей	Ні	Ні	Ні	Ні	Так (підприємець, бухгалтер)
Інтеграція з банком	Частково	Так	Так	Ні	Можливо (через API)
Планування бюджету	Обмежено	Ні	Так	Так	Так
Графічна аналітика	Так	Ні	Так	Так	Так
Мобільна версія	Так	Так	Так	Так	Можливо
Локалізація українською	Так	Так	Так	Частково	Так
Безкоштовна версія	Так (обмежено)	Так	Так (обмежено)	Так	Так

1.4.3 Відмінності запропонованого рішення

Інформаційна система, яка проектується, має низку ключових переваг:

- спеціалізація під українських ФОП – врахування локального податкового законодавства [1][2];
- гнучка структура доходів/витрат з категоризацією та автоматичними розрахунками;
- графічна аналітика для візуального контролю фінансового стану;
- підтримка ролей користувачів – можливість ведення обліку самим підприємцем або делегування бухгалтеру;
- захищене зберігання даних – локально або в хмарі з бекапами;

- можливість розширення – за рахунок модульної архітектури Laravel.

Таким чином, запропоноване рішення заповнює нішу між особистими фінансовими трекерами та складними бухгалтерськими системами, пропонуючи оптимальний баланс функціональності, простоти та адаптованості до українських реалій.

1.5 Постановка завдання

1.5.1 Загальна характеристика задачі

Фізичні особи-підприємці (ФОП) в Україні щодня стикаються з необхідністю ведення обліку доходів, витрат, податкових зобов’язань, складання фінансової звітності та планування бюджету. Більшість таких процесів здійснюється вручну або з використанням загальних офісних програм (наприклад, Excel), що призводить до підвищеної ймовірності помилок, дублювання інформації, ускладнення аналізу та відсутності єдиної структури обліку.

Крім того, чинне законодавство України зобов’язує ФОП своєчасно подавати звітність та сплачувати податки [2]. У зв’язку з цим виникає потреба в сучасному програмному рішенні, яке забезпечить зручне, структуроване та автоматизоване ведення фінансової діяльності ФОП із підтримкою звітності, візуального аналізу та рольового доступу.

1.5.2 Мета проекту

Метою проекту є створення веб-орієнтованої інформаційної системи управління фінансами для фізичної особи-підприємця, яка дозволить:

- вести облік доходів та витрат із категоризацією;
- автоматично формувати звіти (у тому числі податкові);
- візуалізувати фінансову інформацію за допомогою графіків;
- планувати майбутні витрати та прибутки;

- забезпечити зручну авторизацію та рольовий доступ (підприємець, бухгалтер);
- експортувати фінансові дані для архівування або подання у державні органи.

1.5.3 Функціональні вимоги

Функціональні вимоги

Таблиця 1.2

№	Назва функції	Опис функціоналу
1	Авторизація користувачів	Доступ до системи за логіном і паролем. Ролі: підприємець, бухгалтер.
2	Облік доходів	Внесення доходів із зазначенням дати, суми, категорії, джерела.
3	Облік витрат	Фіксація витрат з деталізацією (категорія, опис, дата, сума).
4	Фінансове планування	Можливість задавати очікувані доходи/витрати, бюджети на періоди.
5	Автоматична генерація звітів	Побудова звітів за періодами, категоріями, податковими групами тощо.
6	Графічна аналітика	Відображення фінансової статистики у вигляді графіків і діаграм.
7	Експорт/імпорт даних	Можливість експортувати дані у CSV/XLSX, а також імпортувати з інших систем.
8	Керування категоріями	Створення, редагування, видалення категорій доходів/витрат.
9	Користувацькі ролі	Розмежування прав доступу (перегляд, редагування, лише перегляд звітів тощо).

1.5.4 Нефункціональні вимоги

Нефункціональні вимоги

Таблиця 1.3

№	Вимога	Опис
1	Зручність інтерфейсу	Сучасний інтерфейс з адаптивною верстkою, зрозуміле розміщення елементів керування.
2	Продуктивність	Система повинна реагувати на запити у межах 1-2 секунд при середньому обсязі даних.
3	Безпека	Авторизація, контроль доступу, захист від SQL-ін’єкцій, шифрування паролів.
4	Розширюваність	Можливість додавати нові модулі (наприклад, інтеграцію з банком або API ДПС).
5	Сумісність	Працює в актуальних версіях браузерів: Chrome, Firefox, Safari, Edge.
6	Надійність	Обробка помилок, резервне копіювання бази даних.
7	Мова інтерфейсу	Повна підтримка української мови.

1.5.5 Вимоги до середовища розробки

Виходячи із усіх опрацьованих даних маємо наступні вимоги до середовища розробки:

- мова програмування: PHP 8.4 (фреймворк Laravel 12.x);
- фреймворк клієнтської частини: Vue.js 3;
- СУБД: PostgreSQL;
- інструмент візуалізації: Chart.js;
- середовище розробки: VS Code / PHPStorm;
- система контролю версій: Git (GitHub);
- інструмент розгортання: Docker.

1.5.6 Обмеження

Функціонал обмежений лише поточними потребами обліку ФОП 1-3 групи без ПДВ.

1.6 Висновки до розділу 1

У результаті проведеного системного аналізу предметної області було встановлено актуальність створення автоматизованої інформаційної системи управління фінансами для фізичних осіб-підприємців (ФОП). Проаналізовано характерні особливості фінансової діяльності ФОП, визначено основні функціональні завдання, які мають бути реалізовані в програмному забезпеченні: облік доходів і витрат, формування звітності, податкові розрахунки, бюджетування та ведення клієнтської бази.

Виявлено ключові проблеми, з якими стикаються підприємці в процесі фінансового обліку, а саме: відсутність централізованого інструменту, фрагментованість даних, складність ведення звітності та високі часові витрати. З огляду на це, впровадження автоматизованої системи є ефективним рішенням, що дозволить оптимізувати процеси, мінімізувати помилки та забезпечити зручний доступ до аналітичної інформації.

На основі аналізу сформульовано функціональні, нефункціональні та технічні вимоги до майбутнього програмного продукту. Вони охоплюють як базову функціональність (реєстрація користувачів, облік операцій, звітність), так і важливі аспекти якості (продуктивність, безпека, масштабованість).

Для формалізації уявлення про систему було виконано моделювання предметної області із застосуванням UML-діаграм: діаграми прецедентів, діаграми послідовності та діаграми активності. Це дозволило наочно представити сценарії взаємодії користувачів із системою, структуру бізнес-процесів та логіку виконання функцій.

Окрему увагу було приділено аналізу інформаційних джерел та огляду існуючих рішень, що допомогло врахувати сучасні технологічні підходи та найкращі практики у сфері розробки веб-застосунків для фінансового менеджменту.

Таким чином, розділ створює необхідне підґрунтя для подальшого етапу розробки – проектування архітектури інформаційної системи та вибору технологій її реалізації.

2 Проектування інформаційного та програмного забезпечення

2.1 Логічна модель даних у вигляді ER-діаграми

ER-моделювання (модель “сутність-зв’язок”) є одним з основних етапів проектування бази даних інформаційної системи. Такий підхід дозволяє формалізувати структуру даних, які мають бути збережені в системі, у вигляді логічної моделі, що складається з сутностей, їхніх атрибутів та зв’язків між ними. Це дозволяє розробнику системи сформувати концептуальне бачення архітектури даних ще до початку реалізації фізичної бази даних.

У контексті інформаційної системи управління фінансами фізичної особи-підприємця основними сутностями є:

- Користувач – центральна сутність системи, що представляє підприємця або іншого користувача. Має такі атрибути: ID_Користувач, ПІБ, Пошта, Пароль, Роль, Статус;
- Категорія – довідникова сутність, яка дозволяє класифікувати фінансові операції за типом (дохід або витрата). Атрибути: ID_Категорія, Назва, Тип, ID_Користувач (FK);
- Фінансова_операція – відображає транзакції, що здійснюються користувачем. Атрибути: ID_Фінансова_операція, Тип, Сума, Дата, Опис, зовнішні ключі: ID_Користувач (FK), ID_Категорія (FK);
- Звіт – сутність, що зберігає результати аналітики або підсумкової інформації. Атрибути: ID_Звіт, Тип, Дата_створення, Фільтри, Вміст, ID_Користувач (FK);
- Журнал_подій – призначений для реєстрації дій користувачів у системі. Атрибути: ID_Журнал_подій, Подія, Дата, Опис, ID_Користувач (FK);

- Доступ – визначає рівні прав доступу. Атрибути: ID_Доступ, Права, Дата_надання, Дата_оновлення;
- Користувач_Доступ – зв'язувальна таблиця, що реалізує зв’язок N:M між Користувачами і рівнями Доступу. Містить зовнішні ключі: ID_Користувач (FK), ID_Доступ (FK).

Всі сутності пов’язані між собою відповідно до логіки функціонування системи:

- один Користувач може мати багато Фінансових операцій, Звітів, записів у Журналі подій та Категорій (зв’язки 1:N).
- кожна Фінансова операція належить одному Користувачу і одній Категорії (зв’язок M:1).
- категорії можуть використовуватись у багатьох Фінансових операціях (зв’язок 1:N).
- доступ може бути призначено декільком Користувачам (через зв’язувальну таблицю Користувач_Доступ).

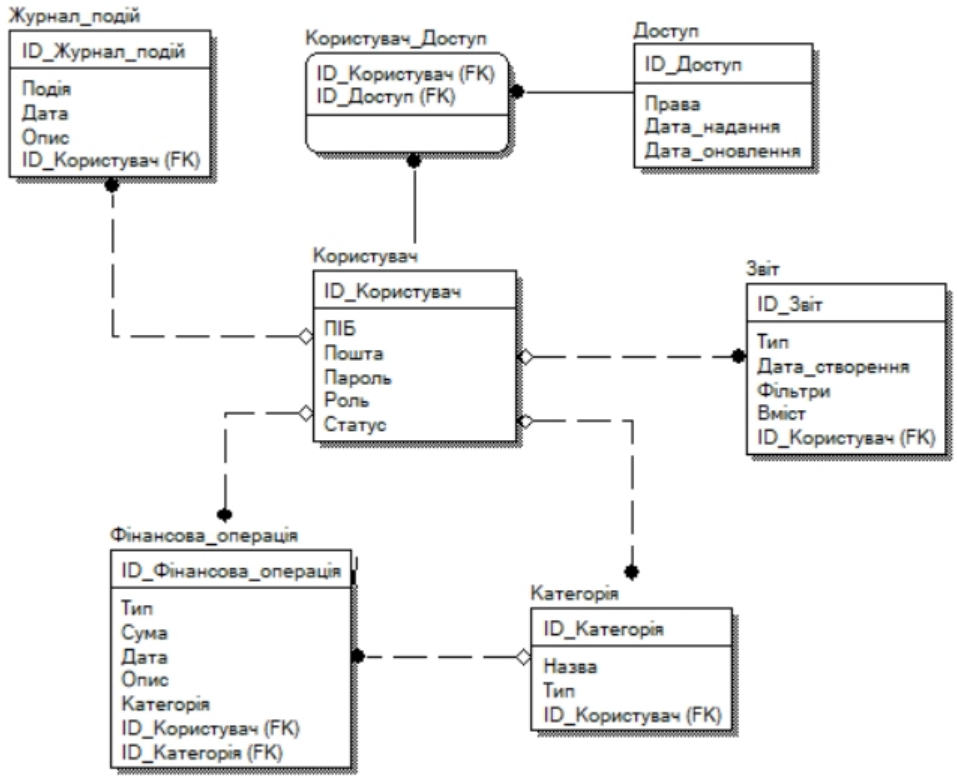

Рисунок 4 – ER-діаграма

На рисунку 4 представлено логічну модель даних системи у вигляді ER-діаграми. Кожна сутність має унікальний ідентифікатор (первинний ключ) та набір атрибутів, що характеризують її. Зв’язки реалізуються через зовнішні ключі, що забезпечують логічну цілісність даних.

Усі відношення в моделі відповідають вимогам третьої нормальної форми (ЗНФ), оскільки:

- кожна таблиця має первинний ключ;
- усі атрибути функціонально залежать від усього первинного ключа (відсутні часткові залежності);
- відсутні транзитивні залежності між неключовими атрибутами.

Таким чином, побудована ER-модель є логічно узгодженою, оптимізованою для подальшої реалізації в реляційній СУБД та здатна забезпечити ефективне зберігання й обробку фінансової інформації.

2.2 Діаграма класів та кооперацій

Для побудови якісної інформаційної системи недостатньо лише зібрати вимоги до її функціонування. Надзвичайно важливо також спроектувати архітектуру програмного забезпечення: визначити основні класи (об’єкти), їхні атрибути, методи та взаємозв’язки. UML-діаграми дозволяють зробити це наочно, зручно та структуровано.

У цьому підпункті буде розглянуто дві важливі моделі: діаграму класів, яка описує структуру системи, та діаграми кооперацій, що демонструють, як об’єкти взаємодіють під час виконання певної задачі.

2.2.1 Діаграма класів

На діаграмі класів, що зображена на рис. 5, присутні класи зі своїми атрибутами, які поєднані асоціативним зв’язком, що представляє структурні відносини між об’єктами (+1, +1..*, +0..*):

- Журнал подій – записує події всіх користувачів у системі;
- Адміністратор – керує користувачами та довідниками системи;

- ФОП – основний користувач системи, веде фінансовий облік;
- Бухгалтер – користувач, що надає послуги ведення бухгалтерії основним користувачам системи;
- Доступ – представляє права доступу бухгалтера для ведення обліку фінансів певного ФОПа;
- Фінансова операція – операція, що фіксує дохід або витрату;
- Категорія – визначає категорію, до якої належить фінансова операція;
- Звіт – згенерований звіт за фінансовими операціями або податковий звіт.

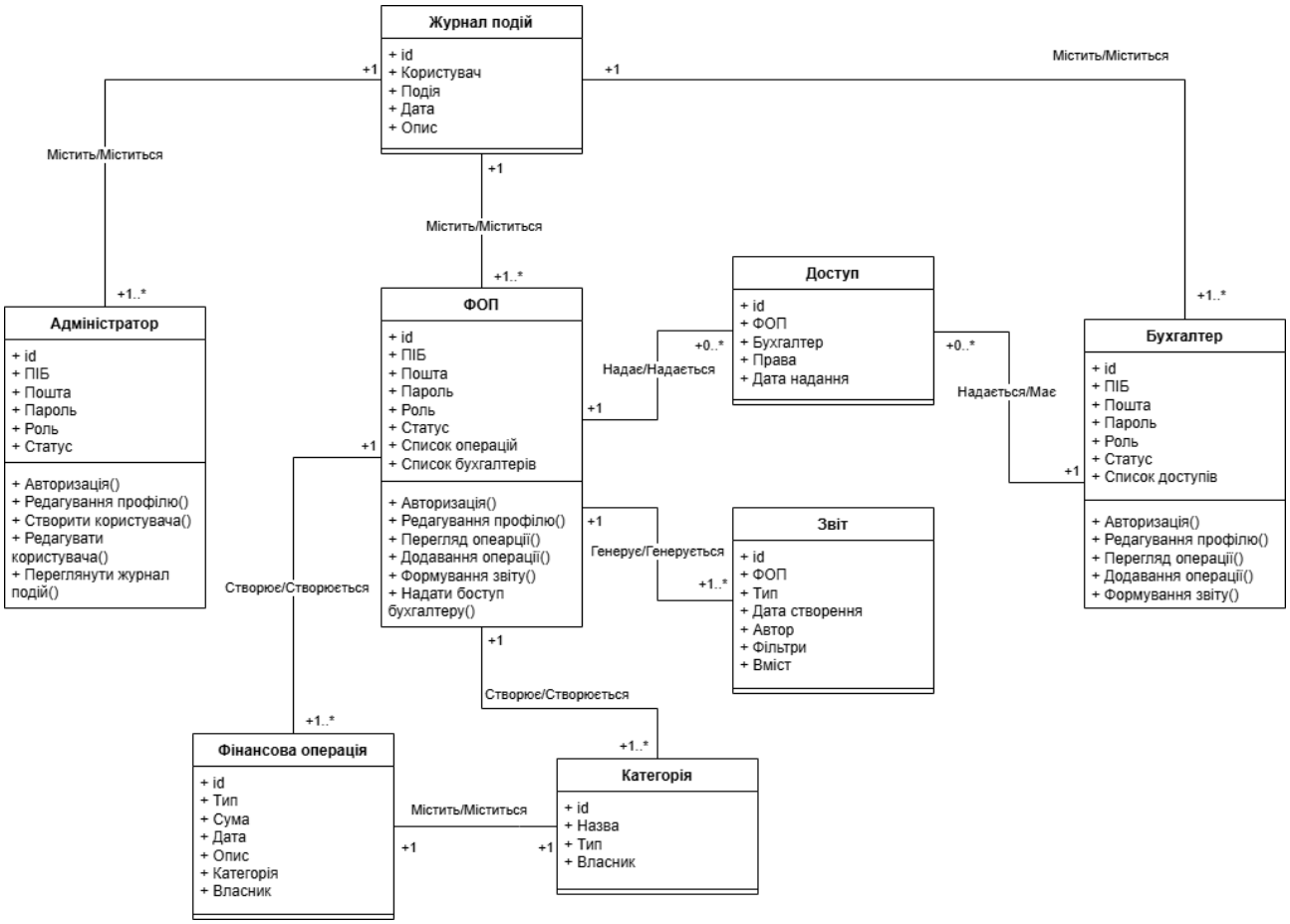

Рисунок 5 – Діаграма класів

2.2.2 Діаграма кооперацій

Діаграми кооперацій ілюструють сценарії взаємодії між об’єктами. Вони демонструють, як різні класи системи співпрацюють для досягнення певної функціональності. В розроблюваній системі розглянуто три ключові сценарії:

1. Створення та звітування по фінансовим операціям (див. рис. 6) – процес створення ФОПом фінансової операції та створення звіту по фінансовим операціям. Класи: ФОП, Фінансова операція, Категорія, Звіт.

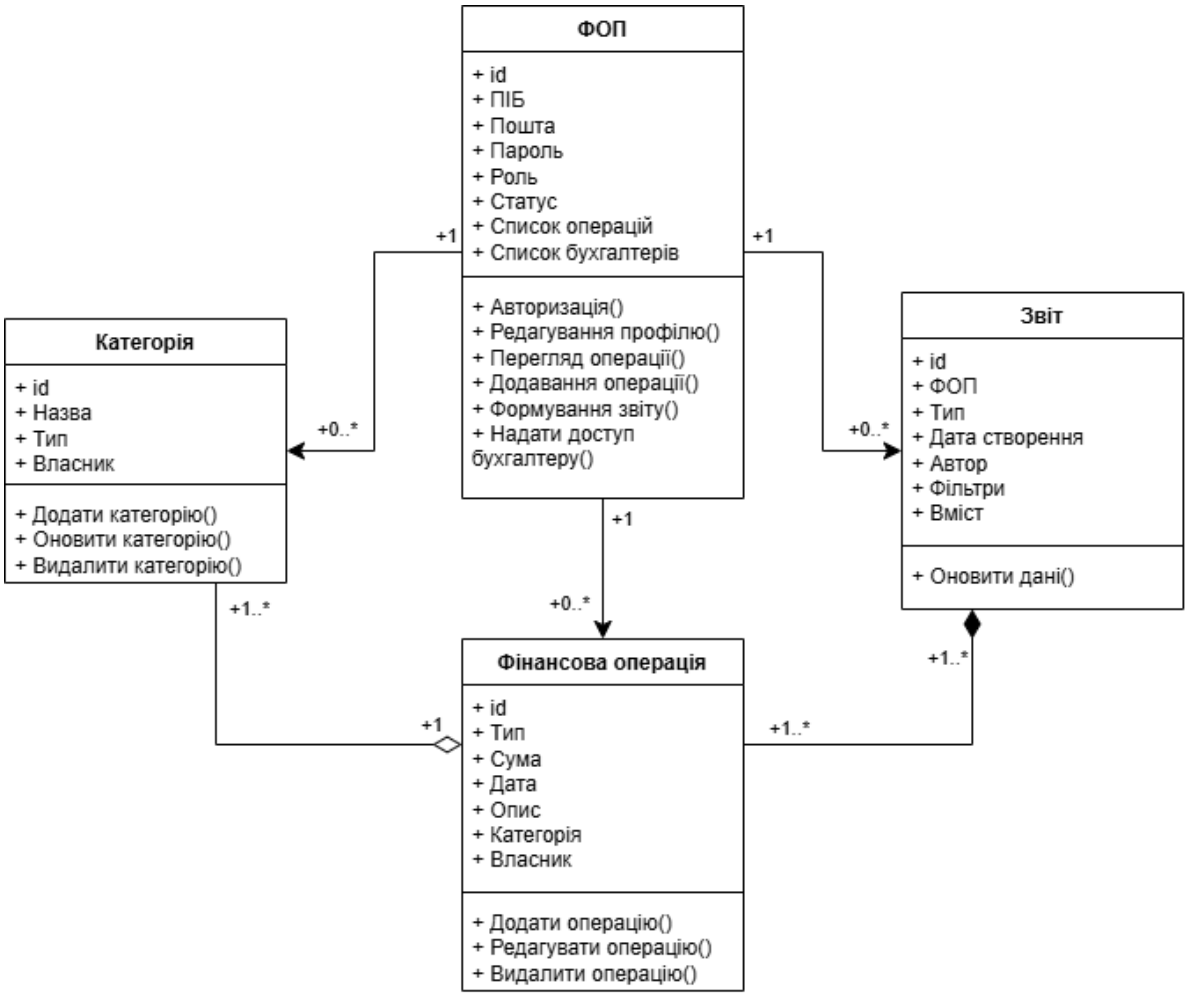

Рисунок 6 – Кооперація класів: створення та звітування по фінансовим операціям

2. Надання ФОПом доступу бухгалтеру (див. рис. 7) – процес надання ФОПом доступу бухгалтеру на виконання певних дій. Класи: ФОП, Доступ, Бухгалтер.

Рисунок 7 – Кооперація класів: надання ФОПом доступу бухгалтеру

3. Запис дій користувачів в журнал подій (див. рис. 8) – процес запису дій користувачів в журнал подій. Класи: Журнал подій, Користувач, Адміністратор, Користувач фінансової системи, ФОП, Бухгалтер.

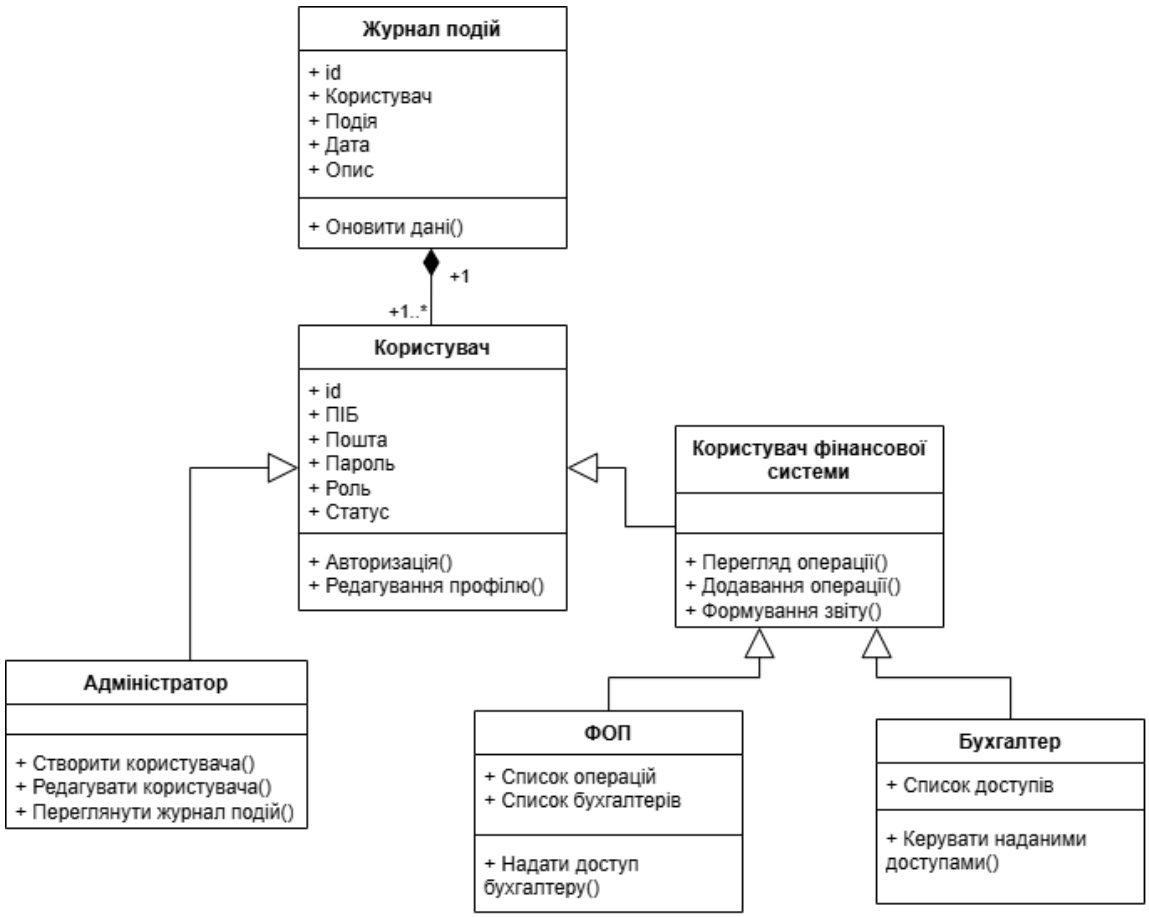

Рисунок 8 – Кооперація класів: запис дій користувачів в журнал подій

2.3 Діаграма пакетів

У процесі проектування складних інформаційних систем важливо не лише розуміти взаємодію окремих класів, а й мати загальне уявлення про логічну структуру всієї програми. Саме для цього використовуються діаграми пакетів (package diagrams), які є частиною стандарту UML і дозволяють візуально відобразити модульну архітектуру програмного продукту.

Діаграма пакетів ілюструє групування класів у логічні пакети, а також залежності між ними. Це дає змогу:

- сформувати чітке уявлення про ієрархію підсистем;
- виділити функціональні області (UI, бізнес-логіка, доступ до даних тощо);
- спростити підтримку та масштабування системи;
- визначити межі відповідальності кожного модуля;
- забезпечити слабке зв’язування між компонентами (loose coupling).

2.3.1 Загальна структура

Для розробленої системи побудовано діаграму, яка складається з двох основних модулів:

- UI (User Interface) – клієнтська частина, відповідальна за взаємодію з користувачем;
- API – серверна частина, відповідальна за бізнес-логіку, обробку запитів та збереження даних.

Кожен із цих модулів, у свою чергу, поділений на підпакети:

- UI:
  - auth — автентифікація та авторизація;
  - admin — модуль адміністрування;
  - finance — модуль фінансових операцій;
  - reporting — звітність;
- API:

- http — обробка HTTP-запитів;
- core — бізнес-логіка;
- database — доступ до сховища даних.

2.3.2 Опис компонентів

1. Пакет UI

Відповідає за інтерфейс користувача, де кожен підпакет виконує окрему функцію:

- auth – реалізує вхід, реєстрацію та перевірку повноважень;
- admin – доступний після авторизації; забезпечує адміністрування системи (керування користувачами, налаштування);
- finance – основний функціонал системи; доступ до фінансових транзакцій після авторизації;
- reporting – формування звітів на основі даних з finance.

Залежності:

- admin, finance → auth – для перевірки прав доступу;
- reporting → finance – для використання фінансових даних.

2. Пакет API

Це серверна частина системи, яка приймає запити з UI та взаємодіє з базою даних:

- http – контролери, що приймають запити з інтерфейсу та передають їх у бізнес-логіку;
- core – логіка обробки даних, реалізація основних алгоритмів;
- database – модуль взаємодії з базою даних.

Залежності:

- http → core – контролери викликають бізнес-методи;
- core → database – ядро маніпулює збереженими даними.

Також ui → api – інтерфейс ініціює запити до API.

2.3.3 Типи залежностей

У діаграмі використано такі типи залежностей:

- Імпорт – один модуль імпортує функції іншого (напр., http → core);
- Доступ – для виконання дій модуль звертається до іншого (напр., admin → auth);
- Злиття – не представлено, проте можливо для майбутніх об’єднань (наприклад, reporting може бути інтегрований у finance).

2.3.4 Візуалізація

На рисунку нижче зображено відповідну діаграму пакетів, яка демонструє логічну структуру системи та зв’язки між її компонентами:

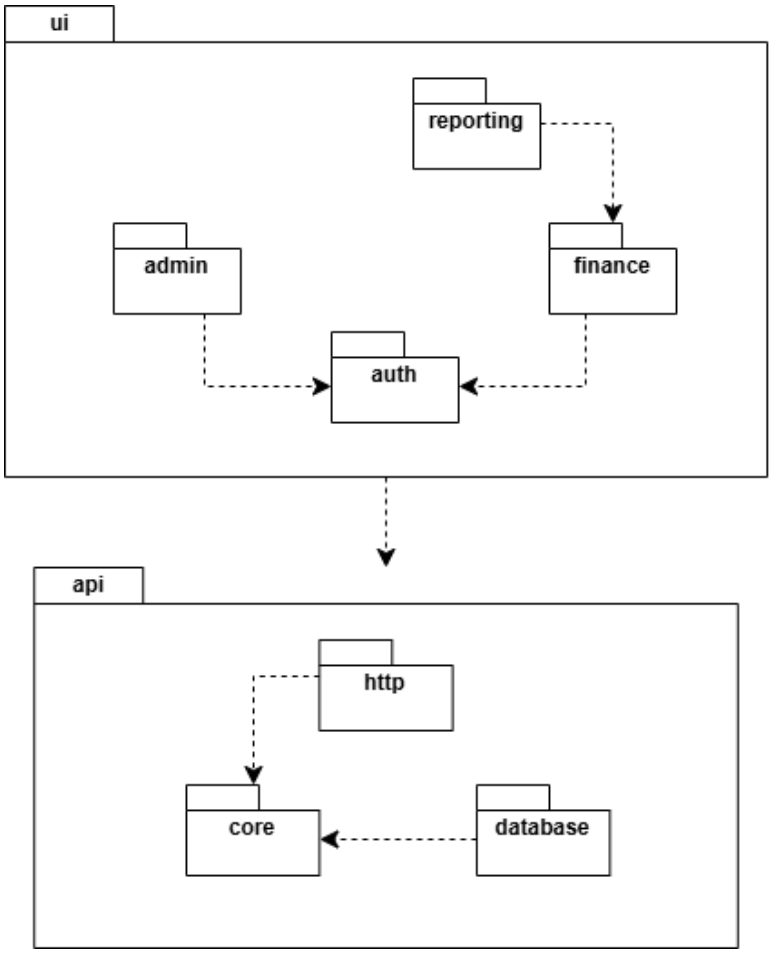

Рисунок 9 – Діаграма пакетів

2.4 Діаграма компонентів

Діаграма компонентів, побудована для розроблюваного програмного забезпечення, має наступний вигляд:

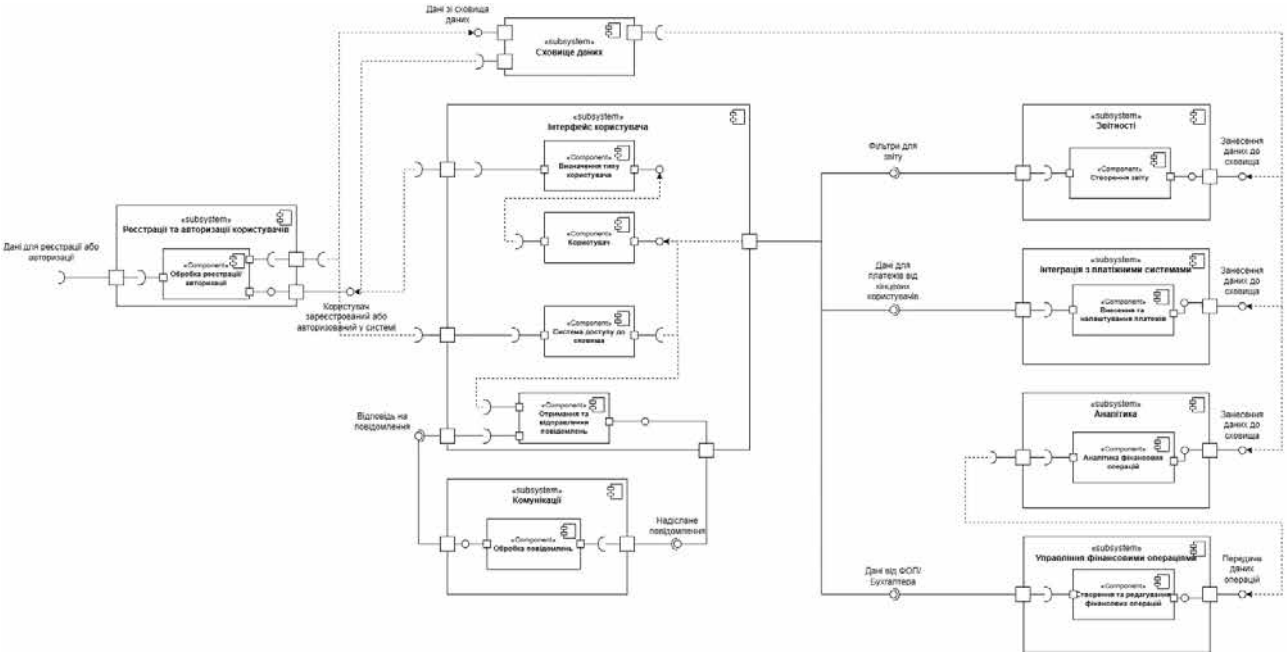

Рисунок 10 – Діаграма компонентів

2.4.1 Структура та взаємодія компонентів

У структурі інформаційної системи управління фінансами для фізичних осіб-підприємців (ФОП) важливою є чітка модульність, яка дозволяє ефективно розділити функціональні обов’язки між окремими підсистемами. Діаграма компонентів, зображена на рис. 10, візуалізує ключові компоненти системи та їхню взаємодію:

- Інтерфейс користувача – організовує взаємодію з користувачем відповідно до ролі, забезпечує формування запитів до аналітики, звітності, інтеграційних сервісів, фінансових операцій і повідомлень;
- Підсистема реєстрації та авторизації – відповідає за автентифікацію користувача. Вона обробляє вхідні дані (логін або реєстрація), після чого передає авторизованого користувача до інтерфейсу;
- Підсистема комунікацій – забезпечує надсилання та обробку повідомлень (запитів і відповідей). Отримує запити з інтерфейсу користувача та відповідає за надсилання відповідей користувачам;

- Підсистема звітності – формує документи на основі користувацьких запитів. Дані надходять з інтерфейсу, після чого результати записуються до сховища даних;
- Підсистема інтеграції з платформами – забезпечує обмін даними з іншими інформаційними системами, наприклад державними або фінансовими платформами;
- Аналітика фінансових операцій – оперативно аналізує дії користувачів після додавання фінансових даних і формує відповідні аналітичні звіти;
- Управління фінансовими операціями – відповідає за створення, редагування та збереження фінансових операцій. Приймає дані безпосередньо від ФОП або бухгалтера;
- Сховище даних – єдиний центр збереження даних, який взаємодіє з усіма основними підсистемами: аналітикою, звітністю, фінансовими операціями, повідомленнями, інтеграцією.

2.4.2 Аналіз діаграми компонентів

Аналіз діаграми показує такі ключові особливості архітектури:

- усі запити користувача маршрутизуються через інтерфейс користувача, який виступає центральною точкою взаємодії;
- система авторизації функціонує як вхідний шлюз, після чого передає контроль інтерфейсу користувача;
- кожна функціональна підсистема (звіти, аналітика, повідомлення, інтеграція) є незалежною і може масштабуватися окремо;
- база даних не є доступною напряму для користувача – це підвищує безпеку та відповідає принципам розділення обов’язків;
- комунікаційна підсистема є проміжною ланкою для обміну інформацією між системою та користувачем.

2.4.4 Переваги архітектурного підходу

Обраний архітектурний підхід має наступні переваги:

- модульність: кожен компонент виконує чітко визначену функцію, що спрощує розробку, супровід і тестування;
- розширюваність: архітектура дозволяє безболісно додавати нові функції або інтегрувати зовнішні сервіси;
- безпека: обмеження прямого доступу до бази даних лише через внутрішні підсистеми знижує ризики витоку або пошкодження інформації;
- зручність супроводу: помилка в одному модулі не впливає на стабільність усієї системи, що значно підвищує її надійність.

2.5 Висновки до розділу 2

У розділі проведено ґрунтовне проектування як інформаційної, так і програмної складових системи управління фінансами фізичної особи-підприємця. Побудова логічної моделі даних у вигляді ER-діаграми дозволила визначити структуру бази даних, сформулювати основні сутності, їх атрибути та взаємозв’язки. Ретельна нормалізація до третьої нормальної форми забезпечила узгодженість, цілісність і відсутність надлишковості даних, що є критично важливим для фінансових систем.

Проектування програмного забезпечення за допомогою діаграм класів та кооперацій дозволило візуалізувати структуру об’єктно-орієнтованої моделі та взаємодію між ключовими компонентами системи. Це забезпечує чітке розуміння архітектури і полегшує подальшу реалізацію та розширення функціональності. Особливу увагу приділено ролям користувачів (ФОП, бухгалтер, адміністратор) і специфіці їх взаємодії.

Діаграма пакетів надала уявлення про логічну структуру програмного продукту на рівні модулів. Розмежування інтерфейсної частини (UI) і серверної частини (API) підвищує масштабованість та підтримуваність системи, а чітко визначені залежності між пакетами сприяють реалізації слабкого зв’язування.

Завершальним етапом є побудова діаграми компонентів, яка узагальнює модульну архітектуру системи та вказує на інтеграцію окремих підсистем

(реєстрація, авторизація, фінансові операції, аналітика, повідомлення тощо) в єдину цілісну систему.

Таким чином, проведене проектування створює надійну основу для ефективної реалізації, тестування та подальшого розвитку інформаційної системи управління фінансами ФОП, що відповідає сучасним вимогам до якості, безпеки та масштабованості програмного забезпечення.

3 Розробка інформаційного та програмного забезпечення

3.1 Система управління інформаційною базою

У процесі розробки програмного забезпечення інформаційної системи управління фінансами для фізичної особи-підприємця (ФОП) особливу увагу приділено вибору системи керування базами даних (СУБД), оскільки саме вона забезпечує збереження, обробку та доступ до фінансових даних користувачів. Така система повинна гарантувати цілісність інформації, підтримку транзакцій, масштабованість і високу продуктивність під час виконання запитів.

Серед сучасних підходів до організації інформаційної бази виділяють дві основні парадигми: реляційні та нереляційні (NoSQL) системи зберігання даних. Для інформаційної системи, що оперує структурованими фінансовими записами (транзакції, категорії витрат, податкові звіти, реквізити клієнтів тощо), найбільш доцільним є використання реляційної СУБД. Такий підхід дозволяє створити чітку структуру таблиць з визначеними зв’язками, забезпечити нормалізацію даних та виконання складних запитів мовою SQL.

У якості СУБД для даного проекту було обрано PostgreSQL – одну з найпотужніших та найбільш стабільних об’єктно-реляційних систем з відкритим кодом. PostgreSQL поєднує переваги класичної реляційної моделі з можливостями об’єктного програмування [7], що дозволяє використовувати такі функції, як:

- підтримка складних типів даних (JSONB, масиви, UUID);
- створення користувацьких функцій та тригерів;
- забезпечення транзакційної цілісності (повна відповідність ACID-властивостям);

- ефективна обробка паралельних запитів завдяки внутрішньому механізму планування.

Оскільки інформаційна система для ФОП оперує не лише табличними записами, а й динамічними об’єктами, такими як бюджети, підкатегорії витрат, нотатки та документи у вільному форматі, PostgreSQL також дозволяє зберігати напівструктуровану інформацію завдяки повноцінній підтримці формату JSONB, що значно підвищує гнучкість зберігання без шкоди для продуктивності [7].

Крім того, PostgreSQL забезпечує:

- масштабованість – можливість розширення на кластерні системи при збільшенні кількості користувачів;
- безпеку – налаштування прав доступу до окремих таблиць або навіть полів;
- розширюваність – підтримка сторонніх модулів та розширень, таких як pg_stat_statements, PostGIS, uuid-ossp тощо;
- кросплатформність – підтримка всіх основних ОС (Linux, Windows, macOS), що дозволяє легко переносити систему на інші середовища.

Також PostgreSQL ефективно інтегрується з фреймворками, що використовуються в даному проекті, зокрема Laravel 12.x, який має вбудовану підтримку PostgreSQL через ORM Eloquent та дозволяє легко реалізовувати CRUD-операції, складні фільтри, агрегацію та перевірки зв’язків між таблицями [6][7][8][16].

Таким чином, обрання PostgreSQL як основної СУБД для проекту обумовлене необхідністю обробки фінансових даних з високою точністю, забезпеченням транзакційної безпеки, підтримкою складної бізнес-логіки та потенційного масштабування для зростання користувацької бази. Це рішення є оптимальним з точки зору надійності, відкритості, продуктивності та довгострокового супроводу системи.

3.2 Розробка інформаційної бази

Розробка інформаційної бази є ключовим етапом створення інформаційної системи управління фінансами для фізичної особи-підприємця. У рамках проекту реалізація бази даних здійснюється з використанням принципу Code First у фреймворку Laravel 12.x, що дозволяє спочатку створити PHP-класи-сутності, а потім – автоматично згенерувати відповідну структуру таблиць в СУБД (в нашому випадку – MySQL або PostgreSQL).

3.2.1 Структура інформаційної бази

Базу даних спроектовано відповідно до нормалізованої реляційної моделі, що дозволяє зберігати дані цілісно, уникати дублювання, забезпечити швидкий доступ до записів та підтримувати референційну цілісність. Основу бази складають наступні сутності:

- User – користувач системи (ФОП, співробітник, адміністратор);
- EventLog – журнал подій користувачів (логування входів, дій, змін тощо);
- FinanceOperation – фінансова операція (доходи/витрати, сума, дата, опис);
- Category – категорія операції (наприклад, «Оренда», «Продаж», «Закупівлі»);
- Report – звіт, згенерований на основі операцій (щомісячний, податковий тощо);
- Access – тип доступу (наприклад, «перегляд», «редагування», «адміністрування»);
- UserAccess – зв’язок між користувачем та доступами.

3.2.2 Визначення моделей (Laravel Eloquent Models)

У відповідності до підходу Code First, для кожної сутності створено відповідну модель Laravel (Eloquent Model). Наприклад, модель User містить базові поля, такі як:

```
public function financeOperations(): HasMany
{
    return $this->hasMany(FinanceOperation::class);
}
```

Подібним чином, у моделі FinanceOperation визначено зв’язки з категоріями та користувачами:

```
public function user(): BelongsTo
{
    return $this->belongsTo(User::class);
}

public function category(): BelongsTo
{
    return $this->belongsTo(Category::class);
}
```

3.2.3 Міграції та створення таблиць

Для кожної моделі створено Laravel міграцію – PHP-файл, який описує схему таблиці, обмеження (типи даних, первинні/зовнішні ключі, обов’язковість полів, унікальність). Наприклад, структура таблиці finance_operations у файлі міграції виглядає так:

```
Schema::create('finance_operations', function (Blueprint $table) {
    $table->id();
    $table->foreignId('user_id')->constrained()->onDelete('cascade');
    $table->foreignId('category_id')->constrained()->onDelete('restrict');
    $table->decimal('amount', 12, 2);
    $table->string('description')->nullable();
    $table->date('operation_date');
    $table->timestamps();
});
```

Аналогічно створюються таблиці для інших моделей. Ключовою перевагою такого підходу є те, що вся логіка зберігається у версіонованих скриптах, що дозволяє зручно керувати змінами в структурі БД.

3.2.4 Взаємозв’язки між таблицями

Завдяки підтримці зовнішніх ключів у міграціях, забезпечується логічна цілісність даних. Наприклад:

- кожна фінансова операція пов'язана з певним користувачем (user_id);
- операція також належить до певної категорії (category_id);
- журнал подій (event_logs) зберігає зв’язок з користувачем і дією;
- таблиця user_access реалізує зв’язок багато-до-багатьох між users та accesses.

3.2.5 Контроль та ініціалізація бази

Для початкового наповнення довідкових таблиць (categories, accesses) використовуються Laravel seeders. Вони дозволяють швидко згенерувати тестові та стандартні записи, наприклад:

```
DB::table('categories')->insert([
   ['name' => 'Оренда'],
   ['name' => 'Закупівлі'],
   ['name' => 'Доходи від продажів'],
]);
```

3.2.6 Підсумкова схема БД

На основі реалізованих моделей автоматично генерується фізична структура бази даних. Типові міжтабличні зв’язки включають:

- users → finance_operations (один-до-багатьох);
- finance_operations → categories (багато-до-одного);
- users ↔ accesses через user_access (багато-до-багатьох);
- users → event_logs (один-до-багатьох);
- users → reports (один-до-багатьох).

Діаграма зв’язків може бути згенерована через Laravel-плагіни або сторонні засоби (наприклад, MySQL Workbench або JetBrains DataGrip).

3.3 Вибір інструментарію для створення прикладного програмного забезпечення

У процесі створення інформаційної системи управління фінансами для фізичних осіб-підприємців важливим етапом стала розробка прикладного програмного забезпечення, що охоплює серверну логіку, клієнтський інтерфейс, механізми взаємодії з базою даних та API. Для забезпечення масштабованості, продуктивності, зручності в обслуговуванні та швидкості розробки було обрано сучасний і перевірений стек веб-технологій: Laravel 12.x для бекенду та Vue.js для фронтенду.

3.3.1 Обґрунтування вибору Laravel 12.x

Laravel – це популярний PHP-фреймворк, який надає широкий спектр готових рішень для побудови надійних і масштабованих веб-додатків. У нашому випадку Laravel 12.x виконує роль серверної частини, реалізуючи обробку запитів, бізнес-логіку, доступ до бази даних, аутентифікацію, авторизацію та API-комунікацію з клієнтом.

Основні причини вибору Laravel:

- модульна архітектура MVC (Model-View-Controller), що забезпечує розділення відповідальностей, зручність тестування та розширюваність;
- вбудована ORM Eloquent, яка дозволяє працювати з базою даних у вигляді об'єктів, зменшуючи кількість SQL-запитів і спрощуючи взаємодію з моделями;
- підтримка RESTful API та Sanctum для побудови безпечної аутентифікації користувачів та інтеграції з клієнтською частиною;
- засоби міграцій і сидів для створення структури бази даних і її початкового наповнення;

- потужна екосистема та документація, що прискорює процес розробки і дозволяє легко інтегрувати додаткові модулі (наприклад, генерацію PDF-звітів, аналітику, черги обробки тощо);
- підтримка тестування, включаючи Unit та Feature-тести, що критично важливо для надійності фінансової системи.

Laravel обрано як серверний фреймворк, оскільки він поєднує високу продуктивність з простотою розробки, багатою документацією та великою спільнотою.

3.3.2 Обґрунтування вибору Vue.js

Vue.js – це прогресивний JavaScript-фреймворк для створення динамічних користувацьких інтерфейсів [10]. У проекті він використовується для реалізації клієнтської частини системи – панелі управління фінансами, форм звітності, графіків, фільтрації транзакцій, тощо.

Причини вибору Vue.js:

- компонентна архітектура, яка дозволяє створювати багаторазові, незалежні частини інтерфейсу (наприклад, таблиці, діаграми, форми);
- реактивність – зміни у даних миттєво відображаються в інтерфейсі, що критично важливо для фінансових систем, де дані оновлюються часто;
- інтеграція з Laravel – Vue.js легко поєднується з Laravel через API, зокрема за допомогою бібліотек Axios або Fetch API;
- висока продуктивність, особливо на сторінках із великою кількістю динамічних елементів (наприклад, фільтрація за датою, категорією витрат тощо);
- інструменти розробника (Vue Devtools) спрощують налагодження та візуалізацію стану компонентів;
- підтримка сучасного JavaScript (ES6+), а також можливість використання бібліотек для побудови графіків (наприклад, Chart.js або ApexCharts).

Vue.js обрано як фронтенд-фреймворк завдяки його гнучкості, легкості в навчанні та ефективності в побудові інтерактивних фінансових інтерфейсів.

3.3.3 Додаткові інструменти та бібліотеки

Перелік додаткових інструментів та бібліотек:

- MySQL/PostgreSQL – системи керування базами даних, які забезпечують збереження фінансових транзакцій, користувацьких даних і звітів;
- Composer та NPM – менеджери пакетів для бекенду та фронтенду відповідно, що спрощують інсталяцію залежностей;
- Git м система контролю версій, що забезпечує історію змін, роботу в гілках і ефективну командну розробку;
- Laravel Mix / Vite – засоби для компіляції та оптимізації ресурсів (JS, CSS) у межах Laravel-проекту;
- Axios – бібліотека для HTTP-запитів між Vue.js та Laravel API;
- Docker – для ізоляції середовища розробки та спрощеного розгортання [9].

Таким чином, вибраний інструментарій дозволяє ефективно реалізувати повнофункціональну, масштабовану та зручну для користувача інформаційну систему управління фінансами ФОП. Використання Laravel 12.x і Vue.js забезпечує гнучке розділення фронтенду і бекенду, що полегшує підтримку, тестування та подальший розвиток системи.

3.4 Алгоритмізація та програмування програмних модулів

На етапі алгоритмізації та програмування здійснюється розробка ключових бізнес-процесів інформаційної системи, зокрема у вигляді блок-схем та відповідної реалізації у вигляді клієнтських і серверних модулів. У розроблюваній системі передбачено декілька важливих процесів, серед яких особливу увагу приділено реєстрації користувача та додаванню фінансової операції.

3.4.1 Алгоритм реєстрації користувача

Першим кроком взаємодії користувача із системою є процес реєстрації. Блок-схема, яка описує цей алгоритм, зображена на рисунку 11.

На початку користувач вводить необхідні реєстраційні дані: ім’я, адресу електронної пошти та пароль. Система перевіряє валідність даних і унікальність email. Якщо email вже існує в базі даних або дані некоректні, користувач отримує відповідне повідомлення. У разі успішного проходження перевірок, дані у форматі JSON відправляються на сервер, де пароль хешується, а новий користувач зберігається у базі. Після всього надсилається статус реєстрації користувача в системі на фронтенд-частину.

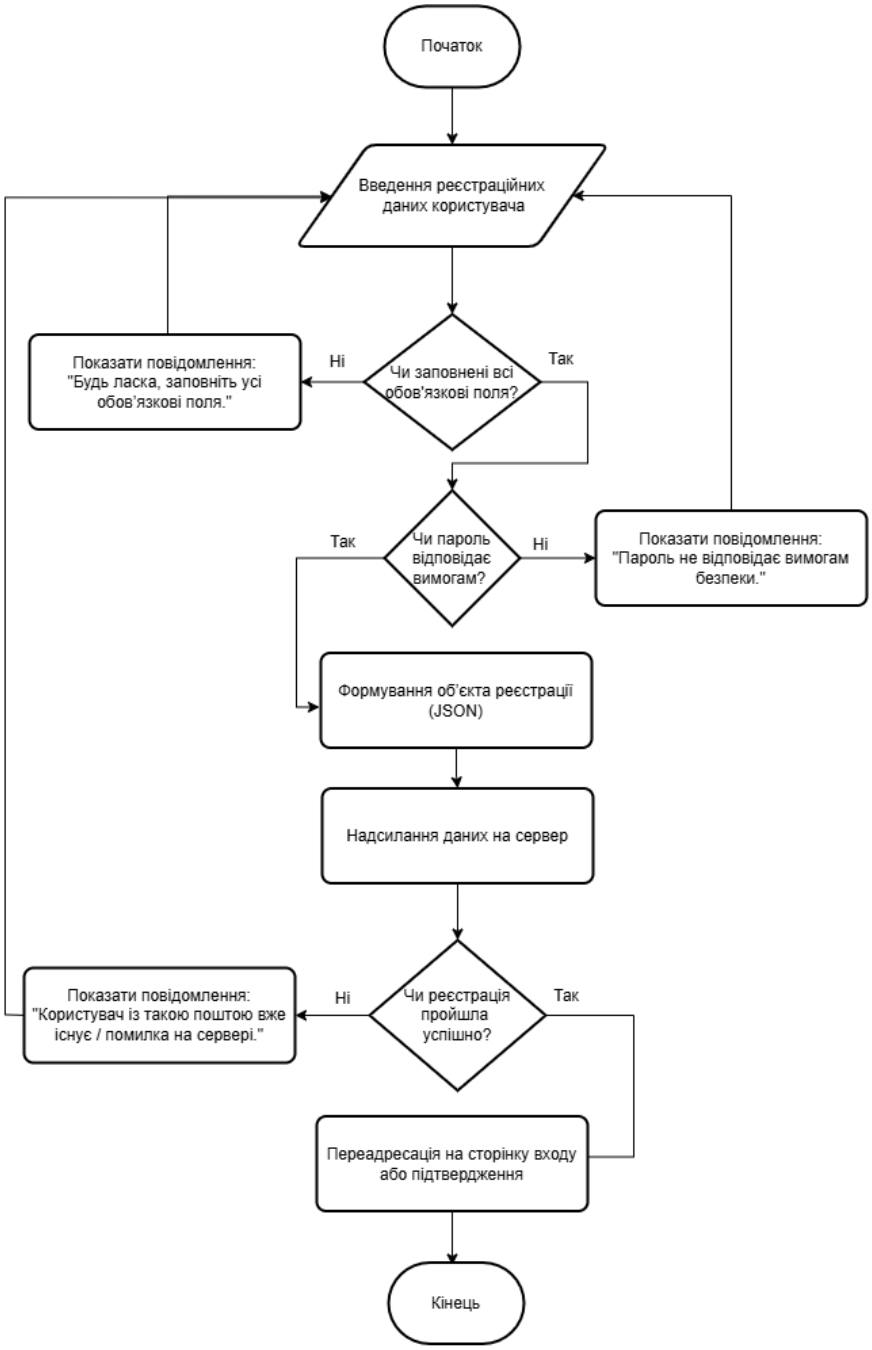

Рисунок 11 – Блок-схема алгоритму реєстрації користувача

Реалізація цього алгоритму на серверній частині, розробленій з використанням Laravel 12.x, виглядає наступним чином:

```
use Illuminate\Support\Facades\Hash;
use Illuminate\Support\Facades\Validator;

Route::post('/register', function (Request $request) {
    $validated = Validator::make($request->all(), [
        'name' => 'required|string|max:255',
        'email' => 'required|string|email|max:255|unique:users',
        'password' => 'required|string|min:8',
    ]);

    if ($validated->fails()) {
        return response()->json(['errors' => $validated->errors()], 400);
    }

    $user = \App\Models\User::create([
        'name' => $request->name,
        'email' => $request->email,
        'password' => Hash::make($request->password),
    ]);

    return response()->json(['message' => 'Реєстрація успішна'], 201);
});
```

На стороні клієнта, створеного з використанням Vue.js, форма реєстрації реалізована через компонент із валідацією введених даних до відправлення на сервер. У разі помилки сервер повертає повідомлення, яке відображається в інтерфейсі.

3.4.2 Алгоритм додавання фінансової операції

Додавання фінансової операції – ключовий процес у системі, що дозволяє фіксувати надходження чи витрати. Блок-схема алгоритму подана на рисунку 12.

Процес починається з ініціації дії користувачем: введення даних для додавання фінансової операції (сума, категорія, тип операції – дохід чи витрата, дата). Після підтвердження форма проходить валідацію, і дані надсилаються на сервер. Якщо дані коректні, операція зберігається в базі даних з прив’язкою до поточного користувача, а на фронтенд-частину надсилається відповідь

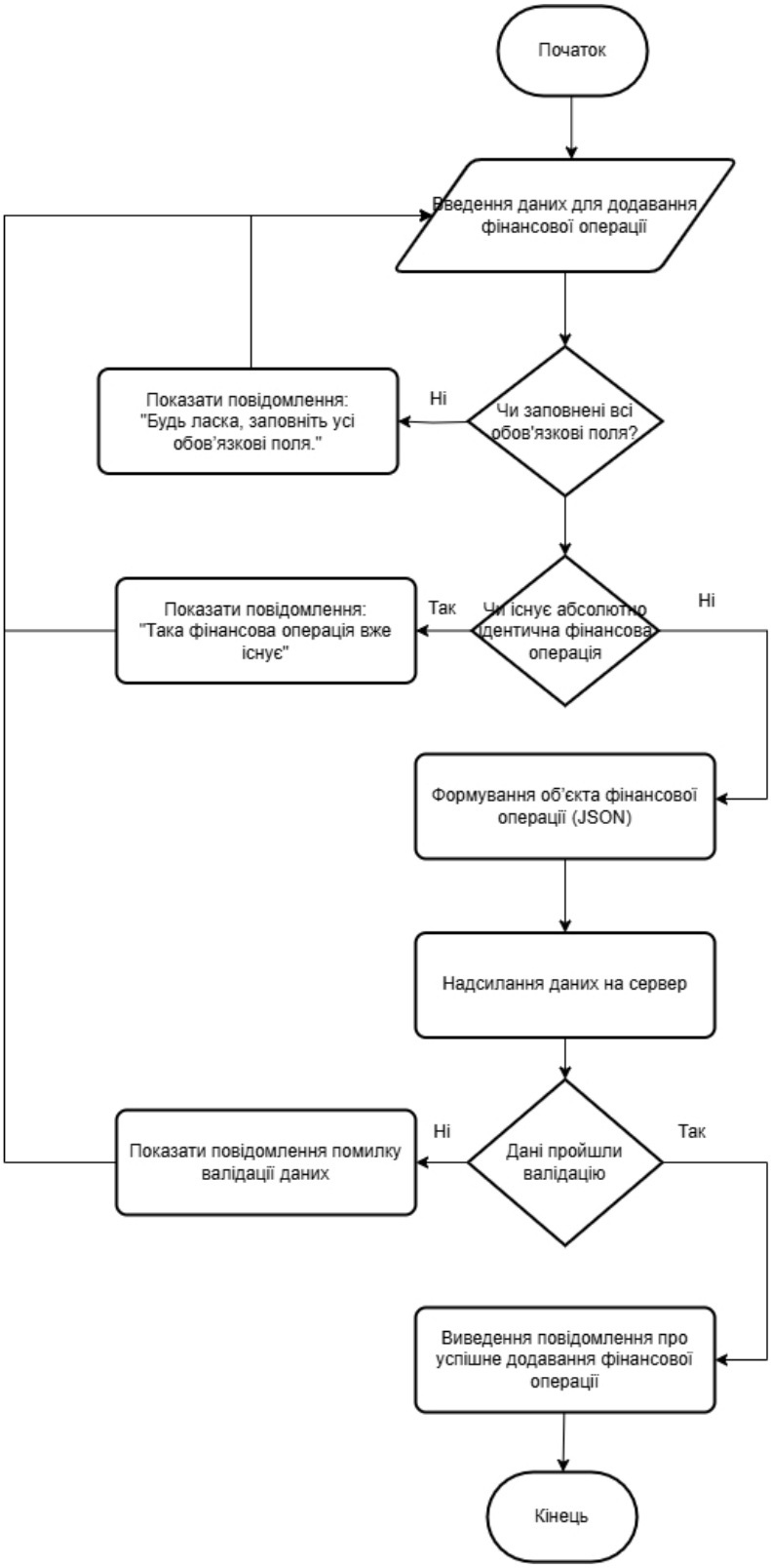

Рисунок 12 – Блок-схема алгоритму додавання фінансової операції

Серверна реалізація цього алгоритму за допомогою Laravel:

```
Route::post('/transactions', function (Request $request) {
    $validated = Validator::make($request->all(), [
```

```
        'user_id' => 'required|exists:users,id',
        'type' => 'required|in:income,expense',
        'amount' => 'required|numeric|min:0.01',
        'category' => 'required|string|max:100',
        'date' => 'required|date',
    ]);

    if ($validated->fails()) {
        return response()->json(['errors' => $validated->errors()], 400);
    }

    \App\Models\Transaction::create($validated->validated());

    return response()->json(['message' => 'Операцію успішно додано'], 201);
});
```

На фронтенді у Vue.js створено відповідний компонент для форми, яка використовує двостороннє зв’язування (v-model) та обробники подій для валідації й відправлення даних через API. У разі помилки – користувач бачить повідомлення. У разі успіху – оновлюється список транзакцій.

3.5 Висновки до розділу 3

У цьому розділі було розглянуто ключові аспекти побудови програмного забезпечення інформаційної системи управління фінансами для фізичної особи-підприємця. Проведено обґрунтований вибір інструментів, архітектурних підходів та технологій, що забезпечують надійну, масштабовану та зручну у супроводі систему.

По-перше, як система управління базою даних обрано PostgreSQL – потужну об’єктно-реляційну СУБД з відкритим кодом, що забезпечує транзакційну цілісність, підтримку складних типів даних, масштабованість та

високу продуктивність. Вона чудово підходить для обробки структурованої танапівструктурованої фінансової інформації, зберігаючи баланс між гнучкістю зберігання та строгістю бізнес-логіки.

По-друге, реалізація інформаційної бази здійснена за допомогою підходу Code First у середовищі Laravel, що забезпечує єдність між кодом, моделями даних та фізичною структурою БД. Завдяки використанню Laravel Eloquent, було реалізовано ефективні міжтабличні зв’язки, автоматичне створення таблиць через міграції, а також зручне початкове наповнення за допомогою сидів.

По-третє, вибір стеку Laravel + Vue.js для розробки прикладного ПЗ обґрунтовано з позиції сучасних вимог до веб-додатків: розділення відповідальностей між серверною та клієнтською частиною, підтримка REST API, зручність у тестуванні, висока продуктивність та масштабованість. Laravel забезпечує надійну обробку логіки, безпеки та роботи з даними, тоді як Vue.js дозволяє створити інтерактивний і реактивний інтерфейс користувача.

У підсумку, обрані технології та реалізовані рішення створюють надійну основу для ефективного функціонування інформаційної системи, що відповідає потребам фізичних осіб-підприємців в обліку, аналізі та плануванні фінансів.

4 Рекомендації щодо впровадження та експлуатації системи

4.1 Тестування системи

Тестування системи є одним із ключових етапів забезпечення якості програмного забезпечення, оскільки дозволяє виявити помилки, невідповідності, дефекти та забезпечити відповідність функціональності вимогам, визначеним у технічному завданні. Основною метою тестування є перевірка правильності роботи основних функціональних модулів, надійності взаємодії з базою даних, стабільності роботи системи та її готовності до використання в реальних умовах.

У процесі розробки інформаційної системи управління фінансами для ФОП було застосовано інтеграційне тестування, що дозволяє перевірити взаємодію між компонентами системи, зокрема бекенду на Laravel, клієнтської частини, бази даних PostgreSQL та API-інтерфейсів.

Для автоматизованого тестування використовувалися такі інструменти:

- PHPUnit – фреймворк для модульного та інтеграційного тестування в екосистемі Laravel;
- Laravel Test Factory та HTTP Test Requests – для симуляції HTTP-запитів до API;
- Testcontainers (PostgreSQL) – для створення ізольованого середовища бази даних під час кожного запуску тестів;
- Faker – для генерації тестових даних.

Тестування охоплювало наступні аспекти:

- перевірка створення фінансових записів (витрати/доходи);
- перевірка обробки запитів на зміну даних користувача;
- перевірка валідації вхідних запитів та повідомлень про помилки;
- перевірка відповідей API (HTTP статус-коди, структура даних);

- імітація помилкових сценаріїв (відсутність токену, некоректні поля, дублікати).

4.1.1 Приклад тестування функції створення запису (POST /api/transactions)

Дана функція відповідає за створення нового запису про транзакцію (дохід або витрату) в системі. Вона приймає тіло запиту з даними та зберігає запис у базі, повертаючи статус 201 Created у разі успіху або відповідні повідомлення про помилку в іншому випадку.

Тест-кейс для функції POST /api/transactions

Таблиця 4.1

№	Умова тестування	Очікуваний результат
1	Коректне тіло запиту, авторизаційний токен валідний	HTTP 201, об’єкт транзакції в відповіді
2	Повторний запит з ідентичним тілом	HTTP 409 (конфлікт), повідомлення про дублікат
3	Коректне тіло, але токен недійсний	HTTP 401, повідомлення “Unauthorized”
4	Некоректне тіло (відсутнє поле amount)	HTTP 422, повідомлення про помилку валідації
5	Невірний формат дати транзакції	HTTP 422, повідомлення про помилку валідації
6	Коректне тіло з надто великим описом	HTTP 422, помилка перевищення довжини
7	Коректне тіло, валідний токен, інша категорія (доходи)	HTTP 201, запис створено

Код тестування:

```
public function test_can_create_transaction_with_valid_data()
{
   // Arrange
   $user = User::factory()->create();
```

```
        $payload = [
            'amount' => 1200,
            'type' => 'income',
            'date' => '2025-05-01',
            'description' => 'Фінансова операція від клієнта'
        ];

        // Act
        $response = $this->actingAs($user)->postJson('/api/transactions', $payload);

        // Assert
        $response->assertStatus(201)
                ->assertJsonFragment(['amount' => 1200, 'type' => 'income']);
    }
```

Даний фрагмент коду демонструє структуру типового тесту у стилі arrange-act-assert: підготовка користувача, генерація даних, виконання запиту та перевірка результату.

4.1.2 Підсумки тестування

Результати тестування свідчать про наступне:

- усі основні API-методи працюють стабільно за коректного вводу даних;
- усі помилки та винятки обробляються коректно, повідомлення зрозумілі для користувача;
- інтерфейс API відповідає принципам REST та документації;
- тестування пройдено на локальному сервері у Docker-середовищі з симульованими сценаріями, наближеними до реальних.

Таким чином, система готова до експлуатації. Її функціональність підтверджена автоматичним та ручним тестуванням, що гарантує надійність, стабільність і відповідність поставленим вимогам.

4.2 Вимоги до апаратного та програмного забезпечення

Ефективне функціонування інформаційної системи управління фінансами для фізичної особи-підприємця можливе лише за дотримання відповідних вимог до апаратного та програмного забезпечення. Система реалізована за моделлю клієнт-сервер, що дозволяє масштабувати рішення залежно від потреб користувача. Нижче наведено технічні вимоги до всіх ключових компонентів інфраструктури.

4.2.1 Архітектура розгортання

Система передбачає класичну трирівневу архітектуру:

- клієнтський пристрій – взаємодія з користувачем через веб-інтерфейс або десктоп-додаток;
- веб-сервер – обробка запитів, авторизація, бізнес-логіка;
- сервер бази даних – зберігання, обробка та резервне копіювання фінансових даних.

4.2.2 Вимоги до клієнтських пристроїв

Апаратне забезпечення клієнтського пристрою

Таблиця 4.2

Компонент	Мінімальні вимоги	Рекомендовані вимоги
Процесор	2 ядра, 1.8 ГГц	4 ядра, 2.5 ГГц або вище
Оперативна пам’ять	2 ГБ	8 ГБ або більше
Дисплей	1024×768	1366×768 або вище
З’єднання з Інтернетом	Від 1 Мбіт/с	Від 10 Мбіт/с

Програмне забезпечення клієнтського пристрою

Таблиця 4.3

Компонент	Вимоги
Операційна система	Windows 10+, macOS, Linux, Android 9+, iOS 13+

Компонент	Вимоги
Браузер	Останні версії Google Chrome, Firefox, Edge
Підтримка технологій	JavaScript, HTML5, CSS3

4.2.3 Вимоги до веб-сервера

Веб-сервер виконує роль логічного центру системи: опрацьовує запити користувача, перевіряє автентичність, здійснює обробку фінансових даних та передає інформацію до бази даних.

Апаратне забезпечення веб-сервера

Таблиця 4.4

Компонент	Мінімальні вимоги	Рекомендовані вимоги
Процесор	4 ядра, 2.5 ГГц	8 ядер, 3.0 ГГц або вище
Оперативна пам’ять	8 ГБ	16 ГБ або більше
Накопичувач	SSD 256 ГБ	SSD 512 ГБ або більше
Мережа	1 Гбіт/с	10 Гбіт/с

Програмне забезпечення веб-сервера

Таблиця 4.5

Компонент	Мінімальні вимоги	Рекомендовані вимоги
Операційна система	Ubuntu 20.04 / Windows Server	Ubuntu 22.04 / Windows Server 2022
Платформа	PHP 8.1+	PHP 8.2+
Фреймворк	Laravel 9+	Laravel 12.x
Сервер запуску	Apache / Nginx	Nginx + Docker
Сертифікати SSL	Let’s Encrypt	Certbot з автоматичним оновленням

4.2.4 Вимоги до сервера бази даних

База даних зберігає інформацію про фінансові операції, клієнтів, звіти тощо. Її стабільність та продуктивність критично важливі для функціонування системи.

Апаратне забезпечення сервера бази даних

Таблиця 4.6

Компонент	Мінімальні вимоги	Рекомендовані вимоги
Процесор	4 ядра, 2.5 ГГц	8 ядер, 3.2 ГГц або вище
Оперативна пам’ять	16 ГБ	32 ГБ або більше
Накопичувач	SSD 512 ГБ	SSD 1 ТБ або більше
Мережа	1 Гбіт/с	10 Гбіт/с

Програмне забезпечення сервера бази даних

Таблиця 4.7

Компонент	Мінімальні вимоги	Рекомендовані вимоги
Операційна система	Ubuntu 20.04 / Windows Server	Ubuntu 22.04 / Windows Server 2022
СУБД	PostgreSQL 14.x	PostgreSQL 16.x
Резервування	pg_dump, ручне копіювання	Автоматичне резервування + WAL архівація
Безпека	TLS 1.2	TLS 1.3 + шифрування таблиць

4.2.5 Забезпечення безпеки та надійності

Для гарантування стабільної роботи та захисту даних необхідно:

- Встановити автоматичне резервне копіювання бази даних.
- Використовувати антивірусне програмне забезпечення на клієнтських пристроях.
- Застосовувати шифрування при передачі даних.
- Використовувати фаєрвол та механізми автентифікації.

Наведені апаратні та програмні вимоги забезпечують стабільне функціонування системи як у тестовому, так і у продуктивному середовищі. Завдяки масштабованій архітектурі та підтримці сучасних технологій, систему легко адаптувати до різних сценаріїв використання – як локально, так і в хмарній інфраструктурі (AWS, GCP, Azure).

4.3 Склад інсталяційного пакету

У зв’язку з тим, що інформаційна система управління фінансами для ФОП реалізована у вигляді веб-додатку з клієнтською частиною (HTML/CSS) та серверною частиною на фреймворку Laravel (версія 12.x), інсталяційний пакет розроблено таким чином, щоб забезпечити швидке та коректне розгортання проекту як у локальному середовищі, так і на хостингу або сервері.

4.3.1 Структура інсталяційного пакету

Основні компоненти інсталяційного пакету

Таблиця 4.8

Компонент	Призначення
.env.example	Шаблон конфігураційного файлу середовища (параметри бази даних, ключі тощо)
composer.json	Файл з описом залежностей Laravel-проекту (пакети, версії, скрипти)
package.json	Конфігурація для frontend-залежностей (npm-пакети для збірки інтерфейсу)
public/	Публічна директорія, що містить HTML/CSS-інтерфейс, початкову точку входу

Компонент	Призначення
routes/web.php	Опис маршрутів доступу до функціональних частин програми
app/	Основна бізнес-логіка проекту (контролери, моделі, обробники запитів)
database/migrations/	Схема бази даних у вигляді міграцій Laravel
database/seeders/	Початкові дані для заповнення таблиць бази даних
storage/	Каталог для логів, кешу та завантажених файлів
artisan	Консольний інструмент Laravel для запуску команд (міграції, сидування тощо)

4.3.2 Етапи встановлення системи

1. Підготовка середовища:

- встановлення PHP (версія 8.2 або вище);
- встановлення Composer (менеджер залежностей PHP);
- встановлення MySQL або PostgreSQL;
- встановлення Docker;

2. Налаштування конфігурації:

- копіювання .env.example у .env;
- генерація ключа додатку: php artisan key:generate;
- вказання параметрів бази даних та порту;

3. Ініціалізація бази даних:

- виконання міграцій: php artisan migrate;
- заповнення початковими даними: php artisan db:seed;

4. Запуск веб-додатку:

- запуск сервера: php artisan serve;
- або через Docker: docker-compose up –build;

4.3.3 Docker-контейнер

Система підтримує контейнеризоване розгортання з використанням docker-compose. У складі інсталяційного пакету є:

```
version: '3.8'

services:
  app:
    build: .
    ports:
      - "8000:8080"
    volumes:
      - .:/var/www
    depends_on:
      - db

  db:
    image: mysql:8.0
    environment:
      PGSQL_ROOT_PASSWORD: secret
      PGSQL_DATABASE: fop_finance
      PGSQL_USER: fop
      PGSQL_PASSWORD: secret
    ports:
      - "5432:5432"
```

4.3.4 Після встановлення

Після завершення розгортання система доступна за адресою http://localhost:8000. Також було зроблено наступне:

- встановлено Nginx як reverse-proxy (для продакшн);
- налаштовано SSL (через Let’s Encrypt);
- забезпечено регулярне резервне копіювання бази даних;
- здійснено налаштування прав доступу для папок storage/ та bootstrap/cache/.

4.3.5 Переваги структури інсталяційного пакету

Обраний інсталяційний пакет має наступні структурні переваги:

- простота налаштування за рахунок .env і Artisan-команд;
- гнучкість завдяки підтримці як ручного, так і контейнеризованого розгортання;
- чітке розділення клієнтської та серверної логіки;
- можливість масштабування у майбутньому через мікросервісну архітектуру Laravel.

4.4 Висновки до розділу 4

У даному розділі було наведено практичні аспекти впровадження та супроводу інформаційної системи управління фінансами для фізичної особи-підприємця. Було детально описано процес тестування функціональних компонентів системи, технічні вимоги до її успішного розгортання, а також структуру та вміст інсталяційного пакету.

Результати тестування підтвердили стабільність, надійність та відповідність реалізованого функціоналу вимогам, сформульованим у технічному завданні. Автоматизоване тестування охоплювало ключові сценарії використання системи, виявило її стійкість до типових помилок користувача та відповідність REST-принципам побудови API.

Надані апаратні та програмні вимоги демонструють, що система є масштабованою, адаптованою до різних конфігурацій середовищ і здатна функціонувати як локально, так і в хмарній інфраструктурі. Особлива увага приділяється безпеці та захисту даних, що має вирішальне значення для фінансових систем.

Інсталяційний пакет сформовано таким чином, щоб забезпечити простоту впровадження системи, її налаштування та подальше розширення. Це дозволяє швидко почати роботу як розробникам, так і кінцевим користувачам із базовими технічними навичками.

Загалом, розділ доводить, що система є технічно зрілою, готовою до практичного використання та подальшої підтримки й масштабування.

ВИСНОВКИ

Під час виконання кваліфікаційної роботи на тему «Програмне забезпечення інформаційної системи управління фінансами для фізичної особи-підприємця» було реалізовано повний цикл розробки інформаційної системи – від аналізу предметної області до створення готового програмного продукту.

На початковому етапі було проведено аналіз діяльності фізичних осіб-підприємців та програм-аналогів, які частково автоматизують управління фінансами. У результаті було виявлено, що переважна більшість існуючих рішень не відповідає сучасним вимогам щодо зручності, функціональності та гнучкості. На основі цього були визначені бізнес-вимоги, функціональні та нефункціональні вимоги до майбутньої системи. Також було сформульовано сценарії використання, визначено основні ролі користувачів та прецеденти.

Другим етапом стало проектування архітектури програмної системи. Було розроблено набір UML-діаграм, серед яких діаграми варіантів використання, класів, послідовності, компонентів, пакетів та сутностей. Вони дозволили чітко структурувати логіку роботи системи, встановити взаємозв’язки між основними компонентами та спроектувати надійну модель даних. Особлива увага приділялася розробці логічної та фізичної моделей даних, що стало основою для створення інформаційної бази.

У третьому розділі було обґрунтовано вибір технологій для реалізації системи. Для збереження даних обрано сучасну та продуктивну систему керування базами даних PostgreSQL. Розробку серверної частини здійснено з використанням Laravel 12.x, що забезпечило гнучку архітектуру та зручну роботу з ORM. Інтерфейс користувача реалізовано за допомогою фреймворку Vue.js, що забезпечило адаптивність і зручність у використанні.

У процесі реалізації було створено окремі модулі для управління доходами, витратами, податковими зобов’язаннями, а також реалізовано систему категоризації операцій та генерації звітів. Було розроблено механізми

автентифікації, валідації даних, ведення обліку податкових періодів і збереження історії фінансової активності.

Важливим етапом стала перевірка працездатності системи в різних сценаріях використання, зокрема автоматичне тестування та ручне проходження тест-кейсів. Для розгортання системи використано Docker-контейнери, що дозволяє швидко запускати застосунок у будь-якому середовищі.

У результаті реалізації проекту було створено інформаційну систему, що повністю відповідає заданим вимогам і дозволяє фізичній особі-підприємцю ефективно управляти особистими фінансами, враховуючи специфіку податкового обліку, витрат і доходів. Програмне забезпечення є зручним у використанні, надійним, гнучким до розширення функціоналу, зокрема інтеграції з банківськими сервісами, додавання функцій штучного інтелекту для прогнозування витрат або автоматичної категоризації платежів.

Підсумовуючи, розроблена система є результатом глибокого аналізу, грамотного проектування та ефективної реалізації, яка не тільки задовольняє поточні потреби цільової аудиторії, а й демонструє високий рівень професійної підготовки розробника, набуті знання й практичний досвід за роки навчання.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Закон України «Про бухгалтерський облік та фінансову звітність в Україні». URL: https://zakon.rada.gov.ua/laws/show/996-14 (дата звернення: 01.02.2025).

2. Податковий кодекс України. URL: https://zakon.rada.gov.ua/laws/show/2755-17 (дата звернення: 03.02.2025).

3. ISO/IEC 25010:2011. Systems and software engineering – Systems and software Quality Requirements and Evaluation (SQuaRE) – System and software quality models.

4. Sommerville I. Software Engineering. 10th ed. – Pearson, 2015. – 792 p.

5. Pressman R. S. Software Engineering: A Practitioner's Approach. 8th ed. – New York : McGraw-Hill, 2014. – 928 p.

6. Брайан Х. Laravel: From Apprentice To Artisan. – Leanpub, 2018.

7. PostgreSQL Documentation. URL: https://www.postgresql.org/docs/ (дата звернення: 05.02.2025).

8. Laravel 12.x Documentation. URL: https://laravel.com/docs (дата звернення: 08.02.2025).

9. Docker Documentation. URL: https://docs.docker.com/ (дата звернення: 14.02.2025).

10. Vue.js Documentation. URL: https://vuejs.org/guide/introduction.html (дата звернення: 12.02.2025).

11. PHP Manual. URL: https://www.php.net/manual/en/ (дата звернення: 15.02.2025).

12. JavaScript Guide – MDN Web Docs. URL: https://developer.mozilla.org/en-US/docs/Web/JavaScript (дата звернення: 15.02.2025).

13. TypeScript Documentation. URL: https://www.typescriptlang.org/docs/ (дата звернення: 15.02.2025).

14. W3C HTML & CSS Standards. URL: https://www.w3.org/ (дата звернення: 17.02.2025).

15. MDN Web Docs (HTML, CSS, JS). URL: https://developer.mozilla.org/ (дата звернення: 24.02.2025).

16. Фрідман Е. Laravel для начинающих. Практическое руководство. – М. : ДМК Пресс, 2020. – 304 с.

17. Гребенюк С. В. Розробка веб-додатків : навч. посіб. – Київ : КНЕУ, 2020. – 268 с.

18. Шелестов А. І., Шевченко А. А. Основи проектування інформаційних систем : навч. посіб. – Київ : Видавничий дім «Слово», 2018. – 240 с.

ДОДАТОК А

Код файлу routes/api.php:

```
use Illuminate\Support\Facades\Route;
use App\Http\Controllers\AuthController;
use App\Http\Controllers\IncomeController;
use App\Http\Controllers\ExpenseController;
use App\Http\Controllers\CategoryController;
use App\Http\Controllers\ReportController;

// Аутентифікація
Route::post('/register', [AuthController::class, 'register']);
Route::post('/login', [AuthController::class, 'login']);

// Маршрути, які вимагають аутентифікації
Route::middleware('auth:sanctum')->group(function () {
    Route::get('/user', [AuthController::class, 'user']);
    Route::post('/logout', [AuthController::class, 'logout']);

    // Доходи
    Route::apiResource('incomes', IncomeController::class);

    // Витрати
    Route::apiResource('expenses', ExpenseController::class);

    // Категорії доходів/витрат
    Route::apiResource('categories', CategoryController::class);

    // Звіти
```

```
    Route::get('/reports/monthly', [ReportController::class, 'monthly']);
    Route::get('/reports/summary', [ReportController::class, 'summary']);
    Route::get('/reports/category', [ReportController::class, 'byCategory']);
});
```

Код міграції для таблиці (сутності) users:

```
use Illuminate\Database\Migrations\Migration;
use Illuminate\Database\Schema\Blueprint;
use Illuminate\Support\Facades\Schema;

return new class extends Migration {
    public function up(): void {
        Schema::create('users', function (Blueprint $table) {
            $table->id('user_id');
            $table->string('full_name');
            $table->string('email')->unique();
            $table->string('password');
            $table->string('role')->default('user');
            $table->boolean('status')->default(true);
            $table->timestamps();
        });
    }

    public function down(): void {
        Schema::dropIfExists('users');
    }
};
```

Код міграції для таблиці (сутності) categories:

```
return new class extends Migration {
```

```
    public function up(): void {
        Schema::create('categories', function (Blueprint $table) {
            $table->id('category_id');
            $table->string('name');
            $table->enum('type', ['income', 'expense']);
            $table->foreignId('user_id')->constrained('users')->onDelete('cascade');
            $table->timestamps();
        });
    }

    public function down(): void {
        Schema::dropIfExists('categories');
    }
};
```

Код міграції для таблиці (сутності) financial_operations:

```
return new class extends Migration {
    public function up(): void {
        Schema::create('financial_operations', function (Blueprint $table) {
            $table->id('operation_id');
            $table->enum('type', ['income', 'expense']);
            $table->decimal('amount', 12, 2);
            $table->date('date');
            $table->text('description')->nullable();
            $table->foreignId('user_id')->constrained('users')->onDelete('cascade');
$table->foreignId('category_id')->constrained('categories')-
>onDelete('cascade');
            $table->timestamps();
        });
    }
```

```
    public function down(): void {
        Schema::dropIfExists('financial_operations');
    }
};
```

Код міграції для таблиці (сутності) reports:

```
return new class extends Migration {
    public function up(): void {
        Schema::create('reports', function (Blueprint $table) {
            $table->id('report_id');
            $table->string('type');
            $table->date('created_at_date');
            $table->json('filters')->nullable();
            $table->longText('content');
            $table->foreignId('user_id')->constrained('users')->onDelete('cascade');
            $table->timestamps();
        });
    }

    public function down(): void {
        Schema::dropIfExists('reports');
    }
};
```

Код міграції для таблиці (сутності) event_logs:

```
return new class extends Migration {
    public function up(): void {
        Schema::create('event_logs', function (Blueprint $table) {
            $table->id('event_log_id');
```

```
            $table->string('event');
            $table->dateTime('event_date');
            $table->text('description')->nullable();
            $table->foreignId('user_id')->constrained('users')->onDelete('cascade');
            $table->timestamps();
        });
    }

    public function down(): void {
        Schema::dropIfExists('event_logs');
    }
};
```

Код міграції для таблиці (сутності) access_levels:

```
return new class extends Migration {
    public function up(): void {
        Schema::create('access_levels', function (Blueprint $table) {
            $table->id('access_level_id');
            $table->string('rights');
            $table->date('granted_at');
            $table->date('updated_at_access');
            $table->timestamps();
        });
    }

    public function down(): void {
        Schema::dropIfExists('access_levels');
    }
};
```

Код міграції для таблиці (сутності) user_access_level:

```
return new class extends Migration {
    public function up(): void {
        Schema::create('user_access_level', function (Blueprint $table) {
            $table->foreignId('user_id')->constrained('users')->onDelete('cascade');
$table->foreignId('access_level_id')->constrained('access_levels')->onDelete('cascade');
            $table->primary(['user_id', 'access_level_id']);
        });
    }

    public function down(): void {
        Schema::dropIfExists('user_access_level');
    }
};
```