МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

Чорноморський національний університет імені Петра Могили

Факультет комп'ютерних наук

Кафедра інтелектуальних інформаційних систем

ДОПУЩЕНО ДО ЗАХИСТУ

Завідувач кафедри інтелектуальних інформаційних систем

_____ Євген СІДЕНКО

«____»_____ 2026 р.

КВАЛІФІКАЦІЙНА РОБОТА

НА ЗДОБУТТЯ ОСВІТНЬОГО СТУПЕНЯ БАКАЛАВРА

ІНТЕЛЕКТУАЛЬНА СИСТЕМА ОБЛІКУ ОСОБИСТИХ КОШТІВ НА ОСНОВІ ФІНАНСОВОГО ТРЕКЕРА

Спеціальність 122 Комп'ютерні науки

Освітня програма «Комп'ютерні науки»

Здобувач _____ Станіслав ЛЯЩЕНКО

«___»_____ 2026 р.

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

«___»_____ 2026 р.

Миколаїв – 2026

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

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

ЗАТВЕРДЖУЮ

Завідувач кафедри інтелектуальних інформаційних систем

_____ Євген СІДЕНКО

«____» _____ 2025 р.

ЗАВДАННЯ
на кваліфікаційну роботу здобувача

Лященка Станіслава Сергійовича
((прізвище, ім’я, по батькові здобувача)

1. Тема кваліфікаційної роботи: «Інтелектуальна система обліку особистих коштів на основі фінансового трекера».

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

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

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

3. Очікуваний результат роботи та початкові дані, якщо такі потрібні: розроблений вебзастосунок для обліку особистих фінансів з функціями ведення доходів і витрат, аналізу даних та формування звітів. Початковими даними є вимоги до системи, результати аналізу аналогів та обрані засоби розробки.

4. Перелік питань, що підлягають розробці: аналіз існуючих програмних рішень для обліку особистих фінансів; вибір інструментів і технологій розробки системи;

проєктування структури та архітектури програмного забезпечення; реалізація функціональних можливостей фінансового трекера; тестування системи та перевірка її працездатності.

5. Перелік графічних матеріалів: презентація, рисунки, таблиці.

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

Здобувач _____ Станіслав ЛЯЩЕНКО
(Особистий підпис) *(Власне ім'я ПРІЗВИЩЕ)*

Дата видачі завдання «24» грудня 2025 р.

КАЛЕНДАРНИЙ ПЛАН
кваліфікаційної роботи

Тема: Інтелектуальна система обліку особистих коштів на основі фінансового трекера

№	Найменування роботи	Початок	Закінчення	Примітки
1	Отримання завдання на виконання КР	21.12.2025	24.12.2025	
2	Аналіз предметної області та постановка задачі	25.12.2025	30.01.2026	Виконано
3	Огляд літературних джерел за темою кваліфікаційної роботи, зокрема аналіз сучасних підходів до обліку особистих фінансів та існуючих фінансових трекерів.	31.01.2026	01.03.2026	Виконано
4	Огляд технологій, методів і програмних засобів, що використовуються для розроблення інтелектуальних систем обліку особистих фінансів.	02.03.2026	01.04.2026	Виконано
5	Реалізація фінансового трекера з аналізом результатів його функціонування та оцінкою ефективності роботи.	02.04.2026	24.05.2026	Виконано
6	Перший попередній захист КР на засіданні комісії кафедри	25.05.2026	25.05.2026	Виконано
7	Корегування роботи за результатами попереднього захисту	26.05.2026	04.06.2026	Виконано
8	Другий попередній захист КР на засіданні комісії кафедри	05.06.2026	05.06.2026	Виконано
9	Доробка та остаточне оформлення КР	06.06.2026	14.06.2026	Виконано
10	Подання КР, її електронної копії та інших документів (відгуку, рецензії) до захисту	15.06.2026	19.06.2026	Виконано

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

Здобувач _____ Станіслав ЛЯЩЕНКО
(Особистий підпис) *(Власне ім'я ПРІЗВИЩЕ)*

Дата складання календарного плану
«29» січня 2026 р.

АНОТАЦІЯ

до кваліфікаційної роботи
здобувача групи 401 ЧНУ ім. Петра Могили

Лященка Станіслава Сергійовича

на тему: **«ІНТЕЛЕКТУАЛЬНА СИСТЕМА ОБЛІКУ ОСОБИСТИХ КОШТІВ НА ОСНОВІ ФІНАНСОВОГО ТРЕКЕРА»**

Актуальність роботи визначається зростаючою потребою у цифрових рішеннях для особистого фінансового менеджменту, що поєднують простоту використання з елементами інтелектуального аналізу даних.

Об'єктом роботи є процес обліку та аналізу особистих фінансів користувача.

Предметом роботи є методи та засоби розроблення інтелектуальних систем і фінансових трекерів, зокрема архітектурний патерн MVC та методи аналізу фінансових даних, а також фреймворк Laravel і СУБД MySQL.

Метою роботи є підвищення рівня фінансової обізнаності та контролю особистих витрат шляхом застосування інтелектуальних методів аналізу фінансових даних і автоматизації процесів їх обліку та інтерпретації.

В результаті проведених досліджень та практичної реалізації поставлених завдань було створено програмний продукт, що автоматизує процес ведення особистого бюджету, забезпечує централізоване зберігання фінансових даних, їх аналіз та візуалізацію.

Пояснювальна записка бакалаврської роботи складається зі вступу, чотирьох розділів, висновків і додатків. У першому розділі розглянуто предметну область управління особистими фінансами, обґрунтовано необхідність створення інтелектуальної системи обліку фінансів та проведено аналіз існуючих фінансових трекерів і систем бюджетування. У другому розділі проаналізовано та обґрунтовано вибір технологій для реалізації вебзастосунку, зокрема засобів розробки клієнтської та серверної частин, системи керування базами даних, а також інструментів для обробки та аналізу фінансових даних. У третьому розділі представлено проєктування системи: описано структуру бази даних, архітектуру

програмного забезпечення, функціональні модулі, алгоритми аналізу фінансових операцій і сценарії взаємодії користувача із системою. У четвертому розділі представлено програмну реалізацію системи. Описано процес обробки фінансових даних, механізм формування звітності, а також результати тестування вебзастосунку. Наведено тест-кейси та проаналізовано результати перевірки працездатності й коректності функціонування розробленої системи.

Загальний обсяг роботи – 80 сторінок. Кваліфікаційна робота містить 3 додатки, 22 рисунки, 28 таблиць і 32 джерела посилання.

Ключові слова: фінансовий трекер, вебзастосунок, Laravel, MySQL, MVC, особисті фінанси, бюджетний ліміт, мультивалютність.

ABSTRACT

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

Lyashchenko Stanislav

"INTELLIGENT PERSONAL FINANCE ACCOUNTING SYSTEM BASED ON A FINANCIAL TRACKER"

A relevance of the topic is determined by the growing need for digital solutions for personal financial management, combining ease of use with elements of intelligent data analysis.

A object of the work is methods and tools for developing intelligent systems and financial trackers, in particular the MVC architectural pattern and methods for analyzing financial data, as well as the Laravel framework and MySQL DBMS.

A subject of the work is methods and tools for developing intelligent systems, as well as software technologies for creating financial trackers.

A purpose of the work is to increase the level of financial awareness and control of personal expenses by applying intelligent methods of analyzing financial data and automating the processes of their accounting and interpretation.

As a result of the research conducted and the practical implementation of the tasks set, a software product was created that automates the process of maintaining a personal budget, provides centralized storage of financial data, its analysis and visualization.

The explanatory note of the bachelor's thesis consists of an introduction, four sections, conclusions and appendices. The first section examines the subject area of personal finance management, justifies the need to create an intelligent financial accounting system, and analyzes existing financial trackers and budgeting systems. The second section analyzes and justifies the choice of technologies for implementing a web application, in particular, client and server development tools, database management systems, and tools for processing and analyzing financial data. The third section presents the system design: describes the database structure, software architecture, functional modules, financial transaction analysis algorithms, and user interaction scenarios with the system. The fourth section presents the software implementation of the system. The

process of processing financial data, the reporting mechanism, and the results of testing the web application are described. Test cases are presented and the results of testing the operability and correctness of the developed system are analyzed.

The overall scope of the work is 80 pages. Thesis contains 3 application, 22 figures, 28 tables and 32 references in it.

Keywords: financial tracker, web application, Laravel, MySQL, MVC, personal finance, budget limit, multi-currency.

ЗМІСТ

СКОРОЧЕННЯ ТА УМОВНІ ПОЗНАКИ

НБУ – Національний банк України

СУБД – Система управління базами даних

API – Application Programming Interface

CSS – Cascading Style Sheets

CSV – Comma-Separated Values

HTML – HyperText Markup Language

JS – JavaScript

MVC – Model-View-Controller

UML – Unified Modeling Language

ВСТУП

У сучасних умовах нестабільної економічної ситуації, що посилилася під впливом глобальних кризових процесів і наслідків повномасштабної війни на території України, суттєво ускладнила процес управління особистими фінансами. За таких умов контроль доходів і витрат, планування бюджету та накопичення коштів стають важливими складовими фінансової безпеки населення. Проте значна частина користувачів стикається з труднощами під час ведення обліку власних витрат і прийняття обґрунтованих фінансових рішень.

Для розв’язання зазначених проблем дедалі активніше використовують інформаційні системи, орієнтовані на автоматизацію фінансового обліку. Такі рішення дають змогу не лише реєструвати фінансові операції, а й аналізувати структуру витрат, виявляти особливості фінансової поведінки користувача та контролювати дотримання встановлених обмежень бюджету. Додатковою перевагою є можливість своєчасно отримувати рекомендації щодо раціонального розподілу коштів. Найчастіше подібний функціонал реалізують у вебзастосунках або мобільних системах, що забезпечує постійний доступ до фінансової інформації незалежно від місця перебування користувача.

Під час розробки подібних інформаційних систем особливого значення набувають питання безпеки, надійності та зручності використання. Система повинна гарантувати належний рівень захисту конфіденційних даних, запобігати втраті або спотворенню фінансової інформації та забезпечувати коректне зберігання історії операцій. Також необхідно надати користувачеві ефективні засоби візуалізації й аналізу накопичених даних.

Декларація про використання ШІ. Під час підготовки наукової роботи (академічного тексту) було використано інструменти штучного інтелекту – Claude (Anthropic) та ChatGPT (OpenAI). Їх було застосовано виключно для таких допоміжних завдань:

- уточнення формулювань окремих визначень і термінів у теоретичній

частині роботи;

– перефразування окремих речень для покращення стилістики та академічного стилю викладу;

– консультування щодо синтаксису окремих фрагментів коду.

Весь основний текст, аналіз, інтерпретація даних та висновки є результатом моєї власної інтелектуальної праці. Я перевірив всю інформацію, отриману від ШІ, на точність та достовірність і несу повну відповідальність за зміст цієї роботи.

1 АНАЛІЗ СУЧАСНОГО СТАНУ ПОСТАВЛЕНОЇ ЗАДАЧІ

1.1 Опис предметної сфери

Предметна сфера дослідження охоплює сукупність процесів, пов’язаних із обліком, структуризацією, аналізом та контролем особистих фінансових ресурсів користувача. Вона включає операції з фіксації доходів і витрат, формування індивідуального бюджету, встановлення фінансових обмежень, а також оцінювання фінансової поведінки з метою її подальшої оптимізації.

Збільшення кількості фінансових операцій та зростання вимог до раціонального використання ресурсів посилюють потребу в автоматизованих засобах обліку. Використання традиційних методів, зокрема паперових записів або електронних таблиць, часто ускладнює отримання цілісної картини фінансового стану. Крім того, такі підходи не забезпечують оперативного аналізу даних, мають обмежені можливості для контролю витрат і потребують значних зусиль для консолідації інформації з різних джерел. У результаті якість фінансового планування та обґрунтованість прийнятих рішень можуть суттєво знижуватися.

Активний розвиток цифрових технологій сприяв поширенню спеціалізованих систем фінансового трекінгу. Такі програмні рішення автоматизують накопичення фінансових даних, забезпечують їх структуроване зберігання та надають інструменти для візуального представлення результатів аналізу. Важливе місце посідають механізми бюджетного контролю. Саме вони дозволяють встановлювати ліміти для окремих категорій витрат, відстежувати їх дотримання та своєчасно реагувати на ризик перевищення запланованих показників [1].

На сьогоднішній день представлено велику кількість фінансових трекерів, які відрізняються за функціональністю, рівнем автоматизації та підходами до аналізу даних. До найбільш відомих належать такі системи, як YNAB, Saldo Finance, Monobudget, Firefly III, Toshl Finance та Spendee.

Поширеність таких систем підтверджує високий інтерес користувачів до

інструментів цифрового управління особистими фінансами. Разом із тим наявні рішення не завжди враховують індивідуальні особливості фінансової поведінки. У багатьох випадках обмеженими залишаються можливості інтелектуального аналізу даних, гнучкого налаштування бюджетних обмежень та адаптації функціоналу до змін у фінансових звичках користувача.

Таким чином, досліджувана предметна сфера даної роботи охоплює питання обліку та аналізу особистих фінансів, організації контролю витрат і дотримання бюджетних обмежень, а також дослідження сучасних інформаційних технологій, що можуть бути використані для розроблення ефективної інтелектуальної системи фінансового трекінгу.

1.2 Аналіз існуючих рішень та огляд наукових публікацій

У зв’язку зі зростанням потреби в ефективному управлінні особистими фінансами, особливо в умовах економічної нестабільності, значної популярності набули цифрові інструменти для ведення обліку доходів і витрат. Розглянемо декілька відомих програмних рішень, які використовуються для фінансового трекінгу.

1. YNAB (You Need A Budget) [2].

YNAB – вебплатформа для управління особистими фінансами, заснована на методології проактивного бюджетування. Ключовий принцип застосунку полягає в тому, що кожна грошова одиниця розподіляється між категоріями витрат ще до того, як її буде витрачено. Платформа підтримує крос-платформну синхронізацію, спільний бюджет для родини, постановку фінансових цілей та детальну звітність.

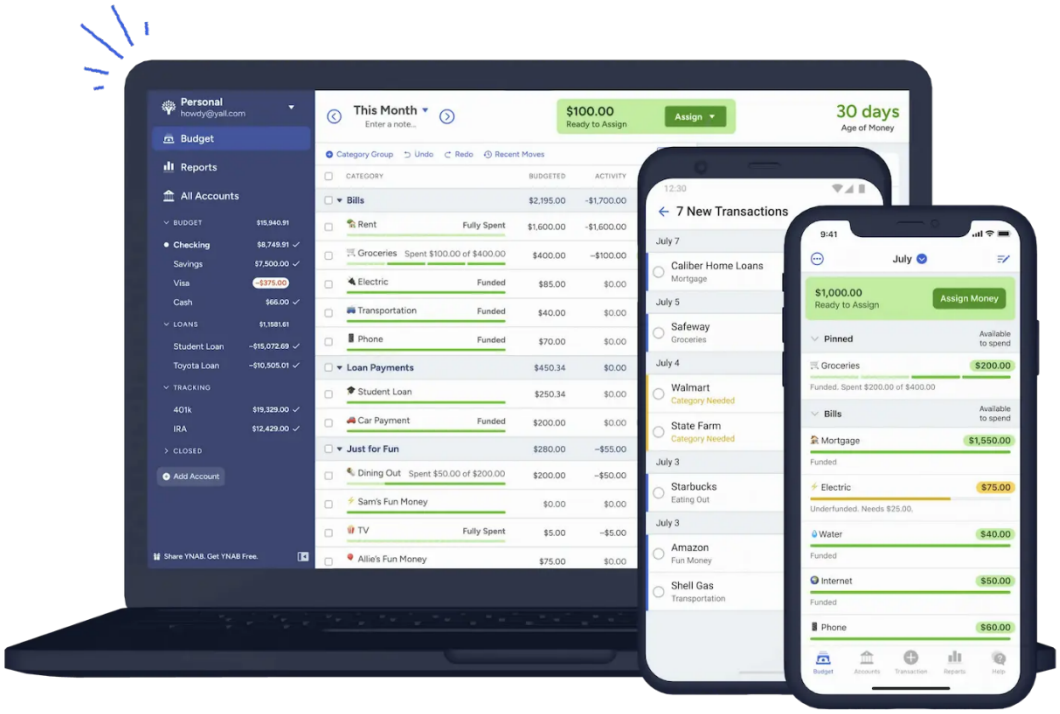

Рисунок 1.1 – Застосунок YNAB [2]

До переваг належить глибока методологічна база, інтуїтивний інтерфейс, висока безпека даних та розвинена освітня екосистема. Для нових користувачів передбачено 34-денний безкоштовний пробний період, а для студентів – річний безкоштовний доступ.

Однак для українського ринку застосунок має суттєві обмеження. Відсутня інтеграція з вітчизняними банками, тому всі транзакції вносяться вручну. Вартість підписки є значною сумою для більшості українських користувачів. Інтерфейс доступний лише англійською мовою, а весь навчальний контент орієнтований на англомовний ринок. Крім того, методологія розрахована на стабільний дохід і передбачувані витрати, що ускладнює її застосування в умовах економічної нестабільності.

2. Saldo Finance [3].

Ще одним інструментом є Saldo Finance – застосунок для обліку особистих і бізнес-фінансів, який дозволяє відстежувати доходи та витрати, формувати бюджети й аналізувати грошові потоки. Користувачам доступне підключення

банківських рахунків для автоматичної синхронізації транзакцій, а також можливість ручного додавання операцій для повного обліку активів.

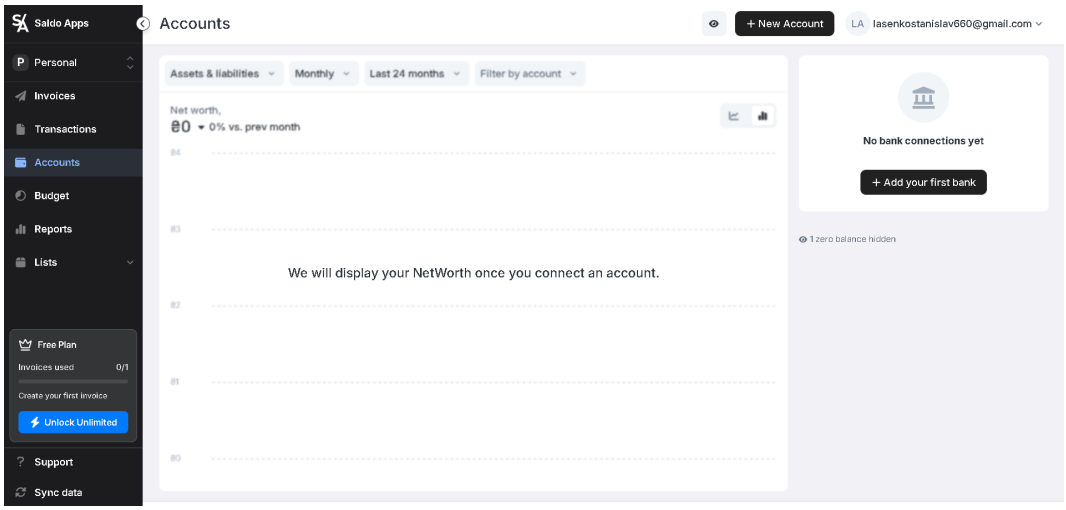

Рисунок 1.2 – Інтерфейс застосунку Saldo Finance [3]

Головна перевага – наявність вебверсії та підтримка автоматичної синхронізації з великою кількістю банків, що дозволяє централізовано відстежувати фінанси без необхідності вести облік вручну.

Суттєвим обмеженням є те, що фактична автоматична інтеграція переважно орієнтована на міжнародні банки (зокрема США та Канади), тоді як для українських банків повноцінна синхронізація відсутня. У результаті користувачам в Україні часто доводиться вручну додавати більшість транзакцій, що знижує рівень автоматизації та зручність використання застосунку.

3. Monobudget [4].

Ще одним поширеним інструментом є Monobudget, який створений спеціально для клієнтів Monobank на базі його офіційного публічного API. Дозволяє автоматично синхронізувати транзакції, встановлювати бюджети по категоріях, відстежувати сімейні витрати через спільний доступ, а також фіксувати інші активи – готівку, акції, криптовалюту.

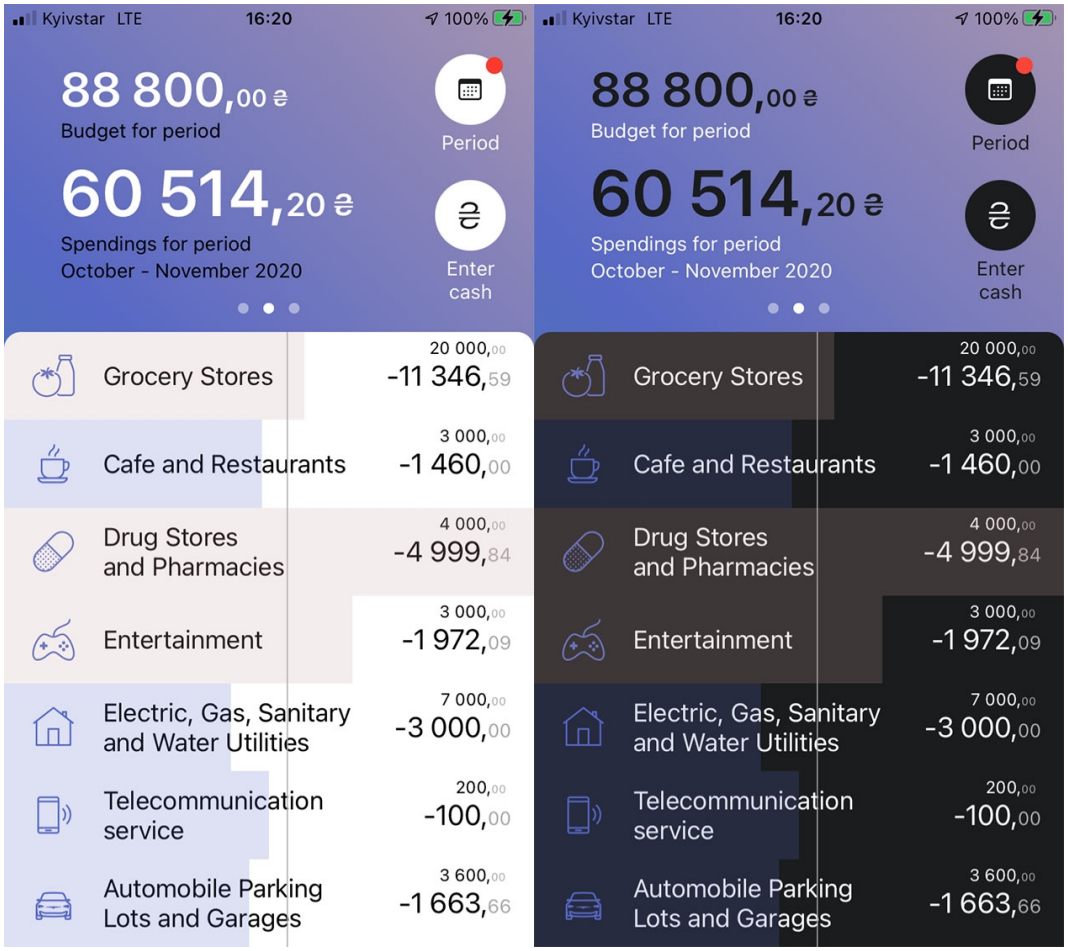

Рисунок 1.3 – Інтерфейс застосунку Monobudget [4]

Головна перевага – повна автоматизація обліку для користувачів Monobank без ручного введення даних.

Суттєвим обмеженням є вузька спеціалізація застосунку: він орієнтований виключно на клієнтів Monobank і не підтримує інші банківські установи, що унеможливлює ведення консолідованого бюджету з рахунками у кількох фінансових організаціях. Окрім того, застосунок доступний лише для пристроїв на платформі iOS, що суттєво звужує коло потенційних користувачів і виключає власників Android-пристроїв.

4. Firefly III [5].

Firefly III – безкоштовний застосунок з відкритим вихідним кодом, який розгортається на власному сервері користувача. Підтримує детальну категоризацію витрат, повторювані транзакції, бюджетні ліміти та імпорт CSV.

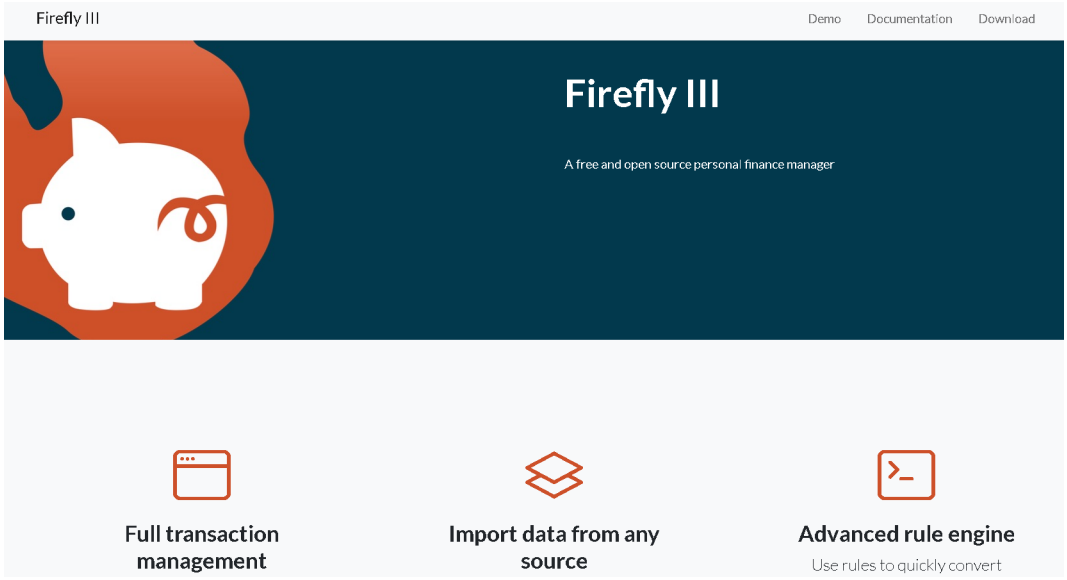

Рисунок 1.4 – Інтерфейс застосунку Monobudget [5]

Головна перевага – повний контроль над даними без передачі інформації стороннім сервісам, що є важливим з огляду на безпеку. Підходить для технічно досвідчених користувачів.

Недолік – складне початкове налаштування, відсутність готової мобільної інтеграції з українськими банками та потреба у власній серверній інфраструктурі.

5. Toshl Finance [6].

Wallet by BudgetBakers – веб та мобільний застосунок для відстеження особистих і сімейних фінансів. Підтримує ручне введення транзакцій, імпорт з банків через CSV та автоматичну синхронізацію для ряду банків.

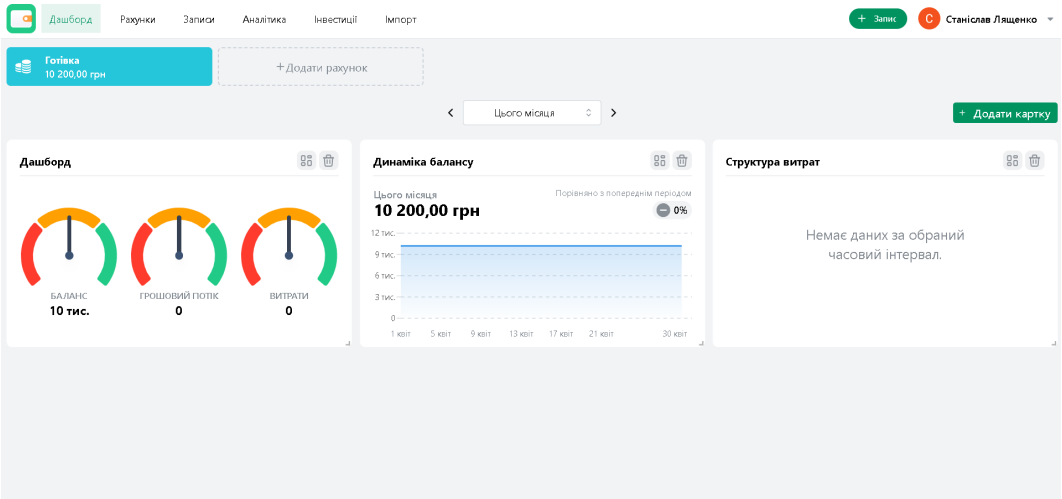

Рисунок 1.5 – Інтерфейс застосунку Toshl Finance [6]

Серед переваг – підтримка гривні, часткова інтеграція з Monobank та ПриватБанком, мультивалютність і зручний спільний бюджет для родини. Безкоштовна версія охоплює базовий функціонал, преміум-підписка коштує близько 30 євро на рік – значно доступніше за YNAB.

Основний недолік – автоматична синхронізація з українськими банками нестабільна і потребує ручного налаштування.

6. Spendee [7].

Spendee – міжнародний веб та мобільний застосунок для відстеження витрат і планування бюджету, розроблений чеською компанією Cleevio. Підтримує ручне введення транзакцій, імпорт банківських виписок і підключення рахунків через відкритий банківський протокол Open Banking для європейських банків. Функціонал включає спільні гаманці, категоризацію витрат, бюджетні ліміти та наочну аналітику.

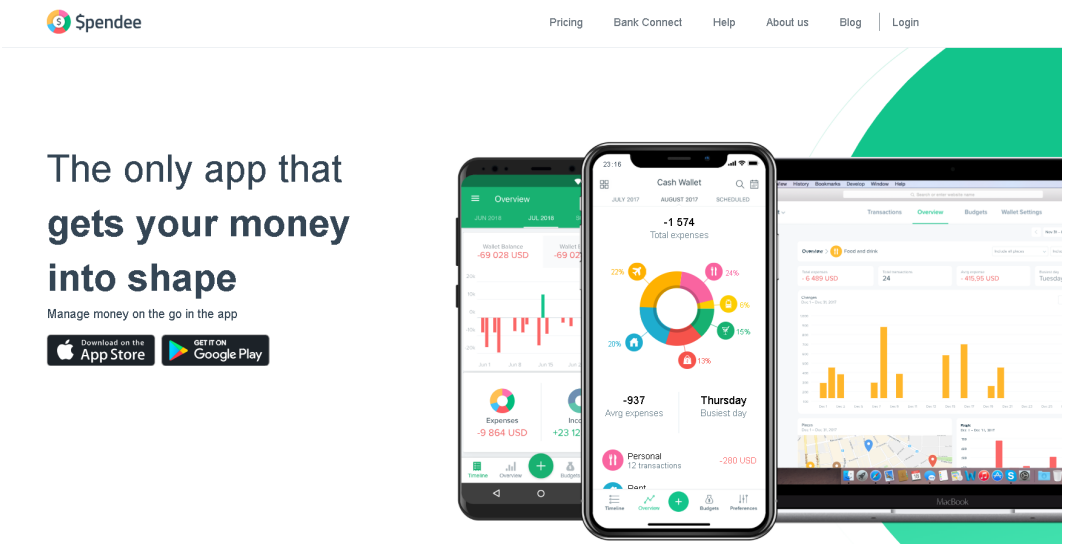

Рисунок 1.6 – Інтерфейс застосунку Spendee [7]

До переваг належить підтримка гривні, зручний інтерфейс і наявність безкоштовної версії.

Недоліком для українських користувачів є відсутність автоматичної інтеграції з вітчизняними банками – усі транзакції вносяться вручну або через CSV-імпорт.

Наукові дослідження останніх років підтверджують зростаючу актуальність цифрових систем для автоматизації обліку та аналізу особистих фінансів. Нижче наведено узагальнений аналіз релевантної кваліфікаційної роботи, що відповідає тематиці дослідження.

1. Кваліфікаційна робота «Розробка вебдодатку для оптимізації ведення бюджету із використанням C#». Автор: Барабаш В. О., Тернопільський національний технічний університет імені Івана Пулюя, 2024 [8].

Бакалаврська кваліфікаційна робота присвячена аналізу, проєктуванню та практичній реалізації вебзастосунку для автоматизації процесів бюджетування з використанням сучасного технологічного стеку платформи .NET. Основу реалізації становить мова програмування C# у поєднанні з фреймворком ASP.NET Core, система управління базами даних MS SQL Server та інструмент об'єктно-реляційного відображення Entity Framework Core.

У першому розділі досліджено предметну область фінансового обліку, проведено аналіз конкурентних рішень та визначено функціональні й нефункціональні вимоги до системи. Другий розділ охоплює вибір архітектурного підходу, проєктування структури застосунку та програмну реалізацію його складових. У третьому розділі описано процес верифікації та тестування розробленого рішення з використанням фреймворку xUnit для unit-тестів.

Результатом роботи є функціональний вебзастосунок для оптимізації ведення особистого бюджету, що реалізує облік доходів і витрат, підтримує категоризацію фінансових операцій та формування аналітичних даних. Отримані результати підтверджують ефективність застосування інструментарію платформи .NET у розробці фінансових вебсистем.

2. Кваліфікаційна робота «Розробка застосунку для ведення обліку особистих фінансів». Автор: Оліфіренко О., Державний університет інформаційно-комунікаційних технологій, 2025 [9].

Бакалаврська кваліфікаційна робота присвячена дослідженню та практичній реалізації застосунку для автоматизації обліку особистих фінансів з акцентом на забезпечення автономності, конфіденційності та зручності для кінцевого користувача. Основу технологічного стеку складають мова програмування Python, бібліотека Tkinter для побудови графічного інтерфейсу та вбудована база даних SQLite для локального зберігання фінансових даних.

У першому розділі досліджено теоретичні основи автоматизації обліку особистих фінансів в IT-середовищі, розглянуто сучасні цифрові технології управління бюджетом та питання безпеки й конфіденційності даних. Обґрунтовано доцільність розробки власного локального застосунку як альтернативи хмарним фінансовим сервісам, що вимагають постійного підключення до мережі. Другий розділ присвячено вибору інструментарію розробки та проєктуванню архітектури системи.

Реалізований застосунок забезпечує ведення транзакцій, категоризацію витрат і доходів, візуалізацію фінансових потоків, формування щомісячних звітів,

аналіз динаміки фінансового стану та захист доступу. Ключовою перевагою є повністю локальна робота без підключення до мережі, що гарантує конфіденційність фінансової інформації користувача. Результати роботи підтверджують практичну цінність запропонованого підходу для осіб, які потребують безпечного та гнучкого інструменту особистої бухгалтерії.

3 Кваліфікаційна робота «Вебзастосунок для управління особистими фінансами». Автор: Бондаренко В. С., 2025 [10].

Бакалаврська кваліфікаційна робота присвячена розробці вебзастосунку для ефективного управління особистими фінансами з використанням сучасних вебтехнологій. У роботі розглянуто основні підходи до побудови клієнт-серверних систем, а також проведено аналіз існуючих рішень у сфері фінансового обліку.

Особливу увагу приділено проєктуванню архітектури застосунку, реалізації інтерфейсу користувача та організації взаємодії з серверною частиною. У межах дослідження обґрунтовано вибір технологічного стеку, що забезпечує масштабованість, продуктивність та зручність використання системи. Для моделювання структури системи використано UML-діаграми, що відображають основні компоненти та сценарії взаємодії користувача із застосунком.

Розроблений вебзастосунок реалізує функціонал обліку доходів і витрат, категоризації фінансових операцій, формування звітів та візуалізації даних. Також передбачено можливості керування обліковим записом користувача та забезпечення базового рівня безпеки даних.

Результатом роботи є функціональний програмний продукт, що може бути використаний як основа для подальшого розвитку систем управління особистими фінансами та інтеграції з додатковими сервісами.

4. Кваліфікаційна робота «Впровадження компонентно-орієнтованої архітектури для створення додатку управління особистими фінансами з використанням React та Electron». Автор: Харлан А. А., Тернопільський національний технічний університет імені Івана Пулюя, 2024 [11].

Магістерська кваліфікаційна робота присвячена дослідженню та практичній

реалізації компонентно-орієнтованої архітектури у процесі розробки десктопного додатку для управління особистими фінансами. Стек технологій включає React та Electron як основу для побудови кросплатформного інтерфейсу, TypeScript як мову програмування зі статичною типізацією, а також Firebase як хмарну платформу для зберігання даних та автентифікації користувачів.

У роботі проведено детальний огляд конкурентних рішень на ринку (YNAB, Empower, Simplifi, Rocket Money, Monarch), здійснено аналіз технології Open Banking та обґрунтовано вибір технологічного стеку. Як архітектурний патерн застосовано модель MVC у поєднанні з принципами компонентно-орієнтованої архітектури. Для планування розробки використано методологію Scrum. Розроблено UML-діаграми варіантів використання, класів та послідовностей, а також макети сторінок додатку.

Реалізований додаток охоплює функціонал обліку доходів і витрат, управління бюджетом та категоріями витрат, відстеження боргів і рахунків, формування звітів з графічною візуалізацією, а також налаштування профілю користувача. Результатом є функціональний десктопний застосунок, протестований на операційних системах Windows 10 та Windows 11, що демонструє практичну ефективність компонентно-орієнтованого підходу у розробці програмного забезпечення фінансового спрямування.

5. Кваліфікаційна робота «Розробка дизайну мобільного додатку для оптимізації фінансових потоків». Автор: Ситняк О. Р., Харківський національний університет радіоелектроніки, 2025 [12].

Бакалаврська кваліфікаційна робота присвячена дослідженню та розробці дизайну мобільного застосунку для обліку й оптимізації особистих фінансових потоків. Цільовою аудиторією визначено молодь віком від 20 до 30 років – студентів та молодих фахівців, які мають складну структуру доходів і потребують зручного інструменту для щоденного фінансового контролю.

У роботі проведено аналіз конкурентних рішень (Money Lover, Wallet by BudgetBakers, Monefy), виявлено їх переваги та недоліки, зокрема надмірну

функціональну складність і платний доступ до ключових можливостей. На основі результатів аналізу сформовано логіку взаємодії користувача із застосунком: розроблено схеми User Flow для сценаріїв реєстрації, управління рахунками, обліку транзакцій, аналітики та налаштувань. Інструментом для проєктування обрано Figma. Дизайн-макет базується на модульній сітці, адаптованій до роздільної здатності екранів iPhone 13/14 (390×844 px), з використанням шрифту Inter та мінімалістичної кольорової палітри з акцентами помаранчевого, червоного та зеленого кольорів.

Результатом роботи є повноцінний дизайн-проєкт мобільного застосунку з інтерактивним прототипом, що охоплює функціонал обліку доходів і витрат, управління гаманцями різних типів (рахунки, цілі накопичення, борги, інвестиції), фінансову аналітику та персоналізовані налаштування. Запропоноване рішення відрізняється простотою інтерфейсу, відсутністю обов'язкових платних підписок та орієнтацією на потреби молодої аудиторії.

Аналіз існуючих вебзастосунків та публікацій для управління особистими фінансами засвідчив, що переважна більшість розглянутих рішень не підтримує української мови та не адаптована до потреб вітчизняних користувачів. Окрім того, значна частина сервісів характеризується перевантаженим інтерфейсом або обмеженим безкоштовним функціоналом, що знижує їх практичну цінність для повсякденного використання. Наявні системи також здебільшого не забезпечують достатньої гнучкості налаштувань та не приділяють належної уваги наочності й зрозумілості статистичних даних для пересічного користувача.

З урахуванням виявлених обмежень наявних рішень, у межах проєкту доцільно розробити вебзастосунок, що реалізує концепцію інтелектуального фінансового трекера з україномовним інтерфейсом, інтуїтивно зрозумілим і естетично привабливим дизайном, а також наочною статистикою фінансової активності. Передбачені гнучкі налаштування дозволять користувачеві адаптувати систему під індивідуальні потреби, що у сукупності забезпечить вищий рівень залученості та ефективності управління особистим бюджетом порівняно з

існуючими аналогами. Такий застосунок може включати наступні ключові можливості.

1. Облік доходів і витрат – фіксація фінансових операцій з можливістю їх категоризації та коментування, що забезпечує повний контроль над особистим бюджетом.

2. Інтуїтивний та естетично привабливий інтерфейс – мінімалістичний дизайн із зрозумілою навігацією, орієнтований на комфортне щоденне використання без необхідності попереднього навчання.

3. Україномовний інтерфейс – повна локалізація застосунку українською мовою, що робить його доступним і зручним для вітчизняних користувачів.

4. Підтримка українських банків – можливість імпорту історії транзакцій з виписок популярних українських банків (наприклад, ПриватБанк, Monobank), що суттєво спрощує процес введення даних.

5. Бюджетні ліміти – встановлення обмежень витрат за категоріями з автоматичним відстеженням їх дотримання та сповіщенням при наближенні до ліміту.

6. Графіки та статистика – наочна візуалізація фінансової активності у вигляді діаграм і звітів, що дозволяє користувачеві аналізувати структуру витрат і динаміку доходів за різні періоди.

7. Мультивалютність – підтримка операцій у різних валютах з можливістю автоматичного перерахунку за актуальним курсом.

1.3 Постановка задачі

Актуальність теми визначається зростаючою потребою у цифрових рішеннях для особистого фінансового менеджменту, що поєднують простоту використання з елементами інтелектуального аналізу даних.

Об’єктом роботи є процес обліку та аналізу особистих фінансів користувача.

Предметом роботи є методи та засоби розроблення інтелектуальних систем і фінансових трекерів, зокрема архітектурний патерн MVC та методи аналізу фінансових даних, а також фреймворк Laravel і СУБД MySQL.

Мета роботи полягає у створені інтелектуальної системи обліку особистих фінансів, яка забезпечує зручне введення даних, їх аналіз та формування рекомендацій для користувача.

Для досягнення мети було поставлено такі основні **завдання**:

– провести аналіз існуючих програмних рішень для обліку особистих фінансів;

– обґрунтувати вибір інструментів і технологій розробки системи;

– розробити структуру та архітектуру програмного забезпечення;

– реалізувати функціональні можливості фінансового трекера;

– здійснити тестування системи та перевірку її працездатності;

– оцінити ефективність розробленого рішення та визначити напрями його вдосконалення.

Під час проєктування вебзастосунку визначено систему функціональних модулів, кожен із яких виконує окрему роль у загальній архітектурі та забезпечує користувачу доступ до повного спектра можливостей платформи. Така модульна організація дозволяє розмежувати відповідальність компонентів і спростити подальший розвиток системи без порушення цілісності її логіки.

Основою системи є модуль реєстрації та автентифікації, що надає можливість створити персональний обліковий запис і захистити доступ до фінансових даних. Авторизований користувач отримує доступ до всіх функцій відповідно до свого профілю.

Після входу користувач потрапляє до особистого кабінету, де відображається зведена інформація про поточний фінансовий стан: баланс, останні операції, виконання бюджетних лімітів та персональні налаштування облікового запису.

Центральним функціональним блоком є модуль обліку доходів і витрат. Він забезпечує додавання, редагування та видалення фінансових операцій із їх розподілом за категоріями, що дає змогу деталізовано відстежувати рух коштів.

Для наочного аналізу фінансової активності реалізовано модуль статистики та графіків. Він відображає динаміку доходів і витрат у вигляді діаграм і зведених звітів за обраний період, що спрощує оцінку власної фінансової поведінки.

Окремий модуль відповідає за управління бюджетними лімітами. Користувач може встановлювати граничні суми витрат за категоріями, а система автоматично відстежує їх дотримання та повідомляє про наближення до встановленої межі.

З метою спрощення введення даних передбачено підтримку імпорту банківських виписок від популярних українських банків. Це дозволяє автоматично завантажувати історію транзакцій без необхідності ручного введення кожної операції.

Для зручності роботи з різними валютами реалізовано модуль мультивалютності з автоматичним перерахунком за актуальним курсом, що є особливо актуальним для користувачів із доходами або витратами в іноземній валюті.

Розгортання модулів здійснюється поетапно: спочатку реалізується базовий функціонал – реєстрація та облік операцій, після чого підключаються інструменти аналізу та бюджетування, а завершальним етапом є інтеграція з банківськими сервісами та розширені налаштування системи.

Висновки до розділу 1

У першому розділі проведено комплексний аналіз предметної сфери та визначено актуальність розроблення інтелектуальної системи обліку особистих фінансів. Встановлено, що в умовах економічної нестабільності питання ефективного управління особистим бюджетом набуває особливої практичної значущості, а попит на цифрові інструменти фінансового трекінгу невпинно зростає.

За результатами огляду існуючих програмних аналогів – YNAB, Saldo Finance, Monobudget, Firefly III, Toshl Finance та Spendee – виявлено ряд спільних недоліків: відсутність україномовного інтерфейсу, обмежена або нестабільна інтеграція з вітчизняними банками, надмірна функціональна складність та платний доступ до ключових можливостей. Огляд публікацій кваліфікаційних робіт підтвердив, що існуючі дослідження здебільшого зосереджені на базовому обліку операцій і не охоплюють питання адаптації системи до потреб українських користувачів.

На основі проведеного аналізу сформульовано мету, об'єкт і предмет дослідження, визначено перелік завдань, а також обґрунтовано доцільність розроблення власного вебзастосунку, що поєднуватиме повноцінний функціонал фінансового трекінгу з україномовним інтерфейсом, підтримкою українських банків, бюджетними лімітами, наочною статистикою та мультивалютністю.

2 МЕТОДИ ТА ІНФОРМАЦІЙНІ ТЕХНОЛОГІЇ ДЛЯ ВИРІШЕННЯ ПОСТАВЛЕНОЇ ЗАДАЧІ

2.1 Обґрунтування вибору методів для розв’язання поставлених задач

При розробці вебзастосунку для обліку особистих фінансів важливим етапом стає обґрунтований вибір технологічного стеку. Від правильності цього вибору залежить продуктивність системи, зручність її подальшої підтримки та масштабованість. У межах даного аналізу розглянуто основні категорії технологій, зокрема серверні фреймворки, системи керування базами даних, інструменти локального середовища розробки та підходи до реалізації клієнтського інтерфейсу.

Для реалізації серверної частини застосунку було розглянуто три широко використовувані фреймворки: Laravel, Django та Express.js.

Django [13] є потужним фреймворком із вбудованою адміністративною панеллю та суворою структурою проєкту, однак вимагає знання мови Python і має вищий поріг входження для розробників. Натомість Express.js [14] пропонує мінімалістичний підхід, забезпечуючи максимальну гнучкість у побудові серверної архітектури. Однак відсутність жорсткої структури призводить до необхідності самостійної організації проєкту, що у великих системах може ускладнювати підтримку коду та вимагати додаткових зусиль для стандартизації компонентів.

Порівняно з ними Laravel [15] виступає більш збалансованим рішенням. Фреймворк надає готові механізми маршрутизації, автентифікації та валідації даних, а також інтегровану ORM Eloquent для роботи з базою даних. Використання архітектурного патерну MVC забезпечує чітке розділення бізнес-логіки, представлення та шару доступу до даних. Додатковою перевагою є зріла екосистема, детальна документація та активна спільнота розробників, що значно спрощує впровадження складного функціоналу та прискорює процес розробки.

У табл. 2.1 наведено узагальнене порівняння розглянутих серверних фреймворків.

Таблиця 2.1 – Порівняння серверних фреймворків

Критерій	Laravel	Django	Express.js
Мова програмування	PHP	Python	JavaScript
Архітектурний патерн	MVC	MTV	Відсутній
Вбудована автентифікація	Так	Так	Ні
ORM	Eloquent	Django ORM	Sequelize
Поріг входження	Низький	Середній	Середній
Документація	Детальна	Детальна	Помірна
Швидкість розробки	Висока	Висока	Середня

Laravel обрано як основний фреймворк завдяки його комплексності, зручності у роботі з реляційними базами даних, вбудованим інструментам безпеки та високій швидкості розробки порівняно з альтернативами.

Для зберігання фінансових даних було розглянуто три СУБД: MySQL, PostgreSQL та MongoDB.

PostgreSQL [16, 17] є потужною об'єктно-реляційною СУБД із розширеною підтримкою складних запитів та типів даних. Однак для завдань даного проєкту її додаткові можливості є надлишковими, а налаштування потребує більших зусиль порівняно з MySQL.

MongoDB [14, 18], своєю чергою, належить до класу документоорієнтованих NoSQL-рішень і ефективно застосовується для роботи з неструктурованими або слабко структурованими даними. Однак специфіка предметної області фінансового обліку передбачає чітко визначену реляційну модель із взаємозв’язками між сутностями, контролем цілісності та транзакційністю. Саме тому використання NoSQL-підходу в даному випадку є менш доцільним, оскільки воно ускладнює забезпечення строгих обмежень цілісності даних.

MySQL є найбільш практичним вибором для даного проєкту. Це одна з найпоширеніших реляційних СУБД, яка добре інтегрується з Laravel через ORM Eloquent, що спрощує роботу з даними на рівні коду. Додатковою перевагою є

наявність підтримки в середовищі Laragon «із коробки», а також зручні інструменти адміністрування, зокрема phpMyAdmin, що забезпечує просте керування базою даних без додаткових складних налаштувань.

У табл. 2.2 наведено узагальнене порівняння розглянутих СУБД.

Таблиця 2.2 – Порівняння систем управління базами даних

Критерій	MySQL	PostgreSQL	MongoDB
Тип	Реляційна	Реляційна	Документоорієнтована
Інтеграція з Laravel	Вбудована	Вбудована	Через пакет
Складність налаштування	Низька	Середня	Низька
Підтримка транзакцій	Так	Так	Обмежена
Підходить для фін. даних	Так	Так	Ні
Інструмент адміністрування	phpMyAdmin	pgAdmin	MongoDB Compass

У результаті порівняльного аналізу як основну систему керування базами даних було обрано MySQL. Таке рішення пояснюється поєднанням практичності, стабільності та низького порогу входження під час налаштування середовища розробки.

Для організації локального середовища розробки розглянуто три популярні підходи: Laragon, XAMPP та Docker.

XAMPP [19] є класичним інструментом для локального розгортання вебзастосунків, однак у практичному використанні він демонструє низку обмежень. Зокрема, запуск сервісів відбувається повільніше, інтерфейс управління виглядає застарілим, а налаштування віртуальних хостів потребує додаткових ручних конфігурацій, що ускладнює робочий процес.

Docker [20], навпаки, забезпечує повну ізоляцію середовища та високу відтворюваність конфігурацій, що робить його особливо ефективним у командній розробці та під час розгортання систем у продакшн-середовищі. Використання

контейнеризації потребує глибшого розуміння принципів роботи Docker, а також має вищий поріг входження для розробників, які раніше не працювали з подібними технологіями.

Натомість Laragon позиціонується як сучасне та легке середовище для Windows, яке поєднує Apache, PHP та MySQL у єдиному пакеті. Його ключовою перевагою є швидкий запуск сервісів – фактично за кілька секунд – а також проста конфігурація, що дозволяє оперативно розпочати розробку без значних витрат часу на налаштування інфраструктури.

Узагальнене порівняння розглянутих систем управління базами даних наведено у табл. 2.3.

Таблиця 2.3 – Порівняння систем управління базами даних

Критерій	Laragon	XAMPP	Docker
Платформа	Windows	Кросплатформний	Кросплатформний
Швидкість запуску	Висока	Середня	Середня
Простота налаштування	Висока	Середня	Низька
Автоматичні домени	Так	Ні	Ні
Поріг входження	Низький	Низький	Високий
Підходить для Laravel	Так	Так	Так

У результаті аналізу середовищ розробки як основне робоче середовище було обрано Laragon. Вибір зумовлений його орієнтацією на швидке розгортання локальних вебпроєктів, мінімальні вимоги до початкового налаштування та зручній інтеграції з Laravel і MySQL без додаткового конфігурування.

Для реалізації користувацького інтерфейсу вебзастосунку було розглянуто три підходи: Blade з Vanilla JS, Vue.js та React.

React [21] орієнтований на створення високодинамічних односторінкових застосунків (SPA) і забезпечує гнучку компонентну архітектуру. Його використання передбачає побудову окремого API-шару для взаємодії із серверною

частиною, що суттєво ускладнює загальну структуру проєкту та підвищує вимоги до координації між фронтендом і бекендом.

Vue.js [22] є більш легким та поступово впроваджуваним фреймворком, який добре інтегрується з екосистемою Laravel. Однак навіть у цьому випадку виникає потреба у додатковому налаштуванні збірки, маршрутизації та стану застосунку, що додає архітектурної складності порівняно з серверно-орієнтованим підходом.

Натомість використання Blade у поєднанні з Vanilla JS дозволяє формувати інтерфейс безпосередньо на стороні сервера, генеруючи готові HTML-сторінки на основі даних із контролерів.

Узагальнене порівняння підходів до побудови інтерфейсу користувача наведено у табл. 2.4.

Таблиця 2.4 – Порівняння підходів до реалізації інтерфейсу

Критерій	Blade + Vanilla JS	Vue.js	React
Інтеграція з Laravel	Вбудована	Через пакет	Через API
Складність архітектури	Низька	Середня	Висока
Швидкість розробки	Висока	Середня	Низька
Потреба в API	Ні	Частково	Так
Поріг входження	Низький	Середній	Високий
Підходить для MVC	Так	Частково	Ні

Blade у поєднанні з Vanilla JS обрано як найбільш органічний підхід для Laravel-проєкту, що забезпечує просту архітектуру, мінімальну кількість залежностей та швидку розробку інтерфейсу.

За результатами проведеного порівняльного аналізу сформовано оптимальний технологічний стек, що дозволяє досягти поставленої мети. В результаті обрано сучасні, сумісні між собою технології, які забезпечать ефективну реалізацію всіх функціональних модулів вебзастосунку.

2.2 Технології розробки системи

До складу технологічного стеку вебзастосунку для обліку особистих фінансів входить набір взаємопов’язаних програмних засобів, які забезпечують реалізацію серверної логіки, збереження даних, побудову інтерфейсу користувача та підтримку допоміжного функціоналу. Кожен із обраних компонентів виконує чітко визначену роль у загальній архітектурі системи та інтегрується з іншими частинами без конфліктів і надлишкової складності.

Laravel [23] – серверний PHP-фреймворк, обраний як основа серверної частини застосунку. Його використання для розробки систем планування бюджету та обліку витрат підтверджується сучасними дослідженнями та практичними проєктами. Фреймворк реалізує архітектурний патерн MVC, що забезпечує чітке розділення логіки, представлення та даних. Вбудовані інструменти маршрутизації, автентифікації, валідації та ORM Eloquent дозволяють швидко реалізовувати складний функціонал без необхідності підключення сторонніх бібліотек.

MySQL [24] – реляційна система управління базою даних, у якій зберігаються всі дані застосунку: облікові записи користувачів, фінансові операції, категорії, рахунки та бюджетні ліміти. Реляційна модель даних забезпечує цілісність інформації та ефективне виконання запитів із використанням зв'язків між таблицями, що є особливо важливим для фінансових застосунків із складною структурою даних.

Laragon [25] – сучасне локальне середовище розробки для операційної системи Windows, що включає вебсервер Apache, інтерпретатор PHP та сервер баз даних MySQL. Відзначається високою швидкістю запуску, автоматичним генеруванням локальних доменів для проєктів та зручним графічним інтерфейсом управління. Використання Laragon дозволяє розгорнути повноцінне середовище розробки без ручного налаштування кожного компонента окремо.

phpMyAdmin [26] – графічний вебінтерфейс для адміністрування бази даних MySQL. Надає зручні засоби для перегляду та редагування структури таблиць,

виконання SQL-запитів, імпорту й експорту даних. Використовується на етапі розробки для проєктування схеми бази даних та перевірки коректності збережених даних.

Blade [27] – вбудований шаблонізатор Laravel, що забезпечує генерацію HTML-сторінок на стороні сервера з використанням даних, переданих із контролерів. Підтримує директиви для умовного виведення, циклів та підключення компонентів, що дозволяє будувати динамічний інтерфейс без надлишкових залежностей.

Vanilla JavaScript [28] – використовується для реалізації клієнтської логіки застосунку: відкриття модальних вікон, фільтрації операцій, контекстних меню та взаємодії з сервером через AJAX-запити. Використання чистого JavaScript без додаткових фреймворків зменшує кількість залежностей і спрощує архітектуру фронтенду.

Carbon [29] – PHP-бібліотека для роботи з датами та часом, вбудована в Laravel. Використовується для групування фінансових операцій за датами: визначення приналежності операції до поточного дня, вчорашнього дня або більш раннього періоду, що забезпечує зручне відображення історії транзакцій.

Висновки до розділу 2

За результатами порівняльного аналізу альтернативних інструментів сформовано обґрунтований вибір технологій для кожного компонента вебзастосунку. Такий підхід дозволив забезпечити узгодженість між усіма частинами системи та сформувати цілісний стек розробки, орієнтований на реалізацію функціональних вимог і подальшу масштабованість проєкту.

Основою серверної частини обрано Laravel, що зумовлено реалізацією архітектурного патерну MVC, наявністю вбудованих механізмів автентифікації та використанням ORM Eloquent для роботи з базою даних. Як систему збереження даних визначено MySQL, оскільки її реляційна модель повністю відповідає

структурі фінансових даних і забезпечує ефективну роботу зі зв’язками між сутностями, а також має нативну інтеграцію з обраним фреймворком.

Для організації локального середовища розробки використано Laragon, яке дозволяє швидко розгорнути необхідні компоненти системи з мінімальними витратами на конфігурацію. Адміністрування бази даних на етапі розробки здійснюється за допомогою phpMyAdmin, що спрощує роботу зі структурою таблиць і перевірку коректності збережених даних.

Користувацький інтерфейс реалізовано із застосуванням шаблонізатора Laravel Blade у поєднанні з Vanilla JavaScript та бібліотекою Carbon, що в сукупності формує узгоджений та функціональний фронтенд-стек.

Описано функціональні модулі системи: реєстрацію та автентифікацію, особистий кабінет, облік доходів і витрат, статистику та графіки, управління бюджетними лімітами, імпорт банківських виписок і підтримку мультивалютності. Реалізовані на поточному етапі сторінки застосунку підтверджують відповідність обраного стеку функціональним вимогам системи.

3 РОЗРОБКА ВЕБЗАСТОСУНКУ ТА АНАЛІЗ ОТРИМАНИХ РЕЗУЛЬТАТІВ

3.1 Аналіз вхідних даних та структура вебзастосунку

У цьому розділі здійснено детальний аналіз архітектурних рішень, вхідних даних та функціональної структури вебзастосунку фінансового трекера. Систему реалізовано на основі технологічного стеку, обґрунтованого у другому розділі: Laravel (PHP) – як серверний фреймворк за патерном MVC; MySQL – як реляційна СУБД для зберігання фінансових даних; Blade – як шаблонізатор для формування серверних HTML-сторінок; Vanilla JavaScript – для клієнтської інтерактивності; Laragon – як локальне середовище розробки.

Система обробляє різноманітні вхідні дані, що надходять переважно від користувача через клієнтський інтерфейс, а також отримуються із зовнішніх запитів до серверної частини. Основні типи вхідних даних наведено нижче.

1. Облікові дані користувача – email та пароль; пароль зберігається у хешованому вигляді завдяки вбудованому в Laravel механізму bcrypt через фасад Hash.

2. Параметри фінансової транзакції – сума, тип операції (дохід/витрата), категорія, рахунок, дата та опис.

3. Бюджетні ліміти – місячна максимальна сума витрат для конкретної категорії.

4. HTTP-запити (GET/POST/PUT/DELETE) – маршрутизація через Laravel Router до відповідних методів контролерів.

5. CSV-файли банківських виписок – для пакетного імпорту транзакцій з Monobank та ПриватБанку.

Перелік та характеристики основних вхідних даних системи наведено у табл. 3.1.

Таблиця 3.1 – Приклади вхідних даних

Поле	Тип	Опис
email	рядок	Електронна адреса (логін) користувача.
password	рядок (хеш)	Пароль, хешований через bcrypt.
amount	десяткове число	Сума фінансової операції.
type	enum (income/expense)	Тип операції: дохід або витрата.
category_id	ціле число (FK)	Зовнішній ключ до таблиці categories.
account_id	ціле число (FK)	Зовнішній ключ до таблиці accounts.
date	дата (DATE)	Дата здійснення транзакції.
description	рядок (nullable)	Коментар до операції (необов'язковий).
currency	рядок (ISO 4217)	Код валюти (UAH, USD, EUR тощо).
budget_limit	десяткове число	Місячний ліміт витрат за категорією.

Архітектура розробленого вебзастосунку базується на клієнт-серверній моделі з чітким розділенням відповідальності між рівнями системи (див. рис. 3.1). На стороні клієнта відбувається формування та надсилання HTTP-запитів до серверної частини, що ініціюють виконання відповідних бізнес-операцій. Серверна логіка реалізована засобами Laravel [14, 30], де вхідні запити обробляються контролерами, які виступають координуючим елементом між представленням і моделями даних. Контролери взаємодіють із моделями, забезпечуючи доступ до даних у MySQL та виконання необхідних операцій над ними. Отримані результати передаються до Blade-шаблону, який формує фінальну HTML-відповідь для клієнтської частини.

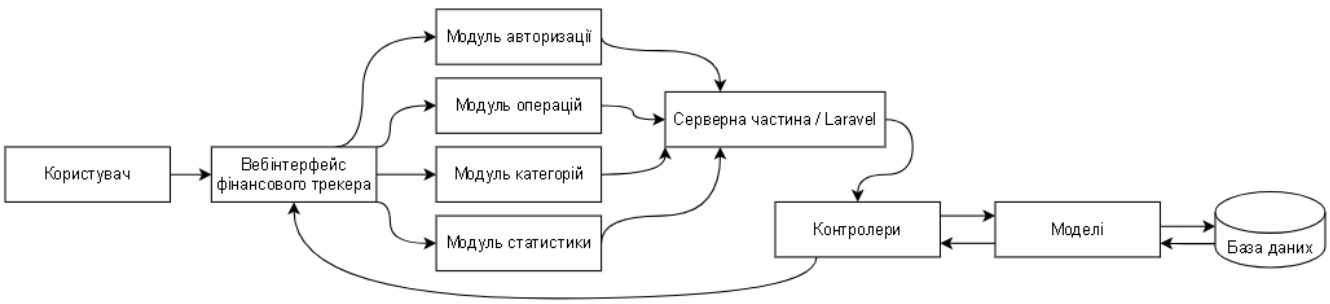

Рисунок 3.1 – Загальна архітектура вебзастосунку фінансового трекера

Структура вебзастосунку охоплює набір ключових компонентів, сформованих відповідно до архітектурного патерну MVC. До них належать маршрутизатор (routes/web.php), контролери (app/Http/Controllers), моделі (app/Models), Blade-шаблони (resources/views) та міграції бази даних (database/migrations). Така організація коду забезпечує чіткий розподіл відповідальності між компонентами, підвищує модульність системи та спрощує її подальший супровід.

Реляційна схема бази даних включає п'ять основних таблиць: users, operations, categories та accounts. Між ними встановлено зв'язки типу «один до багатьох» (1:N) через зовнішні ключі, що забезпечує цілісність даних на рівні СУБД. Детальна структура таблиць та їх взаємозв’язки наведені у табл. 3.2.

Таблиця 3.2 – Структура бази даних вебзастосунку

Таблиця	**Основні поля**	**Призначення**	**Зв’язки**
users	id, name, email, password, avatar_path, currency, theme, monthly_budget, notifications_enabled, remember_token, created_at, updated_at	Облікові записи користувачів	1:N з operations, accounts, categories
operations	id, user_id, account_id, category_id, name, amount, type, icon, method, date, color	Фінансові операції (доходи і витрати)	N:1 з users, accounts, categories
categories	id, user_id, name, type, amount, icon, color	Категорії доходів і витрат	N:1 з users; 1:N з operations
accounts	id, user_id, name, balance, type, icon, color	Фінансові рахунки користувача	N:1 з users; 1:N з operations

Таблиця users є центральною: усі інші таблиці посилаються на неї через поле

user_id. Це забезпечує ізоляцію даних між різними користувачами системи – кожен користувач бачить лише власні транзакції, категорії, рахунки та бюджети. Таблиця operations є найбільш навантаженою, оскільки зберігає всі фінансові операції та посилається одночасно на accounts і categories, що дозволяє ефективно агрегувати витрати як за рахунками, так і за категоріями.

3.2 Проєктування та моделювання вебзастосунку

Проєктування вебзастосунку фінансового трекера виконано із застосуванням засобів уніфікованої мови моделювання UML. Розроблені діаграми дозволяють наочно описати структуру системи, взаємодію між її компонентами та логіку обробки даних. Перелік використаних типів UML-діаграм наведено у табл. 3.3.

Таблиця 3.3 – Типи UML-діаграм, використаних при проєктуванні

Тип діаграми	Зміст	Призначення
Діаграма варіантів використання	Відображає взаємодію акторів (користувач, система) з основними функціями застосунку	Визначення функціональних вимог до системи
Діаграма класів	Описує структуру моделей даних та їх зв'язки (User, Operation, Category, Account, Budget)	Проєктування структури бази даних та об'єктної моделі
ER-діаграма	Графічне представлення реляційної схеми бази даних з усіма таблицями та ключами	Проєктування та документування структури БД

Діаграма варіантів використання описує взаємодію єдиного актора – автентифікованого користувача – із системою. Основні варіанти використання: реєстрація та авторизація, управління транзакціями, категоріями та рахунками, встановлення бюджетних лімітів, перегляд статистики та імпорт банківських виписок.

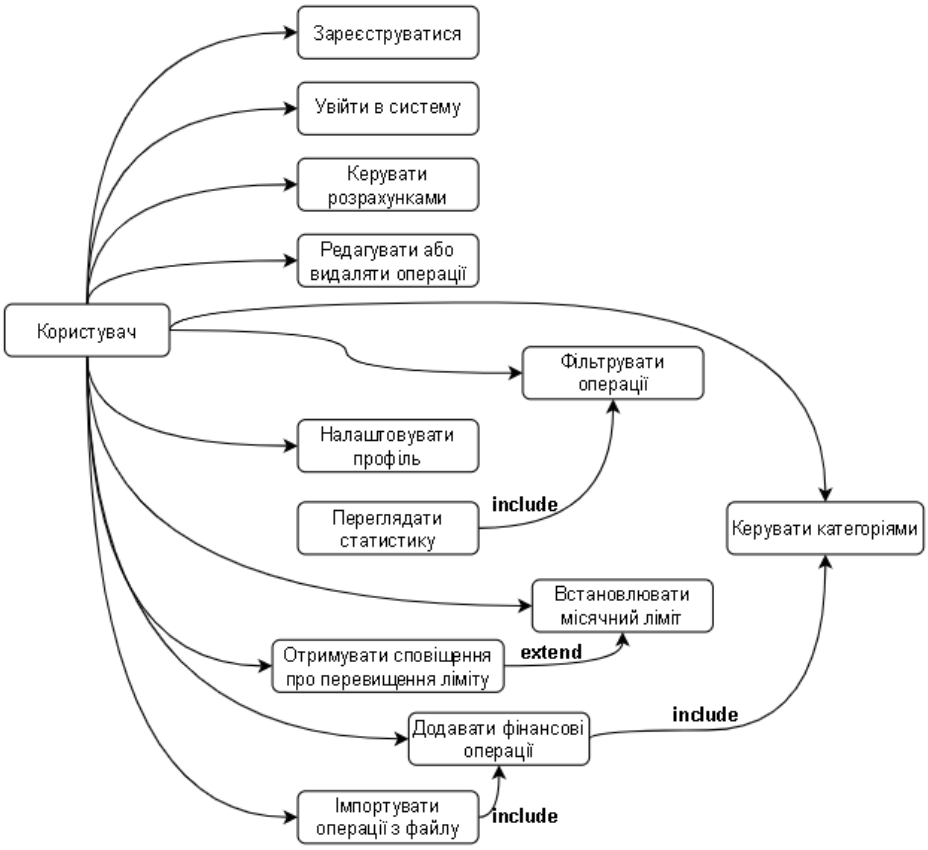

Рисунок 3.2 – Діаграма варіантів використання вебзастосунку

На основі діаграми описано сценарій варіантів використання для Користувача.

1. Реєстрація та автентифікація.

Передумови: користувач має доступ до інтернету та браузера.

Основний сценарій:

– користувач відкриває сторінку реєстрації та вводить email і пароль;

– Laravel валідує дані через клас FormRequest і хешує пароль через bcrypt;

– після успішної реєстрації система автоматично авторизує користувача та перенаправляє до особистого кабінету;

– при повторному вході – перевірка через Auth::attempt() із вбудованого механізму Laravel.

Альтернативний варіант: у разі некоректних даних або вже зареєстрованого email – відображення помилок валідації безпосередньо у формі.

2. Додавання та редагування транзакцій.

Передумови: користувач авторизований.

Основний сценарій:

– користувач натискає кнопку «Додати транзакцію» – відкривається модальне вікно;

– обирає тип (дохід/витрата), вводить суму, категорію, рахунок та опис;

– після підтвердження POST-запит надходить до OperationController@store;

– Eloquent зберігає запис у таблицю operations MySQL, баланс рахунку оновлюється автоматично;

– сторінка оновлюється – транзакція відображається в хронологічному списку з групуванням за датами через Carbon.

3. Управління бюджетними лімітами.

Передумови: користувач авторизований, є принаймні одна категорія витрат.

Основний сценарій:

– користувач переходить до розділу «Бюджетні ліміти» та встановлює місячний ліміт;

– система зберігає ліміт із прив’язкою до поточного місяця;

– на панелі відображається індикатор виконання бюджету (прогрес-бар);

– при перевищенні 80% ліміту – індикатор змінює колір на жовтий, при перевищенні 100% – на червоний.

4. Перегляд статистики та звітів.

Передумови: користувач авторизований, є внесені транзакції.

Основний сценарій:

– користувач переходить до розділу «Статистика» та обирає часовий діапазон;

– Laravel виконує агрегаційні SQL-запити;

– Blade-шаблон передає дані до Chart.js [31] – відображаються кругова діаграма витрат і стовпчастий графік витрат за категоріями.

5. Імпорт банківських виписок.

Передумови: користувач авторизований, має CSV-файл виписки Monobank або ПриватБанку.

Основний сценарій:

– користувач завантажує CSV-файл через форму імпорту;

– Laravel парсить файл, визначає формат банку та автоматично маппить колонки до полів транзакцій;

– система пропонує переглянути та підтвердити список розпізнаних транзакцій;

– після підтвердження транзакції масово зберігаються в.

6. Управління рахунками та мультивалютність.

Передумови: користувач авторизований.

Основний сценарій:

– користувач у розділі «Валюта» обирає можливу валюту (UAH, USD, EUR);

– баланс кожного рахунку автоматично перераховується;

– загальний баланс відображається у гривневому еквіваленті з урахуванням поточного курсу валют.

Діаграма класів відображає об'єктну модель системи, що відповідає структурі Eloquent-моделей Laravel. Клас User є кореневим і пов'язаний із класами Operation, Category та Account через відношення «один до багатьох». Клас Operation додатково пов'язаний з Category та Account, що відповідає реляційній структурі бази даних.

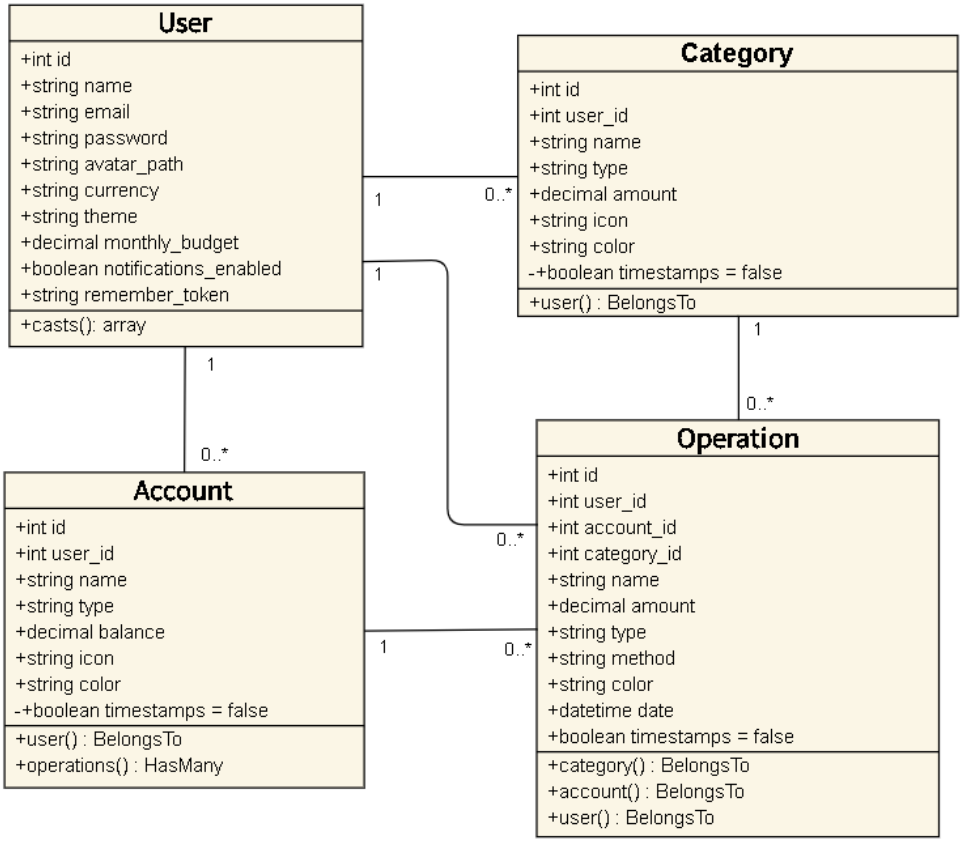

Рисунок 3.3 – Діаграма класів

Діаграма класів відображає основну структуру вебзастосунку персонального фінансового трекера та показує, як між собою пов’язані головні сутності предметної області. У системі виділено чотири основні класи: User, Account, Category та Operation. Вони відповідають користувачу, фінансовому рахунку, категорії операцій та окремій фінансовій операції відповідно.

Центральним класом діаграми є User, оскільки всі фінансові дані в системі належать конкретному користувачу. Клас містить основні атрибути облікового запису: ім’я, електронну пошту, пароль, шлях до аватара, валюту, тему інтерфейсу, місячний бюджет і налаштування сповіщень. Також у моделі передбачене службове поле токен запам’ятовування сесії. Користувач може мати багато рахунків, категорій і фінансових операцій, що відображено зв’язками типу «один до багатьох».

Клас Account описує фінансовий рахунок користувача. До його атрибутів

належать ідентифікатор користувача, назва рахунку, тип рахунку, баланс, іконка та колір для візуального відображення в інтерфейсі. Кожен рахунок належить одному користувачу, але один користувач може створити декілька рахунків, наприклад банківську картку або заощадження. Крім того, рахунок може бути пов’язаний з багатьма операціями, оскільки кожна витрата або дохід може виконуватися з певного рахунку.

Клас Category використовується для групування фінансових операцій за призначенням. Він містить назву категорії, тип, суму, іконку та колір. Категорії також належать конкретному користувачу, що дозволяє кожному користувачу мати власну структуру обліку доходів і витрат. Наприклад, користувач може створити категорії «Продукти», «Транспорт», «Зарплата» або «Розваги». Одна категорія може використовуватися в багатьох операціях, але кожна операція належить лише до однієї категорії або може не мати категорії, якщо вона не була вказана.

Клас Operation є основною сутністю для збереження фінансових дій користувача. Він містить посилання на користувача, рахунок і категорію, а також назву операції, суму, тип, спосіб здійснення, колір і дату. Тип операції дає змогу відрізняти доходи від витрат, а поле method може використовуватися для збереження способу оплати. Кожна операція обов’язково пов’язана з користувачем.

Загалом діаграма класів демонструє логічну модель системи, у якій користувач є власником усіх фінансових об’єктів. Рахунки відображають джерела або місця зберігання коштів, категорії забезпечують класифікацію фінансових подій, а операції фіксують конкретні доходи та витрати. Така структура дозволяє реалізувати основні функції фінансового трекера: ведення обліку коштів, аналіз витрат, групування операцій за категоріями, перегляд балансу рахунків і формування персональної фінансової статистики.

ER-діаграма бази даних використовується для наочного подання структури даних інформаційної системи та зв’язків між її основними сутностями. Для розроблюваного персонального фінансового трекера ER-діаграма відображає таблиці бази даних, їхні атрибути, первинні та зовнішні ключі, а також типи

зв’язків між таблицями. Така діаграма дозволяє краще зрозуміти логіку збереження фінансової інформації користувача та забезпечує основу для подальшої реалізації бази даних у системі керування базами даних.

На рис. 3.4 наведено ER-діаграму бази даних системи, яка складається з чотирьох основних таблиць: users, accounts, categories та operations.

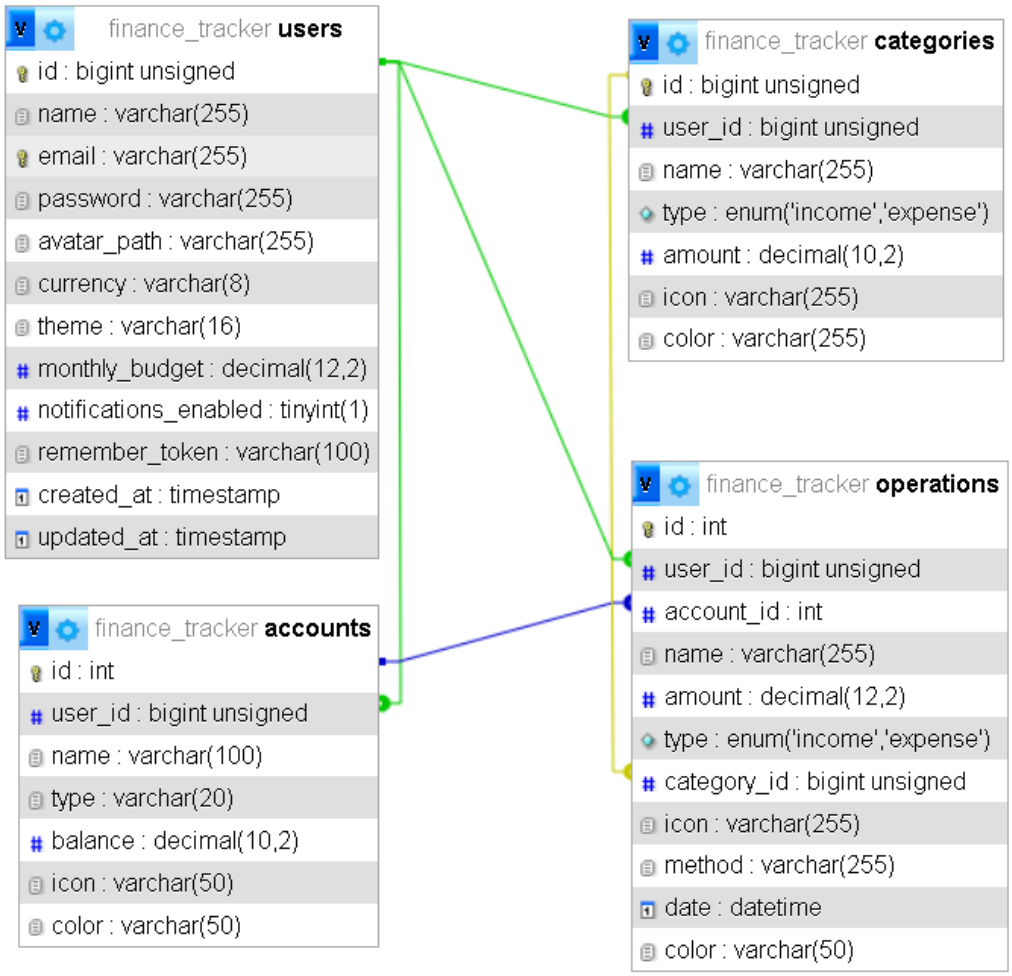

Рисунок 3.4 – ER діаграма

Центральною таблицею є users, яка зберігає дані про зареєстрованих користувачів системи. Вона містить первинний ключ id, ім’я користувача, електронну пошту, пароль, шлях до аватара, валюту, тему інтерфейсу, місячний бюджет, налаштування сповіщень, токен запам’ятовування сесії, а також службові

поля created_at і updated_at. Саме ця таблиця є основою для персоналізації даних, оскільки кожен рахунок, категорія та операція належать певному користувачу.

Таблиця accounts призначена для збереження фінансових рахунків користувача. Вона містить ідентифікатор рахунку id, зовнішній ключ user_id, назву рахунку, тип, баланс, іконку та колір. Зв’язок між users і accounts має тип «один до багатьох»: один користувач може мати багато рахунків, але кожен рахунок належить лише одному користувачу. Це дозволяє вести облік коштів окремо для різних джерел, наприклад банківської картки, готівки або накопичувального рахунку.

Таблиця categories використовується для класифікації фінансових операцій. Вона містить первинний ключ id, зовнішній ключ user_id, назву категорії, тип категорії, суму, іконку та колір. Поле type може набувати значень income або expense, що дає змогу відокремлювати категорії доходів від категорій витрат. Зв’язок між users і categories також має тип «один до багатьох», оскільки один користувач може створювати багато власних категорій.

Основною таблицею для фіксації фінансової активності є operations. Вона зберігає окремі фінансові операції користувача: доходи або витрати. Таблиця містить первинний ключ id, зовнішні ключі user_id, account_id і category_id, назву операції, суму, тип, іконку, спосіб здійснення операції, дату та колір. Поле user_id пов’язує операцію з конкретним користувачем, account_id визначає рахунок, з якого виконано операцію або на який зараховано кошти, а category_id вказує категорію, до якої належить операція.

Між таблицями accounts і operations встановлено зв’язок «один до багатьох»: один рахунок може використовуватися у багатьох операціях, але кожна операція пов’язана з одним рахунком. Аналогічно, між categories і operations існує зв’язок «один до багатьох»: одна категорія може об’єднувати багато операцій, проте кожна операція належить одній категорії. Таблиця operations також напряму пов’язана з users, що забезпечує швидке отримання всіх операцій конкретного користувача та гарантує ізоляцію фінансових даних між різними обліковими записами.

ER-діаграма демонструє реляційну структуру бази даних, у якій таблиця users виступає головною сутністю, а таблиці accounts, categories і operations деталізують фінансову інформацію користувача. Така модель забезпечує цілісність даних, підтримує персоналізований облік фінансів і дозволяє ефективно реалізувати функції аналізу витрат, доходів, балансу рахунків та статистики за категоріями.

Висновки до розділу 3

У третьому розділі виконано комплексне проєктування вебзастосунку системи обліку особистих фінансів відповідно до обраного технологічного стеку Laravel + MySQL + Blade + Vanilla JS.

Визначено повний перелік вхідних даних системи: облікові дані користувача, параметри фінансових транзакцій (сума, тип, категорія, рахунок, дата, валюта), бюджетні ліміти та CSV-файли банківських виписок. Структуру системи побудовано за MVC-патерном із чітким розподілом відповідальності між моделями Eloquent, контролерами Laravel та Blade-шаблонами.

Розроблено UML-діаграми варіантів використання для двох акторів. Для Користувача описано шість ключових сценаріїв: реєстрація та автентифікація через вбудований механізм Laravel Auth, додавання й редагування транзакцій через Eloquent ORM, управління бюджетними лімітами з індикаторами виконання, перегляд статистики через Chart.js [32], імпорт CSV-виписок з Monobank та ПриватБанку, а також управління мультивалютними рахунками.

Спроєктовано реляційну схему бази даних MySQL, що включає 4 нормалізовані таблиці: users, accounts, categories та operations. На ключових полях встановлено складені індекси для оптимізації агрегаційних запитів. Усі міграції реалізовано засобами Laravel Migrations.

4 ПРОГРАМНА РЕАЛІЗАЦІЯ ТА РОЗРОБКА ДОКУМЕНТАЦІЇ

4.1 Опис програмної реалізації

Для розробки вебзастосунку фінансового трекера використано інтегроване середовище розробки Visual Studio Code (VS Code) – безкоштовний редактор коду з відкритим вихідним кодом, розроблений компанією Microsoft. VS Code є одним із найбільш поширених інструментів серед веброзробників завдяки поєднанню легковісності, широкої функціональності та підтримки більшості сучасних мов програмування та фреймворків.

Середовище підтримує роботу з усіма технологіями, застосованими у проєкті: PHP та фреймворк Laravel, JavaScript, HTML та шаблони Blade, а також файли конфігурації формату JSON і YAML. Підсвічування синтаксису, автодоповнення коду та вбудована підтримка IntelliSense значно прискорюють написання коду та знижують кількість помилок на етапі розробки.

Після запуску середовища відкривається головне вікно редактора, яке містить робочу область для написання коду та елементи швидкого доступу до основних функцій (див. рис. 4.1).

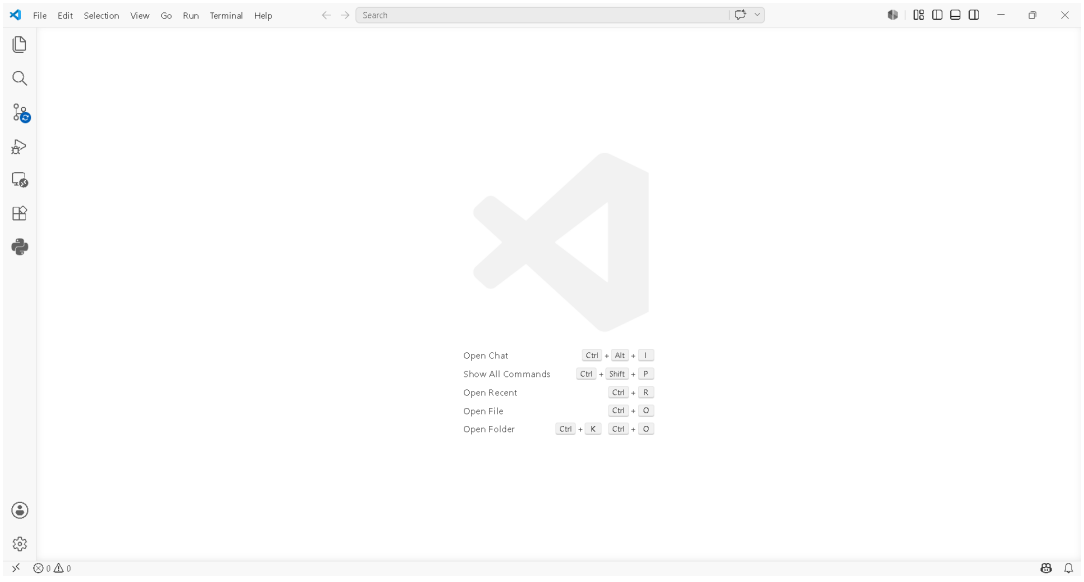

Рисунок 4.1 – Головне вікно редактору

Створимо проєкт Laravel за допомогою командного рядку:

```
composer create-project laravel/laravel personal-finance-tracker
```

Після створення проєкту у лівій панелі буде відображено його повну структуру (див. рис. 4.2).

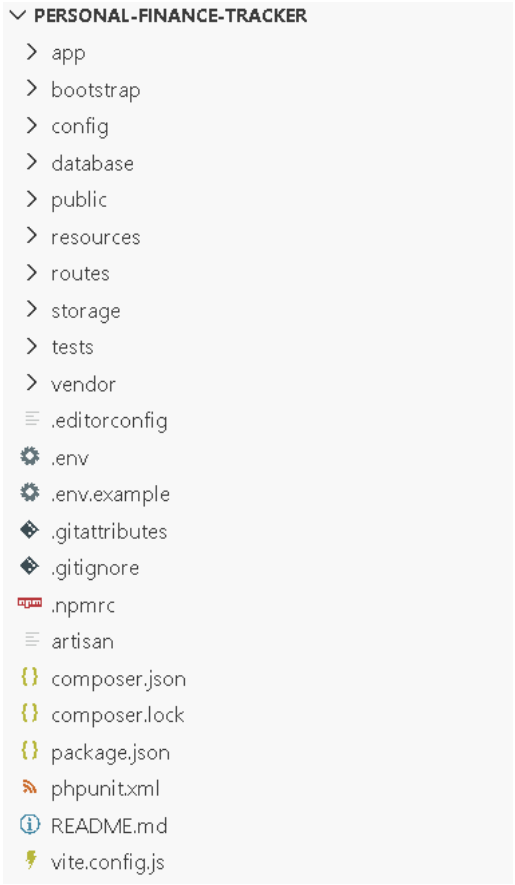

Рисунок 4.2 – Повна структура проєкту

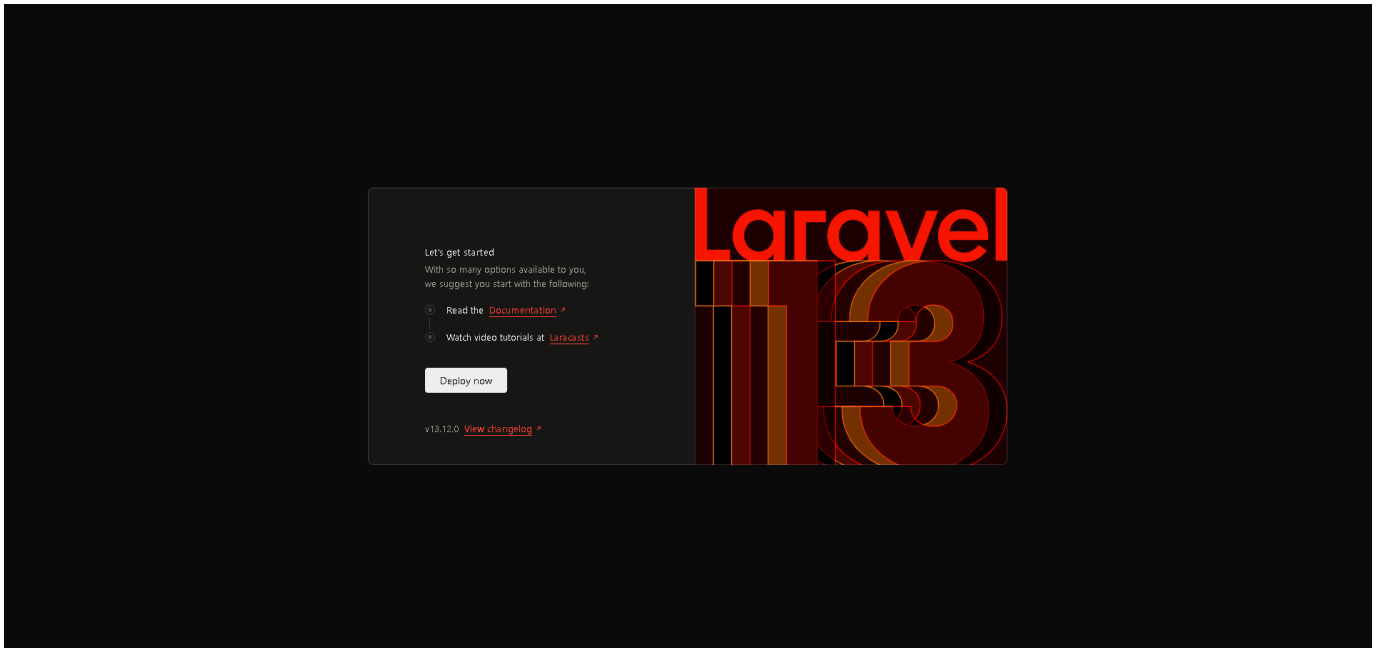

Рисунок 4.3 – Запуск проєкту в браузері

Для розширення базових можливостей редактора встановлено ряд розширень, що безпосередньо використовувались у процесі розробки:

– Laravel Extension Pack – набір розширень для роботи з Laravel, що включає підтримку маршрутів, Eloquent-моделей, Blade-шаблонів і Artisan-команд;

– PHP Intelephense – розширення для статичного аналізу PHP-коду, що забезпечує автодоповнення, навігацію між файлами та виявлення помилок без запуску проєкту;

– Laravel Blade Formatter – автоматичне форматування Blade-шаблонів відповідно до стандартів оформлення коду;

– Prettier – форматувальник коду для JavaScript, CSS та HTML, що забезпечує єдиний стиль оформлення клієнтської частини проєкту;

– Laravel Artisan – вбудований консольний інструмент фреймворку Laravel для автоматизації типових операцій під час розробки вебзастосунків.

Вбудований термінал VS Code використовувався для виконання Artisan-команд Laravel (генерація контролерів, міграцій, моделей), запуску міграцій бази даних, а також для взаємодії з пакетним менеджером Composer.

На рису. 4.4 зображено робочий простір Visual Studio Code з відкритим

файлом контролера OperationController.php.

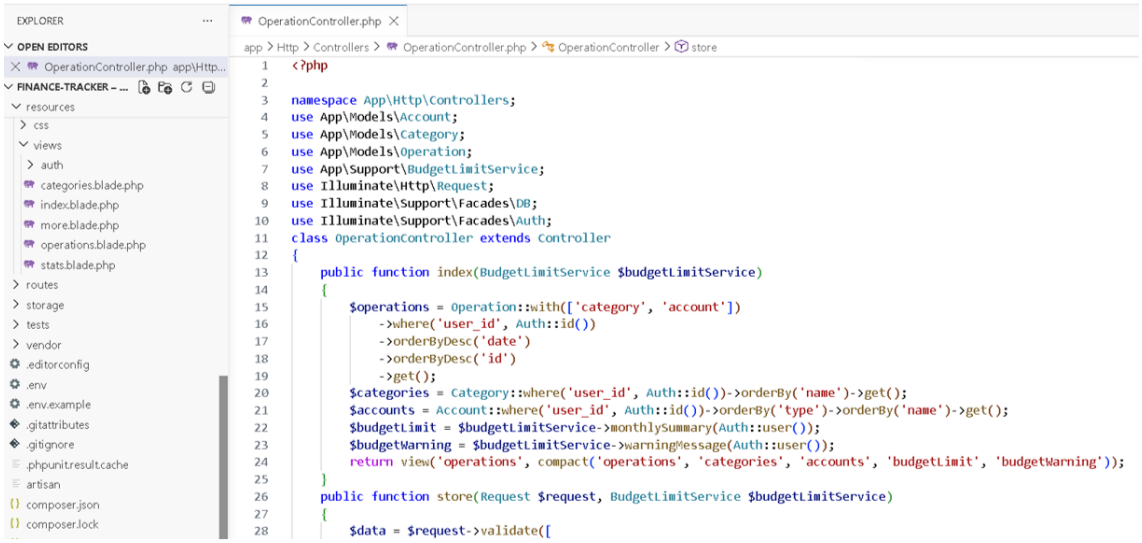

Рисунок 4.4 – Робочий простір Visual Studio Code

Використання VS Code у поєднанні з середовищем розробки Laragon забезпечило зручний та ефективний цикл розробки: редагування коду у VS Code, відображення змін у браузері через локальний домен Laragon та адміністрування бази даних через phpMyAdmin.

4.2 Структура проєкту

Архітектура вебзастосунку структурована у вигляді восьми функціональних модулів, кожен із яких реалізується окремим контролером Laravel і відповідає за чітко визначену ділянку функціоналу. Перелік і короткий опис функціональних модулів наведено у табл. 4.1.

Таблиця 4.1 – Функціональні модулі вебзастосунку

Модуль	**Функціонал**	**Контролер**
Реєстрація та автентифікація	Реєстрація нового облікового запису, вхід за email/паролем, вихід із системи, хешування паролів через bcrypt	AuthController
Облік транзакцій	Додавання, редагування, видалення фінансових операцій; фільтрація за датою, типом, категорією	OperationController
Категорії	Перегляд стандартних та створення кастомних категорій доходів і витрат; прив'язка до операцій	CategoryController
Рахунки	Управління фінансовими рахунками (готівка, карта, заощадження); автоматичний перерахунок балансу	AccountController
Бюджетні ліміти	Встановлення місячних лімітів витрат за категоріями; відстеження використання; сповіщення про перевищення	MoreController
Статистика та графіки	Кругові та стовпчасті діаграми витрат; динаміка балансу; порівняння за місяцями	StatsController
Імпорт виписок	Завантаження та парсинг CSV-файлів виписок Monobank і ПриватБанку; автоматична категоризація операцій	ImportedOperationController
Мультивалютність	Підтримка UAH, USD, EUR, PLN; автоматичний перерахунок за курсом НБУ через зовнішній API	MoreController

Модуль реєстрації та автентифікації є точкою входу до системи. При реєстрації введені дані валідуються за допомогою Laravel Validator, після чого пароль хешується через Hash::make() і новий запис зберігається у таблиці users. Вхід здійснюється через Auth::attempt(), що перевіряє облікові дані та видає сесію. На рис. 4.5 зображено сторінку реєстрації користувача.

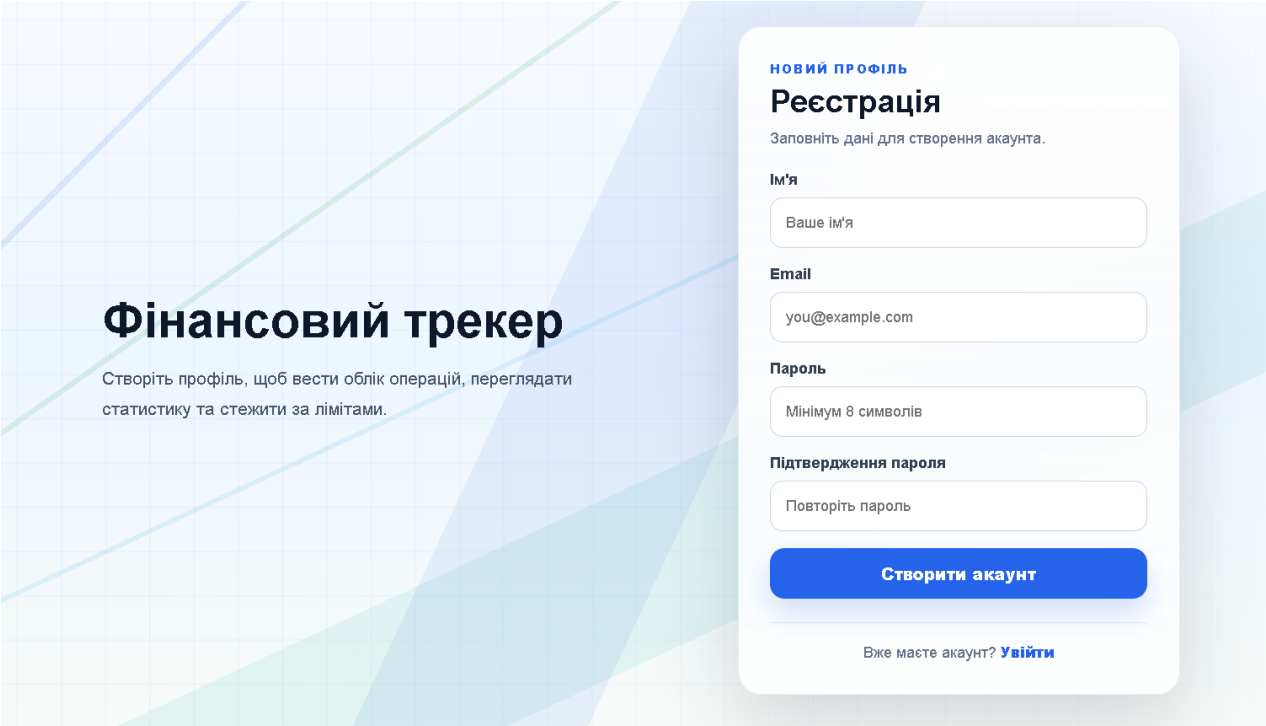

Рисунок 4.5 – Сторінка реєстрації користувача

Особистий кабінет є головною сторінкою після авторизації. AccountController агрегує дані про поточний баланс усіх рахунків, суму доходів і витрат за поточний місяць, останні п'ять транзакцій та стан бюджетних лімітів. Для визначення приналежності транзакцій до поточного/вчорашнього дня використовується бібліотека Carbon.

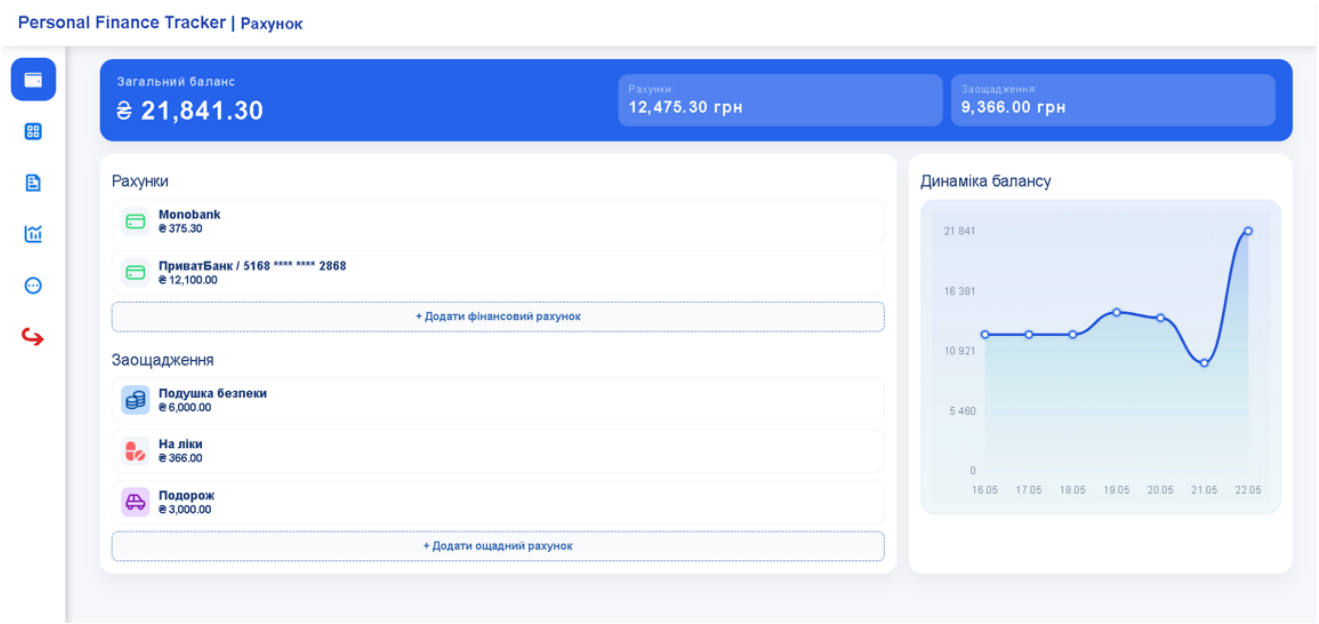

Рисунок 4.6– Головна сторінка фінансового трекера

Модуль обліку транзакцій є центральним функціональним компонентом вебзастосунку та реалізує повний CRUD-цикл для фінансових операцій. Його функціональність охоплює створення нових транзакцій через модальне вікно, редагування вже наявних записів, видалення операцій із обов'язковим підтвердженням дії, а також гнучку фільтрацію даних за типом операції, категорією та часовим інтервалом.

Під час збереження або зміни транзакції виконується автоматичне оновлення балансу відповідного рахунку: для доходних операцій значення збільшується, тоді як для витратних – зменшується. Така логіка забезпечує актуальність фінансових даних у режимі реального часу та мінімізує ризик розсинхронізації стану рахунків.

Загальний вигляд сторінки обліку фінансових операцій наведено на рис. 4.7.

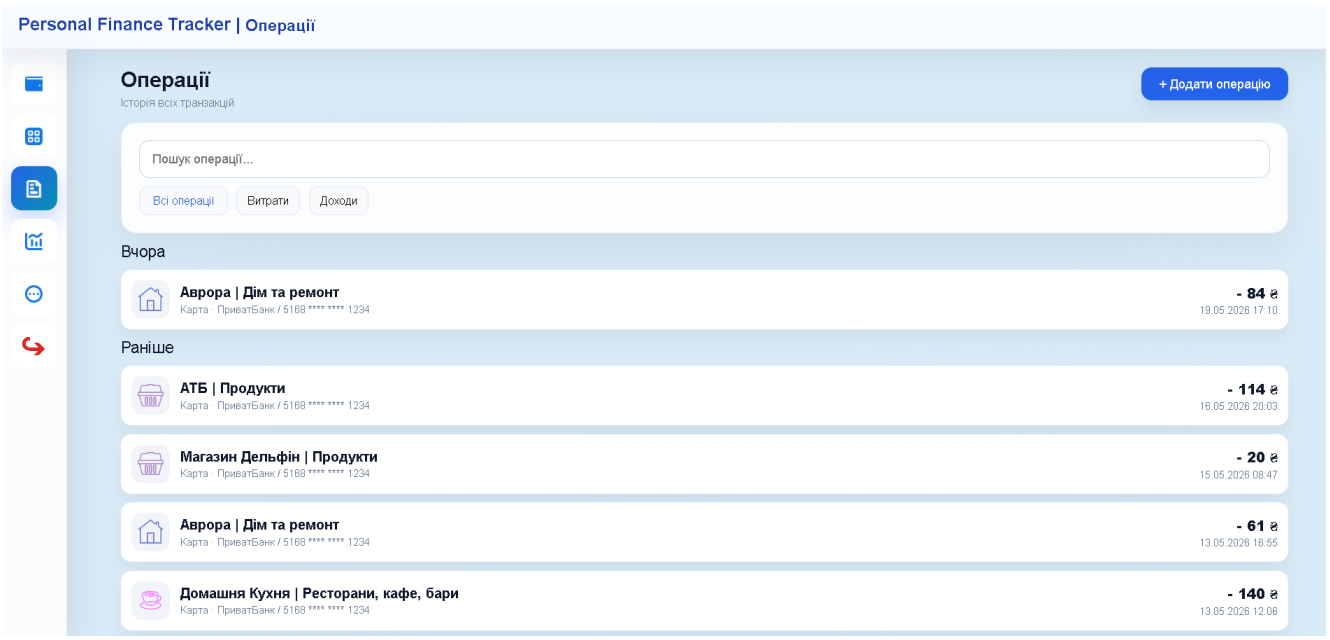

Рисунок 4.7 – Сторінка обліку фінансових операцій

Модуль статистики та графіків формує аналітичні дані про фінансову активність користувача за обраний часовий діапазон. StatisticsController агрегує транзакції та передає дані у Blade-шаблон, де за допомогою бібліотеки Chart.js будуються стовпчаста діаграма розподілу витрат за категоріями та кругова діаграма динаміки доходів і витрат.

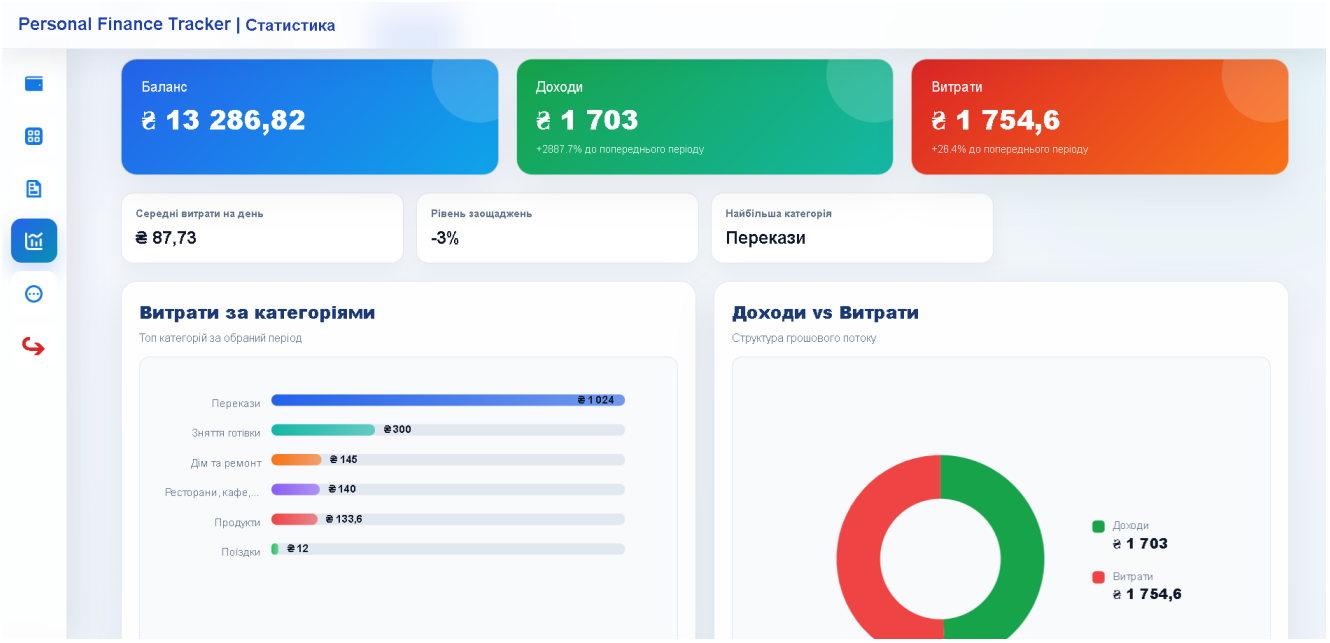

Рисунок 4.8 – Сторінка статистики з графіками витрат

Модуль бюджетних лімітів реалізовано у класі BudgetLimitService. Сервіс розраховує відсоток використання місячного ліміту шляхом зіставлення фактичних витрат із встановленим значенням. Якщо рівень перевищує 80%, система виводить попередження; при перевищенні 100% – критичне сповіщення.

Рисунок 4.9 – Управління бюджетними лімітами

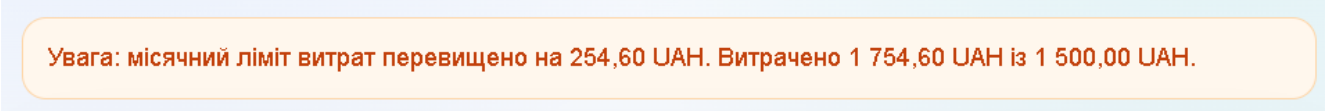

Рисунок 4.10 – Сповіщення перевищеного ліміту

Ключовий фрагмент коду:

```
$spent = Operation::where('user_id', $user->id)
    ->where('type', 'expense')
    ->whereBetween('date', [
        Carbon::now()->startOfMonth(),
        Carbon::now()->endOfMonth()
    ])->sum('amount');
$percent = $limit > 0 ? min(999, round(($spent / $limit) * 100, 1)) : 0;
```

Алгоритм виконується щоразу при завантаженні сторінки «Операції» та сторінки «Статистика». Покроковий опис наведено у табл. 4.2.

Таблиця 4.2 – Алгоритм аналізу витрат та формування повідомлень

№	Крок алгоритму	Реалізація (Laravel/PHP)
1	Отримання транзакцій поточного місяця	Operation::where(user_id)->whereMonth(date)->get()
2	Групування за категоріями	groupBy('category_id') – сума витрат по кожній категорії

Кінець таблиці 4.2

№	Крок алгоритму	Реалізація (Laravel/PHP)
3	Порівняння з лімітами	Budget::where(category_id, month)->pluck('amount') percent = spent/limit*100
4	Формування рекомендацій	percent > 100 – «перевищено ліміт»; percent > 80 – «наближається до ліміту»; інакше – норма

Модуль імпорту банківських виписок підтримує завантаження CSV-файлів у форматах Monobank та ПриватБанку. ImportedOperationController зчитує файл, визначає формат банку за структурою заголовків, парсить рядки, автоматично визначає категорію на основі опису операції та зберігає транзакції у БД. Дублікати виявляються за збігом суми, дати та опису і пропускаються.

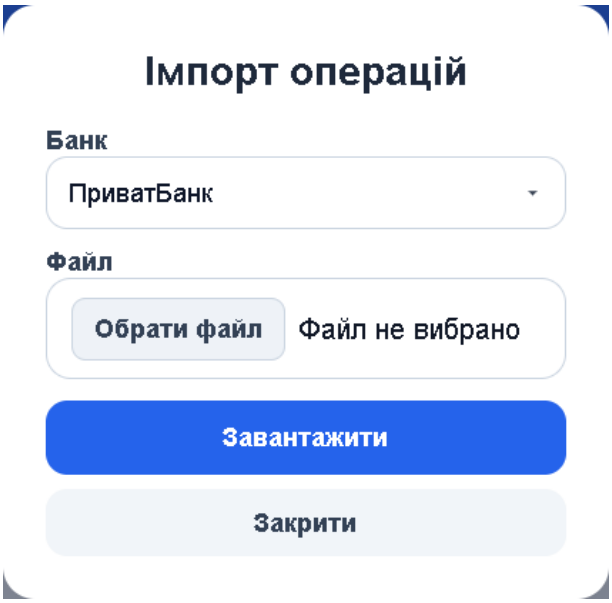

Рисунок 4.11 – Елемент імпорту банківських виписок

На рис. 4.12 зображено послідовність виконання імпорту фінансових операцій із файлів формату CSV.

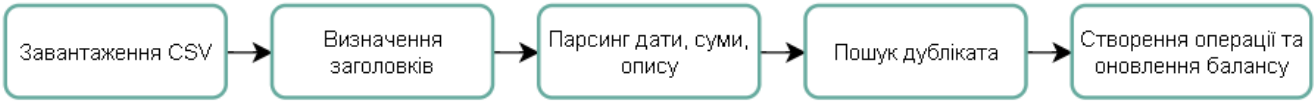

Рисунок 4.12 – Послідовність імпорту операцій банку

Першим етапом є завантаження файлу CSV. Користувач обирає файл із локального пристрою, після чого система приймає його для подальшої обробки. Підтримка двох поширених форматів підвищує зручність використання застосунку, оскільки більшість банківських сервісів і фінансових програм дозволяють експортувати операції саме у таких форматах.

Після завантаження виконується визначення заголовків таблиці. На цьому етапі система аналізує перший рядок або структуру документа та встановлює, які стовпці відповідають даті, сумі, опису, типу операції чи іншим необхідним параметрам. Це потрібно для правильного зіставлення даних із полями бази даних і зменшення кількості помилок під час імпорту.

Наступним етапом є парсинг даних, зокрема дати, суми та опису операції. Система зчитує кожен рядок файлу, перетворює значення у потрібний формат і готує їх до збереження. Наприклад, дата приводиться до формату, який використовується в базі даних, сума перетворюється на числове значення, а текстовий опис очищується від зайвих символів. На цьому етапі також може визначатися тип операції: дохід або витрата.

Після підготовки даних виконується пошук дублікатів. Цей крок необхідний для того, щоб уникнути повторного додавання однакових операцій у разі повторного імпорту одного й того самого файлу. Для перевірки можуть використовуватися такі параметри, як дата, сума, опис, рахунок і тип операції. Якщо система знаходить збіг, така операція не додається повторно.

Завершальним етапом є створення фінансових операцій і оновлення балансу рахунку. Після успішної перевірки дані зберігаються у таблиці операцій, прив’язуються до відповідного користувача та рахунку, а баланс рахунку

перераховується відповідно до імпортованих доходів або витрат. Завдяки цьому користувач одразу отримує актуальну інформацію про стан своїх фінансів.

Автоматична категоризація при імпорті CSV реалізована на основі словника ключових слів. Алгоритм аналізує опис кожної транзакції та зіставляє його з переліком ключових слів для кожної категорії.

Сервіс мультивалютності CurrencyExchangeService отримує актуальні курси НБУ через зовнішній API та кешує їх для оптимізації кількості зовнішніх запитів. Підтримуються чотири валюти: UAH, USD, EUR, PLN.

4.5 Тестування системи

Тестування розробленого вебзастосунку персонального фінансового трекера проводилося з метою перевірки коректності реалізованих функцій, відповідності системи вимогам технічного завдання, стабільності роботи основних модулів та захищеності користувацьких даних. Оскільки застосунок працює з персональною фінансовою інформацією, особлива увага приділялася правильності обробки операцій, розрахунку балансу, імпорту банківських виписок і захисту даних користувача.

Проведено функціональне тестування, яке передбачало перевірку основних сценаріїв роботи користувача із системою. До таких сценаріїв належать реєстрація нового користувача, авторизація, вихід з облікового запису, створення рахунків, додавання, редагування та видалення фінансових операцій, робота з категоріями, перегляд статистики, розрахунок бюджетних лімітів, конвертація валют та імпорт CSV-виписок.

Окрему увагу було приділено тестуванню імпорту банківських виписок. Для перевірки використовувалися реальні файли виписок Monobank та ПриватБанку. Під час тестування перевірялася можливість завантаження файлу, визначення заголовків таблиці, розпізнавання полів дати, суми та опису операції, визначення типу операції, пошук дублікатів і створення нових записів у базі даних. Такий підхід дозволив перевірити роботу системи в умовах, наближених до реального

використання.

Також було проведено тестування інтерфейсу користувача. Перевірялася коректність відображення сторінок, зручність навігації, працездатність кнопок, форм, фільтрів і повідомлень про помилки. Додатково перевірялося, чи не виникають помилки при введенні некоректних або порожніх значень у формах. Це дозволило переконатися, що система коректно реагує на типові помилки користувача.

Розроблено сценарії тестування, які наведено в табл. 4.3 – 4.21.

Таблиця 4.3 – Сценарій тестування «Реєстрація нового користувача»

ID	1
Мета	Створити обліковий запис
Передумови	Користувач не авторизований
Кроки виконання	1. Відкрити сторінку реєстрації. 2. Ввести ім’я, email і пароль. 3. Підтвердити пароль. 4. Натиснути кнопку реєстрації.
Очікуваний результат	Новий користувач створюється та виконується вхід у систему.
Статус	Успішно

Таблиця 4.4 – Сценарій тестування «Реєстрація з уже використаним email»

ID	2
Мета	Перевірити обробку дублювання email
Передумови	Email уже існує в базі даних
Кроки виконання	1. Відкрити сторінку реєстрації. 2. Ввести email, який уже зареєстрований. 3. Заповнити інші поля. 4. Натиснути кнопку реєстрації.
Очікуваний результат	Система відображає повідомлення про помилку, новий користувач не створюється.
Статус	Успішно

Таблиця 4.5 – Сценарій тестування «Авторизація з правильними даними»

ID	3
Мета	Перевірити вхід користувача з коректними обліковими даними
Передумови	Користувач зареєстрований
Кроки виконання	1. Відкрити сторінку входу. 2. Ввести правильний email і пароль. 3. Натиснути кнопку входу.
Очікуваний результат	Користувач успішно авторизується та переходить до головної сторінки застосунку.
Статус	Успішно

Таблиця 4.6 – Сценарій тестування «Авторизація з неправильними даними»

ID	4
Мета	Перевірити реакцію системи на некоректні дані входу
Передумови	Користувач зареєстрований
Кроки виконання	1. Відкрити сторінку входу. 2. Ввести правильний email і неправильний пароль. 3. Натиснути кнопку входу.
Очікуваний результат	Система відображає повідомлення про неправильні облікові дані, вхід не виконується.
Статус	Успішно

Таблиця 4.7 – Сценарій тестування «Імпорт файлу з некоректною структурою»

ID	5
Мета	Перевірити реакцію системи на файл без необхідних колонок
Передумови	Користувач авторизований
Кроки виконання	1. Завантажити файл без необхідних колонок.
Очікуваний результат	Система відображає повідомлення про помилку та не створює некоректні операції.
Статус	Успішно

Таблиця 4.8 – Сценарій тестування «Додавання фінансового рахунку»

ID	6
Мета	Перевірити створення нового рахунку користувача
Передумови	Користувач авторизований
Кроки виконання	1. Відкрити сторінку рахунків. 2. Натиснути кнопку додавання. 3. Ввести назву, баланс та обрати іконку. 4. Зберегти дані.
Очікуваний результат	Новий рахунок з’являється у списку рахунків і зберігається в базі даних.
Статус	Успішно

Таблиця 4.9 – Сценарій тестування «Редагування фінансового рахунку»

ID	7
Мета	Перевірити можливість зміни даних існуючого рахунку
Передумови	Існує хоча б один рахунок
Кроки виконання	1. Відкрити список рахунків. 2. Обрати рахунок для редагування. 3. Змінити дані. 4. Зберегти зміни.
Очікуваний результат	Дані рахунку оновлюються та коректно відображаються в інтерфейсі.
Статус	Успішно

Таблиця 4.10 – Сценарій тестування «Видалення фінансового рахунку»

ID	8
Мета	Перевірити видалення рахунку
Передумови	Існує хоча б один рахунок
Кроки виконання	1. Відкрити список рахунків. 2. Натиснути видалення. 3. Підтвердити дію.

Кінець таблиці 4.10

ID	8
Очікуваний результат	Рахунок видаляється або система блокує видалення, якщо він пов’язаний з операціями.
Статус	Успішно

Таблиця 4.11 – Сценарій тестування «Створення категорії»

ID	9
Мета	Перевірити створення категорії для групування операцій
Передумови	Користувач авторизований
Кроки виконання	1. Відкрити сторінку категорій. 2. Додати нову категорію. 3. Ввести назву та обрати іконку. 4. Зберегти.
Очікуваний результат	Категорія створюється та відображається у списку.
Статус	Успішно

Таблиця 4.12 – Сценарій тестування «Додавання операції доходу/витрати»

ID	10
Мета	Перевірити створення операції доходу/витрати та збільшення/зменшення балансу
Передумови	Існує рахунок і категорія доходу
Кроки виконання	1. Відкрити сторінку операцій. 2. Натиснути додавання операції. 3. Ввести назву, категорію, рахунок, суму, тип та метод оплати. 4. Зберегти.
Очікуваний результат	Категорія створюється та відображається у списку.
Статус	Успішно

Таблиця 4.13 – Сценарій тестування «Редагування операції»

ID	11
Мета	Перевірити оновлення операції та перерахунок балансу
Передумови	Існує хоча б одна операція
Кроки виконання	1. Відкрити список операцій. 2. Обрати операцію для редагування. 3. Змінити суму або тип операції. 4. Зберегти зміни.
Очікуваний результат	Дані операції оновлюються, баланс рахунку перераховується відповідно до змін.
Статус	Успішно

Таблиця 4.14 – Сценарій тестування «Видалення операції»

ID	12
Мета	Перевірити видалення операції та оновлення балансу рахунку
Передумови	Існує хоча б одна операція
Кроки виконання	1. Відкрити список операцій. 2. Натиснути видалення. 3. Підтвердити дію.
Очікуваний результат	Операція видаляється, баланс рахунку оновлюється.
Статус	Успішно

Таблиця 4.15 – Сценарій тестування «Валідація при порожніх полях»

ID	13
Мета	Перевірити обробку незаповнених обов’язкових полів
Передумови	Користувач авторизований
Кроки виконання	1. Відкрити форму додавання операції. 2. Залишити обов’язкові поля порожніми. 3. Натиснути збереження.
Очікуваний результат	Система не створює операцію та відображає повідомлення про обов’язкові поля.
Статус	Успішно

Таблиця 4.16 – Сценарій тестування «Імпорт коректного CSV-файлу»

ID	14
Мета	Перевірити імпорт фінансових операцій із CSV-файлу
Передумови	Користувач авторизований, існує рахунок
Кроки виконання	1. Відкрити сторінку імпорту. 2. Обрати CSV-файл банківської виписки. 3. Завантажити файл. 4. Підтвердити імпорт.
Очікуваний результат	Дані з файлу зчитуються, операції створюються, баланс рахунку оновлюється.
Статус	Успішно

Таблиця 4.17 – Сценарій тестування «Вихід із системи»

ID	15
Мета	Перевірити завершення активної сесії користувача
Передумови	Користувач авторизований
Кроки виконання	1. Натиснути кнопку виходу.
Очікуваний результат	Сесія користувача завершується та відкривається сторінка входу.
Статус	Успішно

Таблиця 4.18 – Сценарій тестування «Перевірка пошуку дублікатів під час імпорту»

ID	16
Мета	Перевірити запобігання повторному додаванню однакових операцій
Передумови	Файл уже імпортувався раніше
Кроки виконання	1. Повторно завантажити той самий CSV-файл. 2. Запустити імпорт.
Очікуваний результат	Система визначає дублікати та не додає однакові операції повторно.
Статус	Успішно

Таблиця 4.19 – Сценарій тестування «Перегляд статистики»

ID	17
Мета	Перевірити правильність формування фінансової статистики
Передумови	У системі є операції доходів і витрат
Кроки виконання	1. Відкрити сторінку статистики. 2. Обрати період.
Очікуваний результат	Система відображає коректні суми доходів, витрат і загальний баланс за вибраний період.
Статус	Успішно

Таблиця 4.20 – Сценарій тестування «Розрахунок бюджетного ліміту»

ID	18
Мета	Перевірити обчислення використаного та залишкового бюджету
Передумови	У користувача задано місячний бюджет
Кроки виконання	1. Встановити місячний бюджет. 2. Додати кілька витрат. 3. Перевірити відображення залишку бюджету.
Очікуваний результат	Система коректно розраховує використану суму та залишок бюджету.
Статус	Успішно

Таблиця 4.21 – Сценарій тестування «Конвертація валют»

ID	19
Мета	Перевірити зміну валюти та коректність відображення сум
Передумови	Доступна функція вибору валюти
Кроки виконання	1. Відкрити налаштування. 2. Змінити валюту. 3. Перевірити відображення сум.
Очікуваний результат	Суми відображаються відповідно до вибраної валюти або валютного налаштування.
Статус	Успішно

За результатами проведеного тестування встановлено, що вебзастосунок коректно виконує основні функції, передбачені технічним завданням. Модулі реєстрації, авторизації, управління рахунками, категоріями та операціями працюють стабільно. Імпорт банківських виписок забезпечує правильне зчитування фінансових даних, обробку основних полів і запобігання дублюванню записів.

Висновки до розділу 4

У четвертому розділі було розглянуто програмну реалізацію інтелектуальної системи обліку особистих фінансів на основі фінансового трекера. Описано структуру проєкту, особливості організації серверної та клієнтської частин вебзастосунку, а також реалізацію основних функціональних модулів системи.

У процесі розробки реалізовано механізми реєстрації та автентифікації користувачів, обліку доходів і витрат, управління категоріями фінансових операцій, встановлення бюджетних лімітів, формування статистики та аналітичних звітів. Також забезпечено підтримку мультивалютності та можливість імпорту банківських виписок для автоматизації введення фінансових даних.

Окрему увагу приділено тестуванню розробленої системи. Проведено перевірку коректності роботи основних функціональних можливостей за допомогою тест-кейсів, що дозволило підтвердити працездатність вебзастосунку, правильність обробки даних та відповідність реалізованого функціоналу поставленим вимогам.

Отримані результати свідчать про успішне досягнення поставленої мети. Розроблена система забезпечує ефективний облік особистих фінансів, надає користувачеві інструменти для контролю бюджету та аналізу фінансової активності, а також створює основу для подальшого розширення функціональних можливостей і впровадження додаткових інтелектуальних механізмів аналізу даних.

ВИСНОВКИ

У результаті виконання кваліфікаційної роботи розроблено інтелектуальну систему обліку особистих фінансів у вигляді вебзастосунку фінансового трекера, орієнтованого на потреби українських користувачів.

У ході роботи вирішено такі основні завдання:

1. Проведено аналіз предметної сфери та існуючих програмних рішень для обліку особистих фінансів (YNAB, Saldo Finance, Monobudget, Firefly III, Toshl Finance, Spendee). Виявлено спільні недоліки: відсутність україномовного інтерфейсу, обмежена інтеграція з вітчизняними банками, платний доступ до ключових можливостей.

2. На основі порівняльного аналізу обґрунтовано вибір технологічного стеку: Laravel як серверний PHP-фреймворк з патерном MVC, MySQL як реляційна СУБД, Blade+Vanilla JS як фронтенд-стек, Laragon як середовище розробки.

3. Розроблено архітектуру та структуру системи: спроєктовано реляційну схему бази даних з 4 нормалізованими таблицями, розроблено UML-діаграми варіантів використання та класів.

4. Реалізовано вісім функціональних модулів: реєстрація та автентифікація, облік транзакцій (повний CRUD), управління категоріями та рахунками, бюджетні ліміти з індикаторами виконання, статистика та графіки (Chart.js), імпорт CSV-виписок Monobank і ПриватБанку, мультивалютність (UAH/USD/EUR/PLN) з автоматичним перерахунком за курсом НБУ.

5. Реалізовано алгоритм інтелектуального аналізу витрат, що автоматично групує транзакції за категоріями та зіставляє з бюджетними лімітами.

6. Проведено тестування всіх функціональних модулів системи, результати якого підтвердили коректну роботу застосунку.

Практична цінність розробленого застосунку визначається орієнтацією на потреби вітчизняних користувачів: україномовний інтерфейс, підтримка імпорту виписок популярних українських банків та мультивалютний облік з актуальними

курсами НБУ.

Подальшими напрямками розвитку системи можуть бути: реалізація мобільного застосунку, підключення прямого API Monobank для автоматичної синхронізації транзакцій у реальному часі, впровадження машинного навчання для вдосконалення алгоритму автоматичної категоризації витрат, а також реалізація модуля фінансового прогнозування на основі аналізу попередніх витрат.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Archeet, Khandelwal. "Budget Plan and Expense Tracker Using PHP." (2025).

2. YNAB (You Need A Budget). URL: https://www.ynab.com (date of access: 20.03.2026).

3. Saldo Finance. URL: https://saldofinanceapp.com/ (date of access: 20.03.2026).

4. Monobudget. URL: https://apps.apple.com/ua/app/monobudget/id1491162880 (date of access: 20.03.2026).

5. Firefly III. URL: https://www.firefly-iii.org (date of access: 20.03.2026).

6. Toshl Finance. URL: https://toshl.com (date of access: 20.03.2026).

7. Spendee. URL: https://www.spendee.com (date of access: 20.03.2026).

8. Барабаш В. О. Розробка вебдодатку для оптимізації ведення бюджету із використанням C#: робота на здобуття кваліфікаційного ступеня бакалавра: спец. 121 - інженерія програмного забезпечення / наук. кер. Ю. М. Стоянов. Тернопіль: Тернопільський національний технічний університет імені Івана Пулюя, 2024. 77 с.

9. Оліфіренко О. В. Застосунок на мові Python: кваліфікаційна робота бакалавра: 122 Комп’ютерні науки / О. В. Оліфіренко; наук. керівник В. В. Василенко. Київ, 2025.

10. Бондаренко В. С. Вебзастосунок для управління особистими фінансами: дипломний проєкт бакалавра: 123 Комп’ютерна інженерія / Бондаренко Владислав Сергійович. – Київ, 2025. – 128 с.

11. Харлан А. А. Впровадження компонентно-орієнтованої архітектури для створення додатку управління особистими фінансами з використанням React та Electron: робота на здобуття кваліфікаційного ступеня магістра: спец. 121 - інженерія програмного забезпечення / наук. кер. М. Р. Петрик. Тернопіль: Тернопільський національний технічний університет імені Івана Пулюя, 2024. 74 с.

12. Ситняк О. Р. Розробка дизайну мобільного додатку для оптимізації фінансових потоків: пояснювальна записка до кваліфікаційної роботи здобувача вищої освіти на першому (бакалаврському) рівні, спеціальність 186 Видавництво та поліграфія / О. Р. Ситняк; М-во освіти і науки України, Харків. нац. ун-т радіоелектроніки. – Харків, 2025. – 56 с.

13. Shaw, B., Badhwar, S., Bird, A., Bhushan, S. B., & Guest, C. (2021). Web Development with Django. Packt Publishing.

14. Bojinov, V. (2018). RESTful Web API Design with Node. js 10: Learn to create robust RESTful web services with Node. js, MongoDB, and Express. js. Packt Publishing Ltd.

15. Yadav, N., Rajpoot, D. S., & Dhakad, S. K. (2019, November). LARAVEL: a PHP framework for e-commerce website. In 2019 Fifth International Conference on Image Information Processing (ICIIP) (pp. 503-508). IEEE.

16. Khoerunisa, A., & Supriyati. (2023, October). Designing a budget accounting information system at a website-based village office using PHP and PostgreSQL. In AIP Conference Proceedings (Vol. 2882, No. 1, p. 070008). AIP Publishing LLC.

17. Momjian, B. (2001). PostgreSQL: introduction and concepts (Vol. 192, p. 2001). New York: Addison-Wesley.

18. Banker, K., Garrett, D., Bakkum, P., & Verch, S. (2016). MongoDB in action: covers MongoDB version 3.0. Simon and Schuster.

19. Chandra, A. Y., & Setyaningsih, P. W. (2025). Benchmarking Local Development Environments: Analyzing the Performance of XAMPP, MAMP, and Laragon. Bulletin of Computer Science Research, 5(3), 193-206.

20. Miell, I., & Sayers, A. (2019). Docker in practice. Simon and Schuster.

21. Chen, S., Thaduri, U. R., & Ballamudi, V. K. R. (2019). Front-end development in react: an overview. Engineering International, 7(2), 117-126.

22. Song, J., Zhang, M., & Xie, H. (2019). Design and implementation of a vue. js-based college teaching system. International Journal of Emerging Technologies in Learning (Online), 14(13), 59.

23. Laravel. URL: https://laravel.com/docs (date of access: 24.03.2026).

24. MySQL. URL: https://dev.mysql.com/doc (date of access: 24.03.2026).

25. Laragon. URL: https://laragon.org (date of access: 24.03.2026).

26. phpMyAdmin. URL: https://www.phpmyadmin.net (date of access: 24.03.2026).

27. Blade Templates. URL: https://laravel.com/docs/blade (date of access: 25.03.2026).

28. MDN Web Docs. URL: https://developer.mozilla.org/uk/docs/Web/JavaScript (date of access: 26.03.2026).

29. Carbon – A simple PHP API extension for DateTime. URL: https://carbon.nesbot.com/guide/getting-started/introduction.html (date of access: 27.03.2026).

30. Jain, D. (2024). Ultimate Laravel for Modern Web Development: Build Robust and Interactive Enterprise-Grade Web Apps using Laravel's MVC, Authentication, APIs, and Cloud Deployment (English Edition). Orange Education Pvt Ltd.

31. Da Rocha, H. (2019). Learn Chart. js: Create interactive visualizations for the web with chart. js 2. Packt Publishing Ltd.

32. Benbba, S. (2021). Comparison of D3. js and Chart. js as visualisation tools. Science and Engineering.

ДОДАТОК А
Код контролера керування фінансовими операціями

```
<?php
namespace App\Http\Controllers;
use App\Models\Account;
use App\Models\Category;
use App\Models\Operation;
use App\Support\BudgetLimitService;
use Illuminate\Http\Request;
use Illuminate\Support\Facades\DB;
use Illuminate\Support\Facades\Auth;
use Illuminate\Validation\Rule;
class OperationController extends Controller
{
    public function index(BudgetLimitService $budgetLimitService)
    {
        $operations = Operation::with(['category', 'account'])
            ->where('user_id', Auth::id())
            ->orderByDesc('date')
            ->orderByDesc('id')
            ->get();
        $categories = Category::where('user_id', Auth::id())->orderBy('name')->get();
$accounts      =      Account::where('user_id',      Auth::id())->orderBy('type')-
>orderBy('name')->get();
        $budgetLimit = $budgetLimitService->monthlySummary(Auth::user());
        $budgetWarning = $budgetLimitService->warningMessage(Auth::user());
        return view('operations', compact(
            'operations',
            'categories',
            'accounts',
            'budgetLimit',
            'budgetWarning'
        ));
    }
    public function store(Request $request, BudgetLimitService $budgetLimitService)
    {
        $data = $request->validate([
            'account_id' => 'required',
            'category_id' => [
                'nullable',
                Rule::exists('categories', 'id')->where(fn($query) => $query
                    ->where('user_id', Auth::id())
                    ->where('type', $request->input('type'))
                ),
            ],
            'name' => 'required|string',
            'amount' => 'required|numeric|min:0.01',
            'type' => 'required|in:income,expense',
            'method' => 'required|in:card,cash',
        ]);
$account            =            Account::where('user_id',            Auth::id())-
>findOrFail($data['account_id']);
$category    =    $this->categoryForOperation($data['category_id']    ??    null,
$data['type']);
        DB::transaction(function () use ($data, $account, $category) {
            $operation = Operation::create([
                'user_id' => Auth::id(),
                'account_id' => $account->id,
                'category_id' => $category->id,
                'name' => $data['name'],
```

```
                'amount' => $data['amount'],
                'type' => $data['type'],
                'method' => $data['method'],
                'date' => now()
            ]);

            $this->applyBalanceEffect($operation);
        });
        $warning = $budgetLimitService->warningMessage(Auth::user());
        return $warning
            ? redirect()->back()->with('budget_warning', $warning)
            : redirect()->back();
    }
    private function applyBalanceEffect(Operation $operation): void
    {
$this->moveAccountBalance($operation, $operation->type === 'income' ? 1 : -
1);
        $this->moveCategoryAmount($operation, 1);
    }
    private function reverseBalanceEffect(Operation $operation): void
    {
$this->moveAccountBalance($operation, $operation->type === 'income' ? -1 :
1);
        $this->moveCategoryAmount($operation, -1);
    }
}
```

ДОДАТОК Б
Код контролера імпорту банківських операцій

```
<?php
namespace App\Http\Controllers;
use App\Models\Account;
use App\Models\Category;
use App\Models\Operation;
use App\Support\BudgetLimitService;
use Carbon\Carbon;
use Illuminate\Http\Request;
use Illuminate\Http\UploadedFile;
use Illuminate\Support\Facades\DB;
use Illuminate\Support\Facades\Auth;
use Illuminate\Support\Str;
use ZipArchive;
class ImportedOperationController extends Controller
{
    public function import(Request $request, BudgetLimitService $budgetLimitService)
    {
        $request->validate([
            'file' => 'required|file|max:10240',
            'bank' => 'required|string',
        ]);
        $rows = $this->readRows($request->file('file'));
        if (count($rows) < 1) {
            return response()->json([
                'success' => false,
                'message' => 'Файл порожній або має невідомий формат.',
            ], 422);
        }
        [$headers, $dataRows] = $this->splitHeaderAndRows($rows, $request->bank);
        $count = 0;
        $skipped = 0;
        $duplicates = 0;
        foreach ($dataRows as $row) {
$transaction     =     $this->extractTransaction($row,     $headers,     $request-
>bank);
if      (!$transaction['date']      ||      !$transaction['description']      ||
$transaction['amount'] === null) {
                $skipped++;
                continue;
            }
            $type = $transaction['amount'] < 0 ? 'expense' : 'income';
            $amount = abs($transaction['amount']);
if                          ($this->operationAlreadyImported($transaction['date'],
$transaction['description'], $amount, $type)) {
                $duplicates++;
                continue;
            }
$account               =               $this->accountForTransaction($request->bank,
$transaction['card'] ?? null);
$category     =     $this->detectCategory($transaction['description'],     $type,
$transaction['category'] ?? null);
DB::transaction(function    ()    use    ($account,    $category,    $transaction,
$type, $amount) {
                $operation = Operation::create([
                    'user_id' => Auth::id(),
                    'account_id' => $account->id,
                    'category_id' => $category->id,
                    'name' => $transaction['description'],
```

```
                    'amount' => $amount,
                    'type' => $type,
                    'method' => 'card',
                    'date' => $transaction['date']
                ]);
                $this->applyBalanceEffect($operation);
            });
            $count++;
        }
        return response()->json([
            'success' => true,
            'count' => $count,
            'skipped' => $skipped,
            'duplicates' => $duplicates,
            'budget_warning' => $budgetLimitService->warningMessage(Auth::user()),
        ]);
    }
    private function readRows(UploadedFile $file): array
    {
        $extension = mb_strtolower($file->getClientOriginalExtension());
if  ($extension  ===  'xlsx'  ||  $this->isZipSpreadsheet($file->getRealPath()))
{
            return $this->readXlsxRows($file->getRealPath());
        }
        return $this->readCsvRows($file->getRealPath());
    }

    private function parseAmount(?string $value): ?float
    {
        if ($value === null) return null;
        $value = trim(str_replace(["\u{00A0}", ' '], '', $value));
        if ($value === '' || !preg_match('/\d/', $value)) return null;
        $isNegative = str_contains($value, '-');
        $value = preg_replace('/[^\d,.\-]/u', '', $value);
        $value = str_replace(',', '.', $value);
        $amount = (float) $value;
        return $isNegative ? -abs($amount) : $amount;
    }
    private function parseDate(?string $value): ?string
    {
        if (!$value) return null;
        $formats = [
            'd.m.Y H:i:s',
            'd.m.Y H:i',
            'd.m.Y',
            'Y-m-d H:i:s',
            'Y-m-d H:i',
            'Y-m-d',
        ];
        foreach ($formats as $format) {
            try {
return          Carbon::createFromFormat($format,          $value)->format('Y-m-d
H:i:s');
            } catch (\Throwable) {
                continue;
            }
        }
        return null;
    }
}
```

ДОДАТОК В
Код контролера формування фінансової статистики

```
<?php
namespace App\Http\Controllers;
use App\Models\Account;
use App\Models\Operation;
use App\Support\BudgetLimitService;
use Carbon\Carbon;
use Illuminate\Http\Request;
use Illuminate\View\View;
class StatsController extends Controller
{
public function index(Request $request, BudgetLimitService $budgetLimitService): View
    {
        $operations = Operation::with('category')
            ->where('user_id', auth()->id())
            ->orderBy('date')
            ->get();
        $balance = (float) Account::where('user_id', auth()->id())->sum('balance');
        [$customStart, $customEnd] = $this->customPeriod($request);
        $statsData = [
'week'     =>     $this->periodData($operations,     $balance,     Carbon::now()->startOfWeek(), Carbon::now()),
'month'     =>     $this->periodData($operations,     $balance,     Carbon::now()->startOfMonth(), Carbon::now()),
'year'     =>     $this->periodData($operations,     $balance,     Carbon::now()->startOfYear(), Carbon::now()),
        ];
        if ($customStart && $customEnd) {
$statsData['custom']       =       $this->periodData($operations,       $balance, $customStart, $customEnd);
        }
        return view('stats', [
            'statsData' => $statsData,
            'selectedPeriod' => isset($statsData['custom']) ? 'custom' : 'month',
            'dateFrom' => $customStart?->toDateString(),
            'dateTo' => $customEnd?->toDateString(),
            'budgetLimit' => $budgetLimitService->monthlySummary(auth()->user()),
            'budgetWarning' => $budgetLimitService->warningMessage(auth()->user()),
        ]);
    }
private function periodData($operations, float $balance, Carbon $start, Carbon $end): array
    {
        $periodOperations = $operations->filter(
fn($operation)     =>     Carbon::parse($operation->date)->betweenIncluded($start, $end)
        );
        $income = (float) $periodOperations->where('type', 'income')->sum('amount');
$expense      =      (float)      $periodOperations->where('type',      'expense')->sum('amount');
        $net = $income - $expense;
        $days = max(1, (int) $start->diffInDays($end) + 1);
        $expenseGroups = $periodOperations->where('type', 'expense')
            ->groupBy(fn($operation) => $operation->category->name ?? 'Без категорії')
            ->map(fn($items) => round((float) $items->sum('amount'), 2))
            ->sortDesc();
        $largestOperation = $periodOperations
            ->where('type', 'expense')
```

```
            ->sortByDesc('amount')
            ->first();
        return [
            'balance' => round($balance, 2),
            'income' => round($income, 2),
            'expense' => round($expense, 2),
            'avgDailyExpense' => round($expense / $days, 2),
            'savingsRate' => $income > 0 ? round(($net / $income) * 100, 1) : 0,
            'topCategory' => $expenseGroups->keys()->first() ?? 'Немає даних',
            'topCategoryValue' => $expenseGroups->first() ?? 0,
            'largestOperation' => $largestOperation?->name ?? 'Немає даних',
'largestOperationValue'    =>    round((float)    ($largestOperation?->amount    ??
0), 2),
            'categoryLabels' => $expenseGroups->keys()->take(7)->values(),
            'categoryValues' => $expenseGroups->take(7)->values(),
            'flowLabels' => ['Доходи', 'Витрати'],
            'flowValues' => [round($income, 2), round($expense, 2)],
            'trendLabels' => $this->trendLabels($start, $end),
            'trendValues' => $this->trendValues($periodOperations, $start, $end),
        ];
    }
}
```