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

ВЕБСЕРВІС ДЛЯ КЕРУВАННЯ ОСОБИСТИМИ ФІНАНСАМИ

Спеціальність 122 Комп'ютерні науки

Освітня програма «Комп'ютерні науки»

Здобувач _____ Нікіта ЯЦЮК

«___»_____ 2026 р.

Керівник д-р пед. наук, професор _____ Олександр МЄЩАНІНОВ

«___»_____ 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

Яцюка Нікіти Андрійовича
(прізвище, ім’я, по батькові здобувача)

1. Тема кваліфікаційної роботи: «Вебсервіс для керування особистими фінансами».

Керівник роботи: Мєщанінов Олександр Павлович, професор кафедри ІІС, д-р пед. наук , професор.

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: повністю функціональний вебсервіс для керування особистими фінансами, що забезпечує облік транзакцій, категоризацію, бюджетування, інтерактивну аналітику та формування звітів у форматах PDF та Excel.

4. Перелік питань, що підлягають розробці: аналіз предметної сфери та огляд існуючих аналогів програм; обґрунтування вибору технологічного стеку; проєктування архітектури системи та структури бази даних; розробка модулів

реєстрації, авторизації та захисту даних користувачів; реалізація журналу фінансових транзакцій із фільтрацією, пошуком та сортуванням; розробка модулю статистики з інтерактивними графіками та аналітичними рекомендаціями; реалізація системи бюджетування з візуальними індикаторами виконання; розробка модулів експорту даних у формати PDF та Excel; тестування функціональності, безпеки та продуктивності системи.

5. Перелік графічних матеріалів: діаграма варіантів використання системи (Use Case Diagram); блок-схема алгоритму обробки HTTP-запиту; схема навігації між екранами системи.

Керівник роботи _____ _____
(Особистий підпис) *(Олександр МЄЩАНІНОВ)*

Здобувач _____ _____
(Особистий підпис) *(Нікіта ЯЦЮК)*

Дата видачі завдання «22» грудня 2025 р.

КАЛЕНДАРНИЙ ПЛАН
кваліфікаційної роботи

Тема: Вебсервіс для керування особистими фінансами

№	Найменування роботи	Початок	Закінчення	Примітки
1	Отримання завдання на виконання КР	21.12.2025	24.12.2025	Виконано
2	Аналіз предметної області та постановка задачі	25.12.2025	30.01.2026	Виконано
3	Огляд літературних джерел за темою кваліфікаційної роботи, зокрема огляд публікацій, щодо процесу керування фінансами, та огляд аналогічних систем (Monobank, Monefy, Spendee)	31.01.2026	15.02.2026	Виконано
4	Обґрунтування вибору технологічного стеку: Python, Django, SQLite, ReportLab, OpenPyXL, Chart.js	16.02.2026	01.03.2026	Виконано
5	Проєктування архітектури системи (патерн MTV), розробка моделей БД, реалізація функціоналу системи	02.03.2026	20.05.2026	Виконано
6	Функціональне тестування системи, тестування безпеки та інтерфейсу	21.05.2026	24.05.2026	Виконано
7	Перший попередній захист КР на засіданні комісії кафедри	25.05.2026	25.05.2026	Виконано
8	Корегування роботи за результатами попереднього захисту	26.05.2026	04.06.2026	Виконано
9	Другий попередній захист КР на засіданні комісії кафедри	05.06.2026	05.06.2026	Виконано
10	Доробка та остаточне оформлення КР	06.06.2026	14.06.2026	Виконано
11	Подання КР, її електронної копії та інших документів (відгуку, рецензії) до захисту	15.06.2026	19.06.2026	Виконано

Керівник роботи _____ _____
(Особистий підпис) *(Олександр МЄЩАНІНОВ)*

Здобувач _____ _____
(Особистий підпис) *(Нікіта ЯЦЮК)*

Дата складання календарного плану
«29» січня 2026 р.

АНОТАЦІЯ

до кваліфікаційної роботи
здобувача групи 401 ЧНУ ім. Петра Могили

Яцюка Нікіта Андрійовича

на тему: **“ВЕБСЕРВІС ДЛЯ КЕРУВАННЯ ОСОБИСТИМИ ФІНАНСАМИ”**

Актуальність даної роботи полягає у необхідності розробки доступного, функціонального та безкоштовного інструменту для автоматизованого обліку й аналізу особистих фінансів. Більше 60 % населення не ведуть систематичного обліку власних доходів і витрат, а більшість існуючих рішень є або платними, або прив'язаними до конкретних банків, або не мають підтримки українського інтерфейсу. Це зумовлює потребу у вебсервісі, орієнтованому на широку аудиторію без технічних знань.

Об'єктом роботи є процеси автоматизованого обліку, аналізу та планування особистих фінансів.

Предметом роботи є методи та засоби розробки вебзастосунків для керування особистими фінансами з використанням мови програмування Python та фреймворку Django.

Метою роботи є розробка функціонального, зручного і масштабованого веб-сервісу для керування особистими фінансами, що забезпечує комплексний облік фінансових операцій, аналітику, бюджетування та формування звітів у форматах PDF та Excel.

В результаті виконання роботи розроблено вебсервіс для керування особистими фінансами, реалізовано модулі реєстрації та авторизації користувачів, ведення журналу фінансових транзакцій із фільтрацією, сортуванням та пошуком, адаптивну систему категоризації витрат і доходів, модуль бюджетування з візуальними індикаторами виконання, модуль статистики з чотирма типами інтерактивних графіків (Chart.js) та автоматичними аналітичними рекомендаціями, а також функціонал експорту даних у формати Excel (OpenPyXL) та PDF (ReportLab). Проведено функціональне тестування (15 сценаріїв), тестування

безпеки (6 перевірок) та тестування інтерфейсу у п'яти браузерах. Усі 15 функціональних та 6 нефункціональних вимог виконано в повному обсязі.

Дана робота складається з чотирьох розділів. У першому розділі проведено аналіз предметної сфери управління особистими фінансами, огляд трьох програмних аналогів (Monobank, Monefy, Spendee) та сформульовано постановку задачі. Другий розділ присвячено методам, підходам та технологіям розробки системи: обґрунтовано вибір архітектурного патерну MTV, технологічного стеку та засобів захисту інформації. У третьому розділі описано програмну реалізацію всіх компонентів вебсервісу: архітектуру, моделі бази даних, бізнес-логіку, шаблони інтерфейсу та модулі експорту. У четвертому розділі наведено керівництво користувача та результати комплексного тестування системи.

Загальний обсяг роботи – 112 сторінок. Кваліфікаційна робота містить 2 додатки, 21 рисунок, 30 таблиць і 32 джерела посилання.

Ключові слова: Python, Django, ORM, вебсервіс, фінансові операції, статистика, аналітика, графіки, база даних, веброзробка.

ABSTRACT

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

Yatsyuk Nikita

“WEB SERVICE FOR PERSONAL FINANCE MANAGEMENT”

A relevance of this work lies in the need to develop an accessible, functional, and free tool for automated accounting and analysis of personal finances. More than 60% of the population do not systematically track their income and expenses, while most existing solutions are either paid, tied to specific banks, or lack Ukrainian language support. This determines the need for a web service oriented towards a wide audience without technical expertise.

An object of the work is the processes of automated accounting, analysis, and planning of personal finances.

A subject of the work is the methods and tools for developing web applications for personal finance management using the Python programming language and the Django framework.

A purpose of the work is to develop a functional, user-friendly, and scalable web service for personal finance management that provides comprehensive transaction accounting, analytics, budgeting, and report generation in PDF and Excel formats.

As a result of the work, the "FinancePro" web service was developed. The following modules were implemented: user registration and authorization, a financial transaction log with filtering, sorting, and search, an adaptive income and expense categorization system, a budgeting module with visual progress indicators, a statistics module with four types of interactive charts (Chart.js) and automatic analytical recommendations, and data export functionality to Excel (OpenPyXL) and PDF (ReportLab) formats. Functional testing (15 scenarios), security testing (6 checks), and interface testing across five browsers were conducted. All 15 functional and 6 non-functional requirements were fulfilled in full.

This work consists of four sections. The first section analyzes the subject area of personal finance management, reviews three software analogues (Monobank, Monefy, Spendee), and formulates the problem statement. The second section covers the methods, approaches, and technologies used: the MTV architectural pattern, technology stack, and information security measures are justified. The third section describes the software implementation of all system components: architecture, database models, business logic, interface templates, and export modules. The fourth section presents the user guide and the results of comprehensive system testing.

The overall scope of the work is 112 pages. Thesis contains 2 application, 21 figures, 30 tables and 32 references in it.

Key words: Python, Django, ORM, web service, financial transactions, statistics, analytics, charts, database, web development.

ЗМІСТ

СКОРОЧЕННЯ ТА УМОВНІ ПОЗНАКИ

БД – База даних

ЕК – Екзаменаційна комісія

КР – Кваліфікаційна робота

ОС – Операційна система

ПЗ – Програмне забезпечення

СУБД – Система управління базами даних

ФМ – Фінансовий менеджмент

API (Application Programming Interface) – Інтерфейс програмування застосунків

CSRF (Cross-Site Request Forgery) – Підробка міжсайтових запитів

CSS (Cascading Style Sheets) – Каскадні таблиці стилів

DSTU – Державний стандарт України

HTML (HyperText Markup Language) – Мова розмітки гіпертексту

HTTP (HyperText Transfer Protocol) – Протокол передачі гіпертексту

IDE (Integrated Development Environment) – Інтегроване середовище розробки

JSON (JavaScript Object Notation) – Формат обміну даними

MTV (Model-Template-View) – Архітектурний шаблон Django

MVC (Model-View-Controller) – Архітектурний шаблон «Модель-Вигляд-Контролер»

ORM (Object-Relational Mapping) – Об'єктно-реляційне відображення

PDF (Portable Document Format) – Формат переносного документа

SQL (Structured Query Language) – Мова структурованих запитів

UI (User Interface) – Інтерфейс користувача

URL (Uniform Resource Locator) – Уніфікований локатор ресурсу

UX (User Experience) – Досвід користувача

UML (Unified Modeling Language) – Уніфікована мова моделюва

ВСТУП

Управління особистими фінансами є однією з ключових компетентностей сучасної людини. За даними досліджень Національного банку України та міжнародних організацій, понад 60 % дорослого населення не ведуть систематичного обліку власних доходів і витрат, що призводить до нераціонального використання коштів, неможливості формування заощаджень та досягнення фінансових цілей [1].

Стрімкий розвиток інформаційних технологій відкриває широкі можливості для автоматизації процесів обліку та аналізу фінансів. Сучасні вебсервіси здатні в реальному часі відображати стан бюджету, будувати аналітичні графіки, формувати звіти та надавати персоналізовані рекомендації. Проте більшість існуючих рішень мають суттєві обмеження: прив'язаність до конкретних банків, закрита платна модель, відсутність підтримки реалій вітчизняного ринку або надмірна складність інтерфейсу.

У даній кваліфікаційній роботі розробляється вебсервіс для керування особистими фінансами на основі мови програмування Python та фреймворку Django. Система реалізує повний цикл роботи з фінансовими даними: від введення транзакцій та їх категоризації до побудови аналітичних графіків, бюджетування та формування звітів у форматах PDF та Excel. На відміну від більшості існуючих аналогів, розроблюваний сервіс є повністю безкоштовним, не прив'язаним до конкретного банку та орієнтованим на україномовних користувачів.

Для досягнення поставленої мети необхідно вирішити такі задачі:

– провести аналіз предметної сфери та огляд існуючих рішень для управління особистими фінансами;

– визначити та обґрунтувати вибір технологічного стеку для розробки вебсервісу;

– спроєктувати архітектуру системи та структуру бази даних;

– реалізувати модулі реєстрації, авторизації та управління профілем користувача;

– розробити функціонал ведення журналу фінансових операцій з підтримкою категоризації, фільтрації та пошуку;

– реалізувати модуль побудови інтерактивних графіків та статистичних звітів;

– розробити функціонал експорту даних у формати PDF та Excel;

– реалізувати систему бюджетування та автоматичних аналітичних рекомендацій;

– провести функціональне тестування та тестування безпеки системи.

Для вирішення поставлених задач використовувались: системний аналіз предметної галузі; об'єктно-орієнтоване проєктування та програмування; архітектурний патерн MTV (Model-Template-View); методи тестування програмних систем. Засобами реалізації слугували: Python 3.10+, Django 5.0, SQLite, ReportLab, OpenPyXL, Chart.js, VS Code.

Практичне значення роботи полягає у тому, що розроблений веб-сервіс може бути безпосередньо використаний для управління особистими або сімейними фінансами, а також слугує базою для подальшого розвитку в напрямку мобільного застосунку, інтеграції з банківськими API або хмарного розгортання.

1 АНАЛІЗ ПРЕДМЕТНОЇ СФЕРИ. ПОСТАНОВКА ЗАДАЧІ

1.1 Опис предметної сфери та актуальність задачі

Управління особистими фінансами (або персональний фінансовий менеджмент) – це систематична діяльність фізичної особи щодо планування, обліку, аналізу та контролю власних грошових потоків з метою досягнення фінансових цілей та забезпечення фінансової стабільності [2, 3].

Основні процеси особистого фінансового менеджменту можна структурувати у чотири взаємопов'язані блоки:

– облік – систематична фіксація всіх фінансових операцій (доходів та витрат) із зазначенням суми, дати, категорії та опису;

– аналіз – вивчення структури та динаміки фінансових потоків, виявлення закономірностей а проблемних зон;

– планування (бюджетування) – встановлення лімітів витрат за категоріями на основі аналізу та фінансових цілей;

– контроль – відстеження відповідності фактичних показників плановим та коригуючі дії.

Ключові поняття предметної сфери, що використовуються у даній роботі, наведено у табл. 1.1.

Таблиця 1.1 – Основні терміни та визначення предметної сфери

Термін	Визначення
Фінансова транзакція	Будь-яка операція, що змінює стан фінансів користувача: отримання доходу або здійснення витрати
Дохід	Надходження грошових коштів (заробітна плата, фріланс, дивіденди тощо)
Витрата	Відтік грошових коштів на придбання товарів, послуг, оплату зобов'язань

Кінець таблиці 1.1

Термін	Визначення
Категорія	Класифікаційна одиниця для групування транзакцій за їх економічним змістом
Бюджет	Плановий ліміт витрат за категорією на визначений обліковий період
Баланс	Різниця між сукупними доходами та витратами за визначений проміжок часу
Звітність	Структуровані дані про фінансовий стан, подані у стандартному форматі

Актуальність автоматизації процесів особистого фінансового менеджменту обумовлена низкою чинників.

По-перше, за даними глобального індексу фінансової грамотності S&P (Standard & Poor's Financial Services), лише 54 % дорослого населення Східної Європи можуть бути визнані фінансово грамотними [4]. В Україні цей показник є ще нижчим. Відсутність навичок фінансового планування проявляється у неспроможності громадян формувати заощадження та реагувати на непередбачені витрати.

По-друге, ринок програмного забезпечення для особистих фінансів розвивається стрімко. За прогнозами аналітиків, до 2028 року глобальний ринок програм для персонального фінансового менеджменту досягне 1,57 млрд. доларів США при середньорічному темпі зростання понад 5,7 % [5]. Це свідчить про зростаючий попит на відповідні рішення.

По-третє, технологічний прогрес суттєво знизив поріг входу для розробки якісних фінансових вебзастосунків. Сучасний стек Python/Django надає розробникам потужні інструменти для швидкого створення безпечних, масштабованих і функціональних рішень. Саме тому тема даної кваліфікаційної роботи є актуальною та практично значущою.

Україна суттєво відстає від розвинених регіонів як за рівнем фінансової грамотності, так і за охопленням населення цифровими інструментами ФМ (табл. 1.2). Це підтверджує необхідність розробки доступних україномовних рішень для персонального фінансового менеджменту.

Таблиця 1.2 – Статистика фінансової грамотності населення за регіонами

Регіон	**Фінансово грамотне населення, %**	**Охоплення ФМ-застосунками, %**
Північна Америка	57	48
Західна Європа	65	39
Східна Європа	54	22
Азія (розвинені країни)	59	61
Україна (оцінка)	40	15

1.2 Огляд та аналіз наявних аналогів і публікацій

Перед проєктуванням власного рішення було проведено аналіз наявних програмних аналогів та наукових публікацій у сфері персональних фінансових менеджерів.

1.2.1 Аналіз наукових публікацій

Аналіз наукових праць у сфері розробки вебзастосунків для управління особистими фінансами дозволяє виявити основні підходи, методи та технології, що застосовуються у сучасних дослідженнях, а також визначити напрями, які потребують подальшого вивчення.

У роботі [6] досліджуються архітектурні рішення для побудови масштабованих вебзастосунків. Автори обґрунтовують перевагу багаторівневої архітектури над монолітною: розподіл системи на рівні представлення, бізнес-логіки та даних суттєво спрощує супровід коду та дозволяє замінювати окремі

компоненти без переробки всієї системи. Саме цей підхід реалізовано у даній роботі через архітектурний патерн MTV фреймворку Django.

Fowler у фундаментальній праці [7] систематизує патерни проєктування корпоративних застосунків, зокрема детально розглядає патерни роботи з джерелами даних: Active Record, Data Mapper та Repository. Django ORM реалізує підхід Active Record, де кожен клас моделі відповідає таблиці БД і містить методи для взаємодії з нею. Цей підхід є оптимальним для проєктів середнього масштабу, де важлива простота коду над максимальною гнучкістю.

У роботі [7] також наголошується на важливості розмежування між логікою домену та логікою збереження даних. У даному проєкті це розмежування реалізовано таким чином: бізнес-логіка (розрахунок балансу, генерація рекомендацій) зосереджена у View-функціях та методах моделей, тоді як взаємодія з БД повністю делегована Django ORM.

Percival та Gregory у роботі [6] розглядають підходи до тестування веб-застосунків та наголошують на важливості поділу тестів на юніт-тести, інтеграційні та end-to-end тести. Автори стверджують, що навіть для невеликих проєктів наявність автоматизованих тестів критично важлива для виявлення регресій при розширенні функціоналу. У даній роботі тестування проводилось на функціональному рівні та рівні безпеки, що відповідає рекомендаціям авторів для проєктів початкового масштабу.

Аналіз публікацій у сфері безпеки веб-застосунків [8] свідчить, що найбільш розповсюдженими вразливостями залишаються ін'єкції (SQL, XSS) та підробка міжсайтових запитів (CSRF). Організація OWASP щорічно публікує перелік десяти найкритичніших вразливостей, і більшість із них можна усунути засобами самого фреймворку. Зокрема, Django за замовчуванням надає захист від SQL-ін'єкцій через параметризацію ORM-запитів, від XSS − через автоматичне екранування в шаблонізаторі, від CSRF − через вбудований middleware з токенами. Це підтверджує обґрунтованість вибору Django як основи для розробки захищеного фінансового застосунку.

В роботах українських дослідників [2, 3] аналізується стан фінансової грамотності населення та потреби у цифрових інструментах фінансового менеджменту. Зокрема, у звіті НБУ [1] зазначається, що лише 40 % українців мають базові навички фінансового планування, а понад половина респондентів висловили зацікавленість у використанні цифрових інструментів для обліку витрат за умови їх доступності та україномовного інтерфейсу. Ці дані безпосередньо обґрунтовують доцільність розробки вітчизняного вебсервісу для управління особистими фінансами.

Аналіз наукових публікацій у галузі візуалізації фінансових даних свідчить про зростання ролі інтерактивних графічних засобів у прийнятті фінансових рішень. Дослідники встановили, що наочне відображення динаміки витрат у вигляді діаграм підвищує усвідомленість користувачів щодо власних фінансових патернів ефективніше, ніж табличне подання тих самих даних. Це обґрунтовує реалізацію у даній роботі чотирьох типів інтерактивних графіків на основі Chart.js замість статичних таблиць.

Дослідження у сфері UX-проєктування фінансових застосунків вказують на те, що ключовими чинниками, які впливають на регулярність використання фінансового менеджера, є мінімальна кількість дій для введення транзакції та негайний зворотний зв'язок у вигляді оновленої статистики. Ці висновки були враховані при проєктуванні інтерфейсу розроблюваного вебсервісу: форма введення транзакції містить лише п'ять полів, а після збереження користувач автоматично повертається до оновленого журналу.

Таким чином, проведений аналіз наукових публікацій підтверджує актуальність обраної теми та дозволяє визначити теоретичну базу дослідження. Встановлено, що для розробки якісного фінансового вебзастосунку необхідно застосовувати: багаторівневу архітектуру з чітким розподілом відповідальності (патерн MTV); ORM-підхід для абстрагування від конкретної СУБД; вбудовані механізми безпеки фреймворку для захисту від актуальних вразливостей OWASP;

інтерактивну візуалізацію даних для підвищення практичної цінності аналітики; мінімалістичний UX з миттєвим зворотним зв'язком.

1.2.2 Огляд програмних аналогів

Для аналізу обрано три типові представники категорії програм персонального фінансового менеджменту: Monobank, Monefy та Spendee. Огляд здійснено за єдиним набором критеріїв, що дозволяє провести об'єктивне порівняння (див. табл. 1.3).

Таблиця 1.3 – Порівняльний аналіз аналогів та розроблюваної системи

Критерій	**Monobank**	**Monefy**	**Spendee**	**Розроблювана система**
Вебверсія	Ні	Ні	Так (платно)	Так (безкоштовно)
Облік готівки	Ні	Так	Так	Так
Кастомні категорії	Обмежено	Так	Так	Так + кольори/іконки
Бюджетування	Ні	Платно	Платно	Так (безкоштовно)
Графіки та аналітика	Базові	Базові	Розширені (платно)	Розширені (безкоштовно)
Експорт PDF	Ні	Ні	Платно	Так
Експорт Excel	Ні	Ні	Платно	Так
Аналітичні рекомендації	Ні	Ні	Платно	Так (вбудовані)
Підтримка UA мови	Так	Ні	Ні	Так
Безкоштовний повний доступ	Залежить від банку	Ні	Ні	Так

Порівняльний аналіз показав, що кожен із розглянутих аналогів має суттєві обмеження, які знижують їх практичну цінність для широкої аудиторії. Нижче наведено детальний огляд кожного.

1.2.3 Monobank

Monobank – найпопулярніший мобільний банк України з вбудованим фінансовим менеджером. Застосунок автоматично категоризує безготівкові транзакції на основі MCC-кодів торгових точок та відображає базову статистику витрат.

Переваги: автоматична категоризація; інтеграція з банківським рахунком; зручна статистика по картках. Недоліки: відсутність обліку готівкових операцій; неможливість ведення бюджетів; прив'язка виключно до рахунків Monobank; відсутність можливості формування детальних звітів [9].

Інтерфейс застосунку «Monobank» зображено на рис. 1.1.

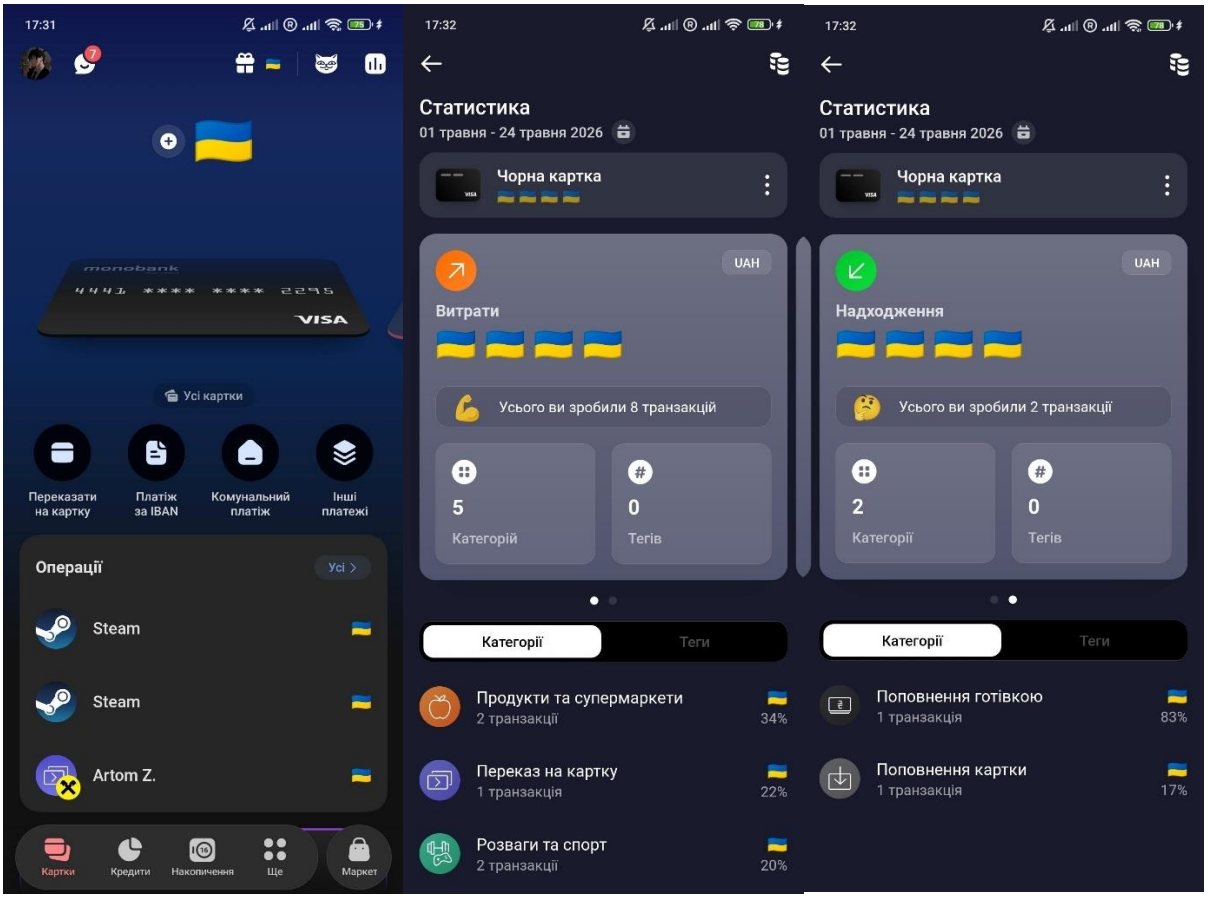


Рисунок 1.1 – Інтерфейс Monobank

1.2.4 Monefy

Monefy – популярний мобільний застосунок для відстеження витрат, доступний на iOS та Android. Орієнтований на максимально просте введення даних через велику кнопку «+».

Переваги: інтуїтивний інтерфейс; швидке введення транзакцій; наочна кругова діаграма. Недоліки: відсутність вебверсії; більшість ключових функцій (синхронізація, місячна статистика) доступні лише у платній версії (від $1,99/міс.); відсутність підтримки української мови [10].

Інтерфейс застосунку «Monefy» зображено на рис. 1.2.

Рисунок 1.2 – Інтерфейс Monefy [11]

1.2.5 Spendee

Spendee – кросплатформений сервіс з вебверсією та мобільними застосунками. Позиціонується як комплексне рішення для управління фінансами.

Переваги: наявність вебверсії; інтеграція з банками; розширена аналітика. Недоліки: складний інтерфейс для початківців; висока вартість преміум-плану (від

$14,99/рік за базовий до $74,99/рік за сімейний); відсутність підтримки українських банків та UI українською мовою [12].

На підставі проведеного огляду можна зробити висновок, що жоден із розглянутих аналогів не надає повного набору функцій у безкоштовному режимі: не поєднує вебінтерфейс, облік готівки, кастомні категорії, бюджетування, розширену аналітику та формування звітів без прив'язки до конкретного банку. Саме ці характеристики становлять конкурентні переваги розроблюваного веб-сервісу.

Інтерфейс сервісу «Spendee» зображено на рис. 1.3.

Рисунок 1.3 – Інтерфейс Spendee [13]

1.3 Постановка задачі

На основі проведеного аналізу предметної сфери та огляду існуючих аналогів сформульовано постановку задачі кфаліфікаційної роботи, а також функціональні та нефункціональні вимоги до розроблюваної системи.

Актуальність роботи зумовлена низьким рівнем охоплення населення України цифровими інструментами фінансового обліку (15 %) та відсутністю на

ринку безкоштовного україномовного вебсервісу, що поєднує облік транзакцій, бюджетування, аналітику та формування звітів без прив'язки до конкретного банку.

Об'єктом роботи є процеси автоматизованого обліку, аналізу та планування особистих фінансів.

Предметом роботи є методи та засоби розробки вебзастосунків для керування особистими фінансами з використанням мови програмування Python та фреймворку Django.

Метою роботи є розробка функціонального, зручного і масштабованого вебсервісу для керування особистими фінансами, що забезпечить комплексний облік фінансових операцій, аналітику, бюджетування та формування звітів у форматах PDF та Excel.

1.3.1 Функціональні вимоги

Функціональні вимоги визначають поведінку системи та описують дії, які вона повинна виконувати. Повний перелік функціональних вимог наведено у табл. 1.4. Перелік нефункціональних вимог наведено у табл. 1.5.

Таблиця 1.4 – Функціональні вимоги до вебсервісу

ID	Вимога	Пріоритет
ФВ-01	Реєстрація нового користувача з валідацією всіх полів форми	Високий
ФВ-02	Авторизація користувача та захист від несанкціонованого доступу	Високий
ФВ-03	Додавання фінансових транзакцій (тип, сума, дата, категорія, опис)	Високий
ФВ-04	Редагування та видалення транзакцій	Високий
ФВ-05	Фільтрація транзакцій за типом, категорією, діапазоном дат	Високий
ФВ-06	Текстовий пошук транзакцій за описом	Середній

Кінець таблиці 1.4

ID	Вимога	Пріоритет
ФВ-07	Сортування транзакцій за датою та сумою	Середній
ФВ-08	Управління кастомними категоріями (CRUD)	Високий
ФВ-09	Відображення інтерактивних графіків доходів/витрат	Високий
ФВ-10	Формування статистики за місяць/рік	Середній
ФВ-11	Генерація аналітичних рекомендацій	Середній
ФВ-12	Встановлення щомісячних бюджетів за категоріями	Середній
ФВ-13	Відображення прогресу виконання бюджету	Середній
ФВ-14	Експорт даних у формат Excel з форматуванням	Низький
ФВ-15	Формування звіту у форматі PDF	Низький

1.3.2 Нефункціональні вимоги

Таблиця 1.5 – Нефункціональні вимоги до вебсервісу

Категорія	Вимога
Продуктивність	Час відгуку системи на стандартні запити не більше 2 секунд при локальному розгортанні
Безпека	CSRF-захист для всіх форм; хешування паролів; ізоляція даних між користувачами
Масштабованість	Модульна архітектура, що дозволяє додавати нові модулі без зміни існуючого коду
Надійність	Валідація всіх вхідних даних на стороні сервера; коректна обробка граничних значень
Зручність	Адаптивний інтерфейс для desktop та мобільних пристроїв; інтуїтивна навігація
Переносимість	Незалежність від ОС; можливість переходу з SQLite на PostgreSQL без зміни коду

Висновки до розділу 1

У першому розділі проведено аналіз предметної сфери управління особистими фінансами. Встановлено, що більше 60 % населення не ведуть систематичного обліку фінансів, а рівень охоплення цифровими ФМ-інструментами в Україні становить лише 15 %. Огляд трьох аналогів (Monobank, Monefy, Spendee) виявив їх ключові обмеження: прив'язку до банків, платний повний функціонал та відсутність україномовного веб-інтерфейсу. Сформульовано 15 функціональних та 6 нефункціональних вимог до розроблюваного вебсервісу. Це підтвердило доцільність та актуальність розробки власного рішення.

2 МОДЕЛІ, МЕТОДИ ТА ІНФОРМАЦІЙНІ ТЕХНОЛОГІЇ

2.1 Методи та підходи для вирішення поставленої задачі

2.1.1 Архітектурний патерн MTV

Для організації коду вебзастосунку обрано архітектурний патерн MTV (Model-Template-View), що є реалізацією класичного MVC (Model-View-Controller) у фреймворку Django. Патерн забезпечує розподіл відповідальності між компонентами системи, що спрощує підтримку та розширення коду.

Загальна ідея будь-якого MVC-подібного патерну полягає у тому, що кожен компонент повинен мати чітко визначену зону відповідальності й не «знати» про деталі реалізації інших компонентів. Це дозволяє змінювати, наприклад, спосіб відображення даних, не торкаючись бізнес-логіки, і навпаки.

Порівняння архітектурних патернів MVC та MTV наведено у табл. 2.1.

Таблиця 2.1 – Порівняння архітектурних патернів MVC та MTV

Компонент MVC	**Відповідний компонент MTV**	**Відповідальність у проєкті**
Model	Model	Опис структури даних; взаємодія з БД через ORM; методи бізнес-логіки
View	Template	HTML-шаблони з динамічним вмістом; успадкування від base.html
Controller	View	Обробка HTTP-запитів; взаємодія між Model та Template
–	URL Dispatcher	Маршрутизація URL-адрес до відповідних View-функцій
–	Forms	Серверна валідація та обробка вхідних даних від користувача
–	Middleware	Наскрізна обробка запитів: сесії, CSRF, аутентифікація

Потік обробки запиту у системі згідно з патерном MTV відбувається наступним чином: браузер надсилає HTTP-запит → URL-диспетчер визначає відповідну View-функцію → View звертається до Model → Model виконує ORM-запит до SQLite → дані передаються у Template → Template-рушій генерує HTML → браузер отримує HTTP-відповідь.

Перевагою MTV перед «чистим» MVC є чіткіше розмежування між логікою отримання даних (View) та логікою їх відображення (Template). Це дозволяє фронтенд-розробнику редагувати шаблони, не торкаючись Python-коду, і навпаки.

2.1.2 Принципи об'єктно-орієнтованого проєктування (SOLID)

При розробці системи застосовано принципи SOLID – п'ять основних принципів об'єктно-орієнтованого проєктування, що забезпечують гнучкість, розширюваність і простоту тестування коду. Застосування принципів SOLID у проєкті наведено у табл. 2.2.

Таблиця 2.2 – Застосування принципів SOLID у проєкті

Принцип	Назва	Реалізація у проєкті
S	Single Responsibility	Кожна View-функція обробляє один тип запитів; кожна модель описує одну сутність; кожна форма – один варіант вводу
O	Open/Closed	Нові категорії та типи звітів додаються розширенням моделей без зміни існуючого коду
L	Liskov Substitution	Стандартна модель User Django замінюється кастомною без порушення логіки будь-якого View
I	Interface Segregation	Форми розділені за призначенням: TransactionForm, CategoryForm, BudgetForm – кожна містить лише свої поля
D	Dependency Inversion	View-функції залежать від абстракцій ORM, а не від конкретного SQL-діалекту СУБД

2.1.3 Патерн проєктування “Репозиторій” та Django ORM

Замість реалізації власного шару доступу до даних у проєкті використовується вбудований ORM Django, який реалізує патерн «Active Record» у поєднанні з елементами патерну «Repository». Це дозволяє будувати запити у вигляді ланцюжків методів Python, не звертаючись до SQL безпосередньо.

ORM (Object-Relational Mapping) – це техніка перетворення даних між несумісними системами типів: об'єктно-орієнтованою мовою програмування (Python) та реляційною СУБД (SQLite). ORM абстрагує взаємодію з БД, забезпечуючи переносимість: код застосунку залишається незмінним при переході між різними СУБД.

Основні можливості Django ORM, що використовуються у проєкті:

– filter() – фільтрація записів за умовами (еквівалент WHERE у SQL);
– annotate() – додавання агрегованих полів до QuerySet;
– aggregate() – обчислення агрегованого значення для всього QuerySet;
– values() – вибірка лише окремих полів (еквівалент SELECT col1, col2);
– select_related() – оптимізація запитів через JOIN для ForeignKey;
– order_by() – сортування результатів;
– _month, _year, _icontains – спеціальні lookup-операції для фільтрації.

Приклад реального агрегаційного запиту з проєкту для отримання витрат за категоріями:

```
expense_by_category = Transaction.objects.filter(
    user=request.user,
    type="expense",
    date__month=month,
    date__year=year
).values(
    "category__name",
    "category__color",
    "category__icon"
).annotate(
    total=Sum("amount")
).order_by("-total")[:6]
```

Django транслює цей Python-код в один оптимізований SQL-запит із GROUP BY та ORDER BY. Це забезпечує ефективну роботу навіть при тисячах транзакцій, оскільки агрегація виконується на рівні СУБД, а не в пам'яті Python.

2.1.4 Метод генерації аналітичних рекомендацій

Для автоматичного формування персоналізованих фінансових рекомендацій реалізовано метод порогових значень із класифікацією на три категорії стану бюджету. Алгоритм базується на розрахунку норми заощаджень (savings rate, SR) – ключового показника фінансового здоров'я домогосподарства.

Норма заощаджень розраховується за формулою:

$$SR = \frac{(Income - Expense)}{Income} \times 100\% \quad (2.1)$$

де Income – загальні доходи за поточний місяць;

Expense – загальні витрати за поточний місяць.

Класифікацію фінансового стану за нормою заощаджень наведено у табл. 2.3.

Таблиця 2.3 – Класифікація фінансового стану за нормою заощаджень

Значення SR	**Категорія**	**Тип повідомлення**	**Зміст рекомендації**
SR ≥ 20 %	Норма	Успішне (зелене)	Відображається поточний відсоток заощаджень, похвала за фінансову дисципліну
0 % < SR < 20 %	Попередження	Попереджувальне (жовте)	Рекомендація збільшити частку заощаджень до 20 % від доходу
SR ≤ 0 %	Критично	Критичне (червоне)	Попередження про перевищення витрат над доходами із вказанням суми перевищення

Додатково алгоритм визначає категорію з найбільшими витратями за поточний місяць та включає її назву та суму до рекомендаційного повідомлення.

Це надає користувачеві конкретну, а не абстрактну пораду щодо оптимізації бюджету.

2.1.5 Метод побудови графіків та візуалізації даних

Для побудови інтерактивних графіків у браузері використовується бібліотека Chart.js. Взаємодія між серверним Python-кодом та клієнтським JavaScript відбувається через серіалізацію даних у формат JSON за допомогою вбудованого модуля json:

```
# views.py — серіалізація даних для Chart.js
chart_data = _get_monthly_chart_data(user, 6)
context = {
    "chart_data": json.dumps(chart_data),  # Python dict → JSON-рядок
}

// dashboard.html — отримання даних у JavaScript
const chartData = {{ chart_data|safe }};
new Chart(ctx, { data: { labels: chartData.labels, ... } });
```

Такий підхід дозволяє передавати будь-яку кількість серій даних без додаткових AJAX-запитів: всі дані вбудовуються безпосередньо в HTML-сторінку під час рендерингу на сервері. Це зменшує кількість мережевих запитів і прискорює початкове завантаження сторінки.

2.2 Технологічний стек розробки системи

2.2.1 Середовище розробки VS Code

Visual Studio Code – безкоштовний редактор коду з відкритим вихідним кодом від Microsoft [14]. Для роботи з проєктом використовуються розширення: Python (Microsoft), Pylance (статичний аналіз типів), Django (підсвічування шаблонів), GitLens (розширена робота з Git).

2.2.1 Мова програмування Python

Для реалізації серверної частини веб-сервісу обрано Python 3.10+. Python є інтерпретованою мовою програмування загального призначення з динамічною типізацією, чистим синтаксисом і розвиненою екосистемою пакетів [15, 16].

Порівняння мов програмування для серверної веб-розробки наведено у табл. 2.4.

Таблиця 2.4 – Порівняння мов програмування для серверної веб-розробки

Характеристика	**Python**	**Node.js**	**PHP**	**Java**
Синтаксис	Мінімалістичний, читабельний	Подібний до C/Java	Подібний до C	Суворий, докладний
Поріг входження	Низький	Середній	Середній	Високий
Головний веб-фреймворк	Django, Flask, FastAPI	Express, NestJS	Laravel, Symfony	Spring Boot
Якість ORM	Висока (Django ORM)	Середня (Sequelize)	Висока (Eloquent)	Висока (Hibernate)
Бібліотеки аналітики/звітів	Відмінні (Pandas, ReportLab)	Обмежені	Обмежені	Хороші (Apache POI)
Продуктивність (CPU)	Середня	Висока	Середня	Висока
Час розробки прототипу	Мінімальний	Малий	Малий	Значний
Спільнота та документація	Дуже велика	Велика	Велика	Дуже велика

Ключові можливості Python, які безпосередньо використовуються у проєкті:

– декоратори – @login_required для захисту всіх View-функцій від неавторизованого доступу;

– генераторні вирази та list comprehensions – для компактної обробки колекцій даних;

– контекстні менеджери (with) – при роботі з файловими буферами під час генерації PDF та Excel;

– f-рядки (f-strings) – для форматування рядків у генераторах звітів;

– модуль io.BytesIO – для формування файлів у пам'яті без збереження на диск;

– модуль json – для серіалізації даних графіків у формат JSON;

– модуль datetime / timezone – для роботи з датами та часовими зонами;

– обробка виключень (try/except) – при пошуку TTF-шрифтів для PDF-генерації.

2.2.2 Веб-фреймворк Django

Django – це високорівневий вебфреймворк Python, що дотримується принципу «батарейки включені» (batteries-included): він надає готові рішення для більшості типових задач веброзробки [17, 18]. Версія Django 5.0, обрана для проєкту, була випущена у грудні 2023 р. і є поточним стабільним релізом.

Порівняно з мінімалістичними фреймворками (Flask, FastAPI), Django надає значно більше готового функціоналу «з коробки», що скорочує час розробки. Порівняльна характеристика веб-фреймворків Python наведена у табл. 2.5.

Таблиця 2.5 – Порівняння вебфреймворків Python

Характеристика	**Django 5.0**	**Flask 3.x**	**FastAPI 0.x**
Парадигма	Full-stack, batteries included	Мікрофреймворк	API-орієнтований
ORM	Вбудований (Django ORM)	Відсутній (потрібен SQLAlchemy)	Відсутній (потрібен SQLAlchemy)
Аутентифікація	Вбудована	Потрібна бібліотека	Потрібна бібліотека

Кінець таблиці 2.5

Характеристика	Django 5.0	Flask 3.x	FastAPI 0.x
Адміністративна панель	Вбудована (/admin/)	Відсутня	Відсутня
Шаблонізатор	Вбудований (DTL)	Jinja2 (підключається)	Jinja2 (підключається)
Захист від CSRF	Вбудований	Потрібна бібліотека	Вбудований (для форм ні)
Міграції БД	Вбудовані	Потрібен Alembic	Потрібен Alembic
Складність налаштування	Мінімальна	Мінімальна	Мінімальна
Доцільність для даного проєкту	Висока	Середня	Низька

На основі порівняльного аналізу (табл. 2.5) Django є оптимальним вибором для даного проєкту. Завдяки вбудованим ORM, системі аутентифікації, адміністративній панелі та захисту від CSRF, час розробки суттєво скорочується порівняно з мінімалістичними фреймворками.

Ключові компоненти Django та їх використання у проєкті наведено у табл. 2.6.

Таблиця 2.6 – Ключові компоненти Django та їх використання у проєкті

Компонент Django	Модуль	Використання у проєкті
ORM	django.db.models	Моделі Transaction, Category, Budget; всі запити до БД
Аутентифікація	django.contrib.auth	Реєстрація, вхід, @login_required, хешування паролів
Форми	django.forms	TransactionForm, CategoryForm, BudgetForm, LoginForm, RegisterForm

Кінець таблиці 2.6

Компонент Django	**Модуль**	**Використання у проєкті**
Шаблонізатор	django.template	15 HTML-шаблонів із тегами if, for, url, static, safe
URL-маршрутизація	django.urls	18 URL-маршрутів застосунку у finance_app/urls.py
Міграції	django.db.migrations	Версіонування схеми БД; автоматична синхронізація
Повідомлення	django.contrib.messages	Flash-повідомлення про успішні операції та помилки
Адмін-панель	django.contrib.admin	Управління даними через /admin/ для адміністратора
Захист	django.middleware.csrf	CSRF-токени у всіх POST-формах
Пагінація	django.core.paginator	Розбиття журналу транзакцій по 15 записів на сторінку

2.2.3 Система шаблонів Django (DTL)

Django Template Language (DTL) − це мова розмітки шаблонів Django, що дозволяє вбудовувати динамічний вміст у HTML-сторінки за допомогою спеціальних тегів і фільтрів. DTL свідомо обмежений у своїх можливостях: він не дозволяє виконувати довільний Python-код у шаблонах, що забезпечує чітке розмежування бізнес-логіки та представлення.

Основні можливості DTL, що використовуються у проєкті наведено у табл. 2.7.

Таблиця 2.7 – Теги та фільтри DTL, що використовуються у проєкті

Тег / фільтр	**Призначення**	**Приклад використання**
{% extends %}	Успадкування шаблону від батьківського	{% extends "finance_app/base.html" %}
{% block %}	Визначення замінюваного блоку	{% block content %} ... {% endblock %}
{% for %}	Ітерація по колекції	{% for t in page_obj %} ... {% endfor %}
{% if %}	Умовне відображення	{% if t.type == "income" %} ... {% endif %}
{% url %}	Генерація URL за назвою маршруту	{% url "transaction_edit" t.pk %}
{% csrf_token %}	Вставка CSRF-токену у форму	У кожній POST-формі обов'язково
{{ var|floatformat:0 }}	Форматування числа (без дробової частини)	{{ balance|floatformat:0 }} ₴
{{ var|safe }}	Виведення без екранування HTML/JS	{{ chart_data|safe }} у script-блоці
{{ var|default:"—" }}	Значення за замовчуванням якщо порожньо	{{ t.description|default:"—" }}
{{ var|truncatechars:40 }}	Обрізання рядка до N символів	{{ t.description|truncatechars:40 }}

Ключовою особливістю організації шаблонів є механізм успадкування. Базовий шаблон base.html визначає спільну структуру сторінки (sidebar, topbar, підключення CSS та JS). Усі дочірні шаблони розширюють його через {% extends

%} і перевизначають лише потрібні блоки (page_title, content, extra_js), що виключає дублювання розмітки.

2.3 Проєктування бази даних

2.3.1 Обґрунтування вибору реляційної моделі

Для зберігання даних вебсервісу обрано реляційну модель бази даних. Реляційна модель представляє дані у вигляді пов'язаних таблиць (відношень) і забезпечує цілісність даних через механізм зовнішніх ключів та обмежень.

Вибір реляційної моделі обґрунтований структурою даних предметної галузі: транзакції пов'язані з категоріями та користувачами через чіткі відношення «один до багатьох». Реляційна модель природно відображає ці відношення та гарантує цілісність: неможливо зберегти транзакцію для неіснуючого користувача.

Порівняльна характеристика реляційної та NoSQL моделей бази даних наведено у табл. 2.8.

Таблиця 2.8 – Порівняння реляційної та NoSQL моделей для задачі проєкту

Критерій	**Реляційна БД (SQLite)**	**NoSQL (MongoDB)**
Структура даних	Жорстка схема (таблиці)	Гнучка схема (документи)
Зв'язки між сутностями	ForeignKey, JOIN – нативно	Вбудовані документи або ручні посилання
Підтримка Django ORM	Повна	Часткова (через mongoengine)
Агрегаційні запити	SQL: GROUP BY, SUM – нативно	Aggregation Pipeline – складніше
ACID-транзакції	Повна підтримка	Обмежена
Відповідність задачі	Висока	Середня

Структурованість фінансових даних (фіксований набір полів транзакції, чіткі відношення між сутностями) робить реляційну модель оптимальним вибором.

NoSQL доцільний для неструктурованих або ієрархічних даних, що не є характерним для даної задачі.

2.3.2 СУБД SQLite

SQLite − вбудована файлова реляційна СУБД, що зберігає всю базу даних у єдиному файлі (db.sqlite3). SQLite повністю підтримує стандарт SQL:2016 і забезпечує відповідність вимогам ACID (Atomicity, Consistency, Isolation, Durability) [19]. Порівняльна характеристика СУБД для Django-проєктів наведена у табл. 2.9.

Таблиця 2.9 – Порівняльна характеристика СУБД для Django-проєктів

Характеристика	**SQLite**	**PostgreSQL**	**MySQL**	**MariaDB**
Тип	Вбудована, файлова	Клієнт-серверна	Клієнт-серверна	Клієнт-серверна
Налаштування	Нульова конфігурація	Потребує сервера	Потребує сервера	Потребує сервера
Зберігання	Один .sqlite3 файл	Системний каталог	Системний каталог	Системний каталог
Паралельні з'єднання	Одне на запис	Багато	Багато	Багато
JSON-типи	Обмежено	Нативно	Так (5.7+)	Так (10.2+)
Підтримка Django ORM	Повна	Повна	Повна	Повна
Рекомендований обсяг	До ~100 тис. записів	Необмежено	Необмежено	Необмежено
Розгортання	+ (один файл)	– (сервер)	– (сервер)	– (сервер)

SQLite обрано з огляду на цільовий сценарій використання: персональний або сімейний фінансовий менеджер з одним активним користувачем. За таких умов SQLite повністю задовольняє вимогам продуктивності та надійності. При необхідності масштабування (наприклад, розгортання як SaaS-сервісу) перехід на PostgreSQL потребує зміни лише одного рядка у файлі settings.py:

```
# settings.py — перехід на PostgreSQL (без зміни будь-якого іншого
коду)
DATABASES = {
    "default": {
        "ENGINE": "django.db.backends.postgresql",
        "NAME": "financedb",
        "USER": "admin",
        "PASSWORD": "секретний_пароль",
        "HOST": "localhost",
        "PORT": "5432",
    }
}
```

2.3.3 Нормалізація бази даних

Схема бази даних приведена до третьої нормальної форми (3НФ), що виключає надлишковість даних та аномалії оновлення. Нижче наведено обґрунтування відповідності кожній нормальній формі.

Перша нормальна форма (1НФ): усі атрибути таблиць є атомарними (неподільними). Жодне поле не містить списків або складених значень. Кожна таблиця має первинний ключ (поле id типу BigAutoField).

Друга нормальна форма (2НФ): кожен неключовий атрибут повністю залежить від первинного ключа. Оскільки всі таблиці мають простий (одноколонковий) первинний ключ, 2НФ виконується автоматично.

Третя нормальна форма (3НФ): відсутні транзитивні залежності між неключовими атрибутами. Наприклад, назва та колір категорії зберігаються у таблиці Category, а не дублюються у Transaction. У Transaction зберігається лише зовнішній ключ category_id.

2.4 Проєктування захисту інформації

2.4.1 Аналіз загроз безпеці

Веб-застосунки є об'єктами численних типів атак. Для даного проєкту проведено аналіз актуальних загроз відповідно до переліку OWASP Top 10 та визначено заходи протидії, реалізовані у системі (табл. 2.10).

Таблиця 2.10 – Аналіз загроз та заходи захисту у проєкті

Загроза	Опис ризику	Захід протидії у проєкті
SQL-ін'єкція	Вставка шкідливого SQL-коду у параметри запиту	Django ORM автоматично параметризує всі запити; сирий SQL не використовується
CSRF-атака	Підробка запиту від імені авторизованого користувача	Django CsrfViewMiddleware + {% csrf_token %} у кожній POST-формі
Перехоплення сесії	Викрадення session cookie для підміни ідентифікації	SESSION_COOKIE_HTTPONLY=True; SESSION_COOKIE_SECURE=True (HTTPS)
Несанкціонований доступ	Доступ до захищених сторінок без авторизації	Декоратор @login_required на всіх View, крім login/register
Витік даних між акаунтами	Доступ користувача А до даних користувача Б	Фільтр user=request.user у всіх ORM-запитах; GET 404 для чужих даних
Ненадійне зберігання паролів	Паролі у відкритому вигляді у БД	Django використовує PBKDF2-SHA256 + salt для хешування паролів
XSS (Міжсайтовий скриптинг)	Вставка шкідливого JS через поля вводу	Django DTL автоматично екранує змінні; |safe лише для довірених даних

2.4.2 Аутентифікація та управління сесіями

Система аутентифікації базується на вбудованому модулі django.contrib.auth. Процес реєстрації використовує форму UserCreationForm, яка забезпечує:

– перевірку складності пароля (мінімум 8 символів, заборона поширених паролів, заборона паролю, що збігається з іменем користувача);

– перевірку збігу двох введених паролів;

– перевірку унікальності логіна в межах системи.

Хешування паролів виконується алгоритмом PBKDF2 (Password-Based Key Derivation Function 2) з хеш-функцією SHA-256. Алгоритм використовує 870 000 ітерацій та унікальну сіль (salt) для кожного пароля, що унеможливлює атаки за допомогою райдужних таблиць.

Управління сесіями здійснюється через механізм сесій Django: після успішної аутентифікації у БД створюється запис сесії, а у браузері встановлюється зашифрований cookie з ідентифікатором сесії. При виході з системи (logout) сесійний запис видаляється з БД.

2.5 Засоби формування звітності

2.5.1 Бібліотека ReportLab для генерації PDF

ReportLab – Python-бібліотека для програмної генерації PDF-документів стандарту ISO 32000 [20]. Версія 4.2, що використовується у проєкті, є актуальним стабільним релізом. Бібліотека надає два рівні API:

– низькорівневий (Canvas API) − прямий контроль над розташуванням елементів на сторінці у координатах (x, y);

– високорівневий (Platypus) − потоковий спосіб компонування документа з абзаців, таблиць та інших об'єктів-«Flowable».

У проєкті використовується Platypus API, оскільки він дозволяє автоматично розбивати таблиці з транзакціями на кілька сторінок без ручного розрахунку координат. Основні компоненти Platypus, що застосовуються у проєкті, наведено у табл. 2.11.

Таблиця 2.11 – Компоненти ReportLab Platypus у проєкті

Компонент	Клас	Використання
Документ	SimpleDocTemplate	Контейнер документа; визначає розміри сторінки та поля

Кінець таблиці 2.11

Компонент	Клас	Використання
Абзац	Paragraph	Заголовки та підписи; підтримує теги форматування
Стиль абзацу	ParagraphStyle	Визначення шрифту, розміру, кольору тексту
Таблиця	Table	Таблиця зведеної статистики та таблиця транзакцій
Стиль таблиці	TableStyle	Заливка, рамки, вирівнювання клітинок
Відступ	Spacer	Вертикальні відступи між елементами

Особливою задачею при генерації PDF є забезпечення відображення кирилиці. Стандартні шрифти ReportLab (Helvetica, Times-Roman) підтримують лише Latin-1 символи. Для вирішення цієї проблеми реалізовано механізм автоматичного пошуку TrueType-шрифту з підтримкою Unicode на трьох платформах:

```
font_paths = [
    # Windows
    "C:/Windows/Fonts/arial.ttf",
    # Linux
    "/usr/share/fonts/truetype/dejavu/DejaVuSans.ttf",
    # macOS
    "/Library/Fonts/Arial.ttf",
]
for path in font_paths:
    if os.path.exists(path):
        pdfmetrics.registerFont(TTFont("CyrFont", path))
        break
```

2.5.2 Бібліотека OpenPyXL для генерації Excel

OpenPyXL − Python-бібліотека для читання та запису файлів Microsoft Excel форматів .xlsx та .xlsm. Бібліотека підтримує всі елементи форматування Excel і є

рекомендованим інструментом для програмної роботи з Excel-файлами у Python-проєктах [21].

Ключові об'єкти OpenPyXL, що використовуються у проєкті, наведено у табл. 2.12.

Таблиця 2.12 – Об'єкти OpenPyXL та їх застосування у проєкті

Об'єкт	**Клас OpenPyXL**	**Застосування**
Книга Excel	openpyxl.Workbook	Створення нової Excel-книги у пам'яті
Аркуш	Worksheet	Аркуші «Транзакції» та «Зведення»
Клітинка	Cell (ws.cell())	Запис значень та застосування форматування
Шрифт	Font	Жирний білий шрифт для заголовків; жирний для підсумків
Заливка	PatternFill	Синя (#1E40AF) для заголовків; зелена/червона для рядків
Рамка	Border + Side	Тонка рамка навколо всіх клітинок таблиці
Вирівнювання	Alignment	Горизонтальне центрування для заголовків
Ширина стовпця	column_dimensions	Встановлення ширини кожного стовпця

Перевага OpenPyXL перед генерацією CSV полягає у збереженні кольорового форматування, числових форматів та структури аркушів. Це дозволяє створити документ, який виглядає як професійно підготовлений звіт, а не сирий дамп даних.

2.5.3 Бібліотека Chart.js

Chart.js – JavaScript-бібліотека для побудови інтерактивних графіків у браузері. Підключається через CDN, що не потребує встановлення на сервері. Підтримує різні типи діаграм: стовпчасті, лінійні, кругові, точкові [22]. У проєкті використовується для відображення чотирьох типів графіків: стовпчастої діаграми

місячної динаміки, лінійного графіку щоденної активності та двох кругових діаграм розподілу витрат і доходів за категоріями.

2.6 Проєктування інтерфейсу користувача

2.6.1 Принципи проєктування UI/UX

Інтерфейс веб-сервісу спроєктовано з дотриманням базових принципів UI/UX-дизайну. Основні принципи, що застосовувались:

– принцип мінімальної когнітивної навантаженості – інтерфейс не перевантажений елементами; кожна сторінка вирішує одне конкретне завдання;

– принцип наочності – числові показники доповнено кольоровими індикаторами (зелений/жовтий/червоний), що дозволяють зрозуміти стан бюджету без читання цифр;

– принцип консистентності – однакові елементи однаково виглядають і поводяться на всіх сторінках;

– принцип зворотного зв'язку – після кожної дії (додавання, редагування, видалення) відображається flash-повідомлення про результат операції;

– принцип прогресивного розкриття – складні функції (статистика, бюджети) доступні у відповідних розділах, а не перевантажують головну сторінку.

2.6.2 Кольорова схема інтерфейсу

Кольорова схема системи базується на CSS Custom Properties (CSS-змінних) (див. табл. 2.13), що забезпечує централізоване управління кольорами та можливість зміни теми без редагування шаблонів.

Таблиця 2.13 – Кольорова схема інтерфейсу

Змінна CSS	Hex-код	Призначення
--primary	#1E40AF	Основний колір бренду; sidebar; кнопки
--primary-light	#3B82F6	Акценти; посилання; border при фокусі

Кінець таблиці 2.13

Змінна CSS	Нех-код	Призначення
--primary-dark	#1E3A8A	Hover-стан для кнопок та пунктів меню
--success	#10B981	Доходи; позитивний баланс; норма бюджету
--danger	#EF4444	Витрати; від'ємний баланс; перевищення бюджету
--warning	#F59E0B	Попередження; бюджет 70–90 %
--bg	#F1F5F9	Фон сторінки
--card	#FFFFFF	Фон карток
--text	#1E293B	Основний колір тексту
--text-muted	#64748B	Другорядний текст, дати, підписи
--border	#E2E8F0	Рамки карток та полів вводу

Типографіка базується на системному шрифті Segoe UI (Windows) з fallback на system-ui та -apple-system. Використання системних шрифтів виключає необхідність завантаження шрифтових файлів та прискорює початкове відображення сторінки.

2.6.3 Адаптивний дизайн

Адаптивний дизайн реалізований за допомогою CSS Grid та медіа-запитів. Система використовує три breakpoints для адаптації макету:

– desktop (> 1024 px) – статистичні картки у 4 колонки, двоколонкові блоки графіків та таблиць;

– tablet (768–1024 px) – картки у 2 колонки, бічна панель залишається;

– mobile (< 768 px) – всі елементи у 1 колонку, бічна навігаційна панель прихована.

2.7 Зведена характеристика технологічного стеку

На підставі проведеного аналізу методів, підходів та технологій у даному розділі складено зведену характеристику технологічного стеку вебсервісу (табл. 2.14).

Таблиця 2.14 – Зведена характеристика технологічного стеку

Компонент	**Технологія**	**Версія**	**Обґрунтування вибору**
Мова (backend)	Python	3.10+	Чистий синтаксис, відмінна екосистема, нативна інтеграція з Django
Веб-фреймворк	Django	5.0	Batteries-included: ORM, Auth, Admin, CSRF без сторонніх бібліотек
СУБД	SQLite	3.x	Нульова конфігурація; достатня для персонального застосунку; легко масштабується до PostgreSQL
PDF-генерація	ReportLab	4.2	Потужний Platypus API; підтримка Unicode через TTF-шрифти
Excel-генерація	OpenPyXL	3.1	Повна підтримка форматування .xlsx; рекомендований для Django
Графіки (frontend)	Chart.js	4.4	Легка JS-бібліотека (< 60 KB); 4 типи діаграм; CDN
CSS	Custom CSS + Grid	–	Без зовнішніх CSS-фреймворків; CSS-змінні для теми; адаптивність
IDE	Visual Studio Code	Latest	Python + Django розширення; вбудований термінал; Git-інтеграція

Обраний технологічний стек забезпечує виконання всіх функціональних та нефункціональних вимог, визначених у розділі 1. Відсутність зовнішніх CSS-фреймворків (Bootstrap, Tailwind) зменшує розмір завантажуваних ресурсів; підключення Chart.js через CDN виключає необхідність npm-збірки; використання SQLite спрощує розгортання до однієї команди python manage.py runserver.

Висновки до розділу 2

У другому розділі обґрунтовано теоретичну та технологічну основу розробки вебсервісу. Розглянуто та описано такі аспекти:

– архітектурний патерн MTV та принципи SOLID з конкретними прикладами їх застосування у проєкті (табл. 2.1, 2.2);

– методи роботи з даними через Django ORM – агрегаційні запити, фільтрація, оптимізація через select_related;

– алгоритм генерації аналітичних рекомендацій на основі норми заощаджень із трирівневою класифікацією (табл. 2.3);

– порівняльний аналіз мов програмування (табл. 2.4), вебфреймворків (табл. 2.5) та СУБД (табл. 2.8, 2.9) – Python/Django/SQLite визнані оптимальними для задачі;

– система шаблонів Django DTL (табл. 2.7) та механізм успадкування шаблонів;

– нормалізацію БД до 3НФ та обґрунтування реляційної моделі (табл. 2.8);

– заходи захисту від актуальних загроз OWASP Top 10 (табл. 2.10) та механізм хешування паролів PBKDF2-SHA256;

– технічні характеристики ReportLab (табл. 2.11) та OpenPyXL (табл. 2.12) для реалізації експорту;

– принципи проєктування UI/UX, кольорову схему (табл. 2.13) та адаптивний дизайн.

Зведена характеристика технологічного стеку (табл. 2.14) підтверджує, що обраний набір технологій є цілісним, взаємно сумісним та оптимальним для реалізації поставленої задачі.

3 РОЗРОБКА ВЕБСЕРВІСУ ТА АНАЛІЗ РЕЗУЛЬТАТІВ

3.1 Архітектура та структура системи

3.1.1 Загальна структура проєкту

Проєкт організовано відповідно до стандартної структури Django-застосунку. Кореневий каталог finance_project містить конфігураційний пакет та основний застосунок. Файлова структура представлена у таблиці 3.1.

Таблиця 3.1 – Файлова структура проєкту

Файл / директорія	Призначення
finance_project/settings.py	Конфігурація: БД, застосунки, безпека, шляхи до файлів, мова
finance_project/urls.py	Головний маршрутизатор, підключає маршрути finance_app
finance_project/wsgi.py	WSGI-інтерфейс для розгортання на продакшн-сервері
finance_app/models.py	Визначення трьох моделей: Transaction, Category, Budget
finance_app/views.py	Обробники HTTP-запитів (18 View-функцій)
finance_app/forms.py	Django-форми для валідації вхідних даних (6 форм)
finance_app/urls.py	Маршрутизація URL-адрес застосунку (18 маршрутів)
finance_app/admin.py	Реєстрація моделей в адмін-панелі Django
finance_app/migrations/	Автоматично згенеровані файли міграцій бази даних
finance_app/templates/	HTML-шаблони для всіх сторінок (15 файлів)
requirements.txt	Перелік залежностей проєкту (5 пакетів)
manage.py	Утиліта управління Django-проєктом
db.sqlite3	Файл бази даних SQLite

3.1.2 Маршрутизація URL

Система маршрутизації URL забезпечує зручні RESTful-адреси для кожного функціонального розділу. Повний перелік URL-маршрутів наведено у таблиці 3.2.

Таблиця 3.2 – Маршрутизація URL-адрес веб-сервісу

URL-адреса	HTTP-метод	View-функція	Опис
/login/	GET, POST	login_view	Форма авторизації
/register/	GET, POST	register_view	Форма реєстрації
/logout/	POST	logout_view	Вихід із системи
/dashboard/	GET	dashboard	Головна інформаційна панель
/transactions/	GET	transaction_list	Журнал транзакцій з фільтрацією
/transactions/add/	GET, POST	transaction_add	Додавання транзакції
/transactions/<pk>/edit/	GET, POST	transaction_edit	Редагування транзакції
/transactions/<pk>/delete/	GET, POST	transaction_delete	Видалення транзакції
/categories/	GET	category_list	Список категорій
/categories/add/	GET, POST	category_add	Додавання категорії
/categories/<pk>/edit/	GET, POST	category_edit	Редагування категорії
/statistics/	GET	statistics	Статистика та аналітика
/budgets/	GET	budget_list	Список бюджетів

Кінець таблиці 3.2

URL-адреса	HTTP-метод	View-функція	Опис
/budgets/add/	GET, POST	budget_add	Додавання бюджету
/export/excel/	GET	export_excel	Завантаження Excel-файлу
/export/pdf/	GET	export_pdf	Завантаження PDF-звіту
/api/categories/	GET	api_categories	JSON API для AJAX-запитів

3.1.3 Діаграма варіантів використання системи

На рисунку 3.1 наведена спрощена діаграма варіантів використання системи (Use Case Diagram), що відображає взаємодію між користувачем та основними функціональними можливостями вебсервісу.

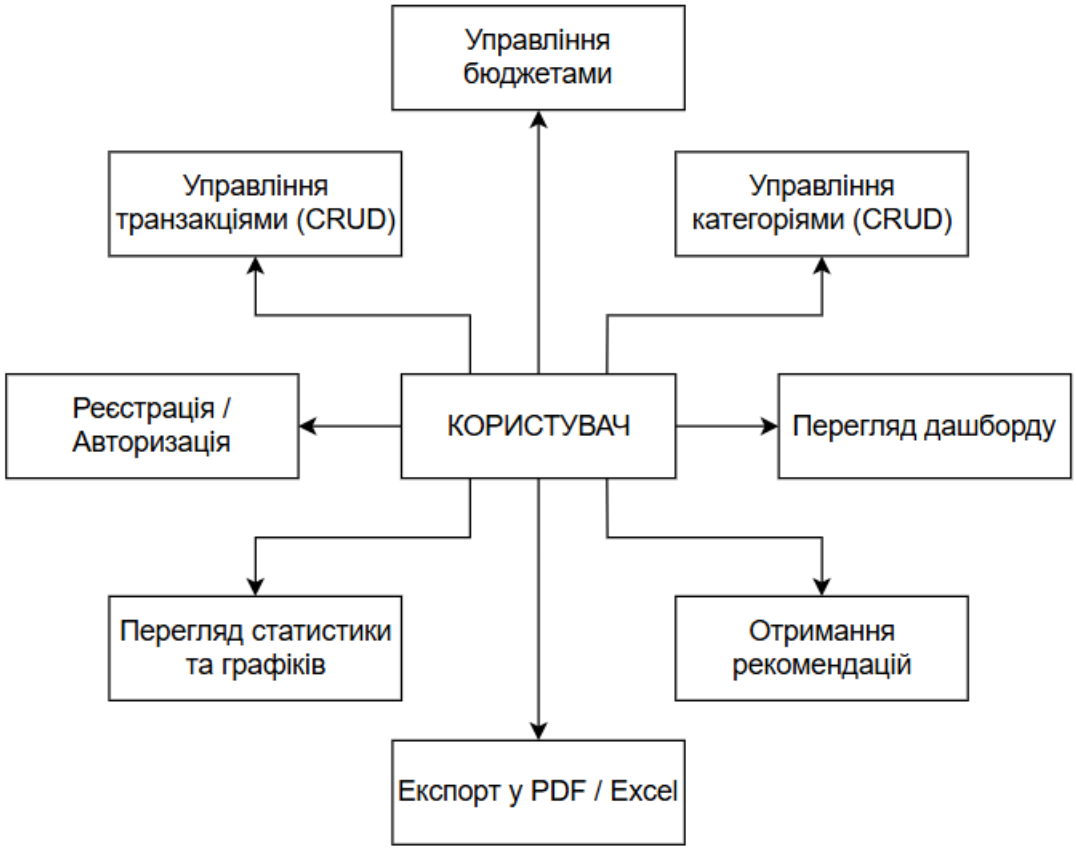


Рисунок 3.1 – Спрощена діаграма варіантів використання системи

3.1.3 Блок-схема загального алгоритму роботи

На рисунку 3.2 представлена блок-схема загального алгоритму обробки HTTP-запиту в системі.

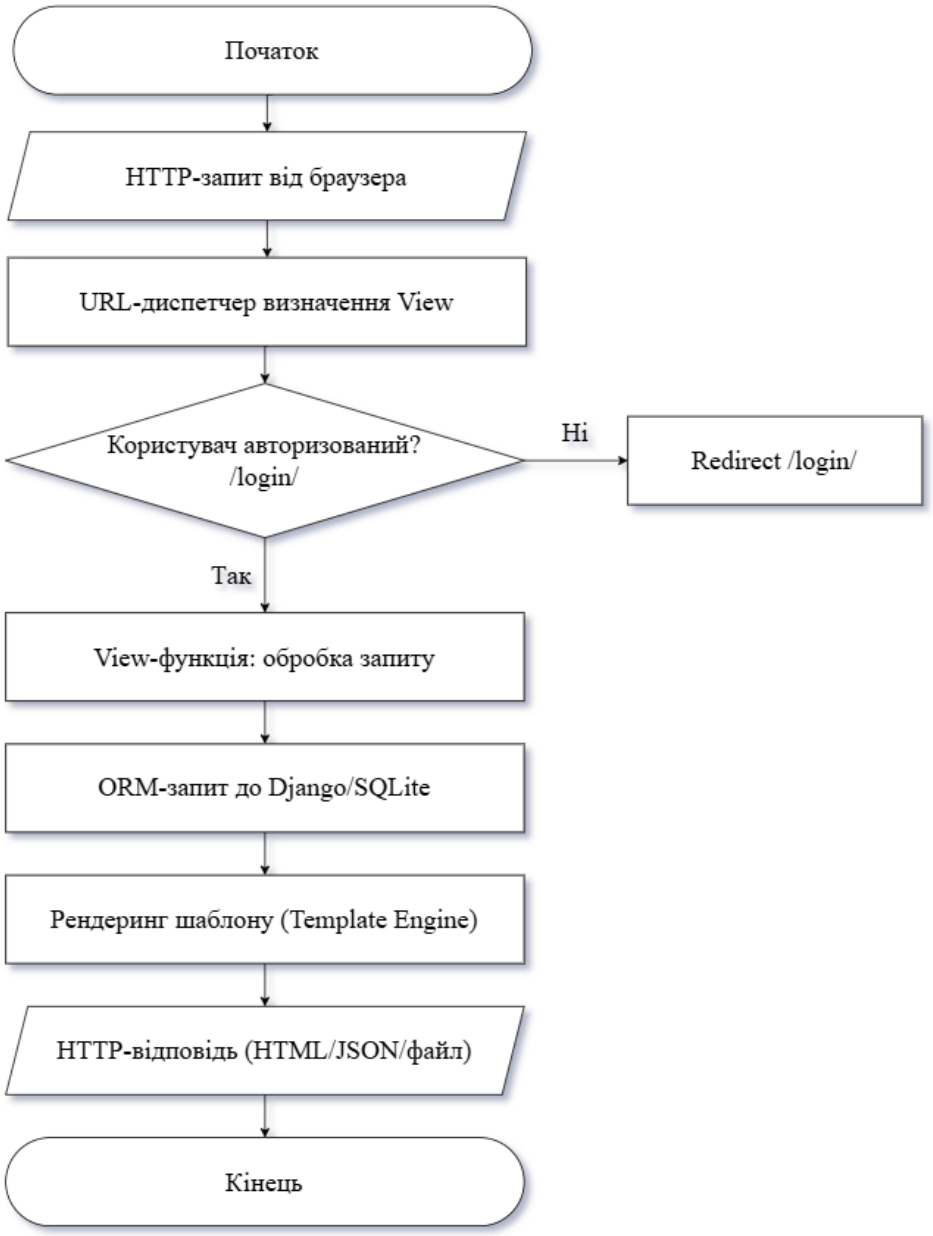


Рисунок 3.2 – Блок-схема обробки HTTP-запиту у системі

3.2 Розробка моделей бази даних

3.2.1 Опис моделей

Модель Category описує категорію фінансових операцій. Обмеження unique_together на полях (user, name, type) гарантує відсутність дублікатів категорій одного типу в межах одного акаунта.

Модель Transaction є центральною сутністю системи. Поле amount визначено як DecimalField з параметрами max_digits=12, decimal_places=2 для уникнення помилок округлення, характерних для типу float. Поле category дозволяє NULL (параметр null=True, blank=True), що дає можливість зберігати транзакції без категорії. При видаленні категорії пов'язані транзакції зберігаються, але поле category стає NULL (on_delete=SET_NULL).

Модель Budget містить два обчислювані методи: spent() – повертає суму фактичних витрат за відповідний місяць через агрегаційний запит ORM; percentage() – розраховує відсоток використання бюджету для відображення прогрес-бару.

В таблиці 3.3 наведена специфікація моделі Transaction.

Таблиця 3.3 – Специфікація моделі Transaction

Поле	**Тип**	**Параметри**	**Опис**
id	BigAutoField	primary_key=True	Первинний ключ (автоматично)
user	ForeignKey	User, on_delete=CASCADE	Власник транзакції
category	ForeignKey	Category, null=True, on_delete=SET_NULL	Категорія (необов'язково)
type	CharField	max_length=10, choices	Тип: income / expense
amount	DecimalField	max_digits=12, decimal_places=2	Сума операції
description	CharField	max_length=255, blank=True	Опис (необов'язково)

Кінець таблиці 3.3

Поле	Тип	Параметри	Опис
Date	DateField	default=timezone.now	Дата операції
created_at	DateTimeField	auto_now_add=True	Час створення запису
updated_at	DateTimeField	auto_now=True	Час останнього оновлення

3.3 Реалізація бізнес-логіки (Views та Forms)

3.3.1 View dashboard()

View dashboard() є центральною View-функцією системи. Вона виконує 9 послідовних операцій для формування контексту головної сторінки:

1) Отримання поточної дати (timezone.now()).
2) Фільтрація транзакцій поточного місяця.
3) Агрегація: total_income = SUM (amount) WHERE type = ‘income’.
4) Агрегація: total_expense = SUM (amount) WHERE type = ‘expense’.
5) Розрахунок балансу та загального балансу за весь час.
6) Отримання 8 останніх транзакцій (SELECT … LIMIT 8).
7) Групування витрат за категоріями (GROUP BY Category)
8) _get_monthly_chart_data () → дані для Chart.js (6 місяців)
9) _generate_tips () → аналітичні рекомендації

3.3.2 View transaction_list() з фільтрацією

View transaction_list() реалізує журнал транзакцій із гнучкою системою фільтрації. Алгоритм застосовує фільтри методом послідовного звуження вибірки: спочатку вибираються всі транзакції користувача, потім послідовно додаються умови WHERE відповідно до заповнених полів форми фільтру.

Реалізовано такі фільтри: за типом операції (income/expense); за категорією; за датою «від» (date_gte); за датою «до» (date_lte); за ключовим словом в описі (description_icontains, регістронезалежний пошук).

3.3.3 Форми системи

Всі форми вводу реалізовані через Django Forms, що забезпечує серверну валідацію даних та захист від некоректних введень. Перелік форм системи та їх призначення наведено в таблиці 3.4.

Таблиця 3.4 – Форми системи та їх призначення

Клас форми	Базовий клас	Поля	Призначення
RegisterForm	UserCreationForm	username, first_name, last_name, email, password1, password2	Реєстрація нового користувача
LoginForm	AuthenticationForm	username, password	Авторизація
TransactionForm	ModelForm	type, amount, category, description, date	Додавання/редагування транзакції
CategoryForm	ModelForm	name, type, icon, color	Додавання/редагування категорії
BudgetForm	ModelForm	category, limit, month, year	Встановлення бюджету
TransactionFilterForm	Form	type, category, date_from, date_to, search	Фільтрація журналу транзакцій

3.4 Розробка шаблонів інтерфейсу

3.4.1 Структура шаблонів

Інтерфейс системи побудований на основі системи успадкування шаблонів Django. Базовий шаблон base.html визначає загальну структуру: бічну навігаційну панель (sidebar), верхній рядок (topbar) та область основного вмісту. Всі інші

шаблони розширюють base.html. На рисунку 3.3 зображено структуру шаблонів інтерфейсу системи.

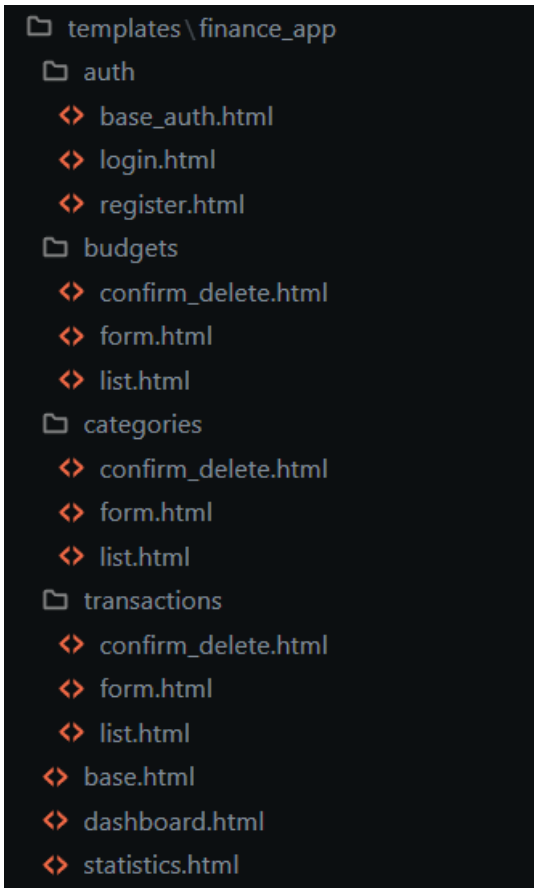


Рисунок 3.3 – Структура шаблонів інтерфейсу системи

3.4.2 Компоненти інтерфейсу

Дизайн системи базується на CSS Custom Properties (CSS-змінних), що забезпечує централізовану кольорову схему. В таблиці 3.5 наведені уніфіковані стилі для кожного типу інтерфейсного компонента.

Таблиця 3.5 – Компоненти інтерфейсу та їх призначення

Компонент	CSS-клас	Призначення	Кольорова схема
Картка статистики (дохід)	.stat-card.income	Відображення суми доходів	Зелений (#10B981)
Картка статистики (витрата)	.stat-card.expense	Відображення суми витрат	Червоний (#EF4444)

Кінець таблиці 3.5

Компонент	CSS-клас	Призначення	Кольорова схема
Картка статистики (баланс)	.stat-card.balance	Відображення балансу	Синій (#3B82F6)
Прогрес-бар (норма)	.progress-fill (зелений)	Бюджет < 70 %	#10B981
Прогрес-бар (попередження)	.progress-fill (жовтий)	Бюджет 70–90 %	#F59E0B
Прогрес-бар (критичний)	.progress-fill (червоний)	Бюджет > 90 %	#EF4444
Бейдж «дохід»	.badge-income	Маркування транзакцій	Зелений фон
Бейдж «витрата»	.badge-expense	Маркування транзакцій	Червоний фон

3.4.3 AJAX-взаємодія

Для динамічного оновлення списку категорій у формі транзакції реалізовано AJAX-запит до ендпойнту /api/categories/. При виборі типу операції (дохід або витрата) JavaScript-код відправляє GET-запит із параметром type, а сервер повертає JSON із відповідними категоріями. Це виключає необхідність перезавантаження сторінки при зміні типу операції.

3.5 Реалізація модулю експорту даних

3.5.1 Алгоритм генерації Excel-файлу

View export_excel() формує Excel-файл за допомогою OpenPyXL. Нижче наведено алгоритм роботи модулю:

1) Завантаження всіх транзакцій користувача з БД.
2) Створення нової книги Excel (openpyxl.Work ()).
3) Активація аркуша “Транзакції”.

4) Визначення стилів: синій заголовок, зелена/червона заливка.
5) Запис заголовкового рядка.
6) Встановлення ширини стовпців.
7) Цикл: запис рядків із застосуванням кольорових стилів.
8) Створення аркуша “Зведення” (дохід, витрати, баланс).
9) Запис книги у буфер пам’яті (io.BytesIO).
10) Повернення HttpResponse з Content-Type xlsx.

3.5.2 Алгоритм генерації PDF-звіту

View export_pdf() формує PDF-звіт за допомогою ReportLab. Особливою задачею є підтримка кирилиці. Реалізовано механізм автоматичного пошуку TrueType-шрифту: система перебирає список стандартних шляхів до TTF-файлів для Windows, Linux та macOS. При знаходженні підходящого шрифту він реєструється в ReportLab через pdfmetrics.registerFont(). Структура PDF-звіту наведена в таблиці 3.6.

Таблиця 3.6 – Структура PDF-звіту

Секція	Зміст	Бібліотечний компонент
Заголовок	Назва звіту, ім'я користувача, дата формування	Paragraph з ParagraphStyle
Зведення	Таблиця: загальний дохід, загальні витрати, баланс	Table з TableStyle (синя схема)
Деталі	Таблиця останніх 100 транзакцій (дата, тип, категорія, опис, сума)	Table з TableStyle (чергування рядків)

Висновки до розділу 3

У третьому розділі описано повну програмну реалізацію веб-сервісу. Спроєктовано архітектуру системи з 18 URL-маршрутами (табл. 3.2) та діаграму бази даних (рис. 3.3). Розроблено три моделі (Transaction, Category, Budget), 6 Django-форм (табл. 3.4), систему із 15 HTML-шаблонів та AJAX-взаємодію для

динамічного оновлення категорій. Реалізовано модулі генерації Excel (алгоритм рис. 3.6) та PDF зі підтримкою кирилиці. Система має функціонал дашборду, журналу транзакцій із фільтрацією, статистики з 4 типами графіків, бюджетування та автоматичних аналітичних рекомендацій.

4 ПРОГРАМНА РЕАЛІЗАЦІЯ ТА ТЕСТУВАННЯ

4.1 Керівництво користувача

4.1.1 Системні вимоги та встановлення

В таблиці 4.1 наведені мінімальні та рекомендовані вимоги для розгортання та запуску вебсервісу.

Таблиця 4.1 – Системні вимоги для запуску вебсервісу

Компонент	Мінімальні вимоги	Рекомендовані вимоги
ОС	Windows 7 / Ubuntu 18.04 / macOS 10.14	Windows 10/11 або Ubuntu 22.04
Python	Python 3.8	Python 3.10+
Оперативна пам'ять	512 MB	2 GB
Місце на диску	200 MB	500 MB
Браузер	Chrome 80 / Edge 80 / Opera	Актуальна версія Chrome, Edge або Opera

Процедура встановлення та запуску включає наступні кроки:

1) відкрити папку проєкту у VS Code (File → Open Folder);

2) відкрити термінал (Ctrl+~) та створити віртуальне середовище: python -m venv venv;

3) активувати середовище: venv\Scripts\activate (Windows) або source venv/bin/activate (Linux/macOS);

4) встановити залежності: pip install -r requirements.txt;

5) виконати міграції бази даних: python manage.py migrate;

6) запустити сервер розробки: python manage.py runserver;

7) відкрити браузер за адресою: http://127.0.0.1:8000;

4.1.2 Опис основних екранів системи

Вебсервіс складається з 8 функціональних розділів. На рисунку 4.1 наведено схему навігації між екранами системи.

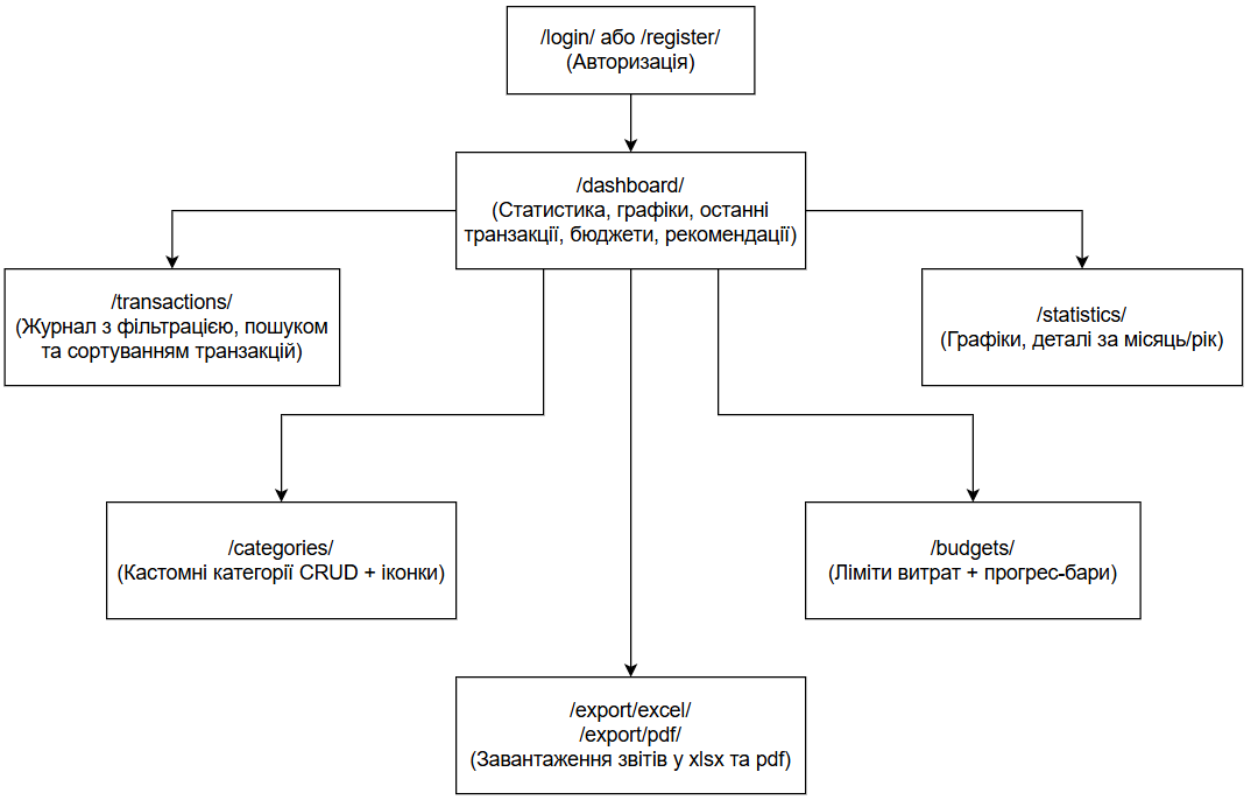


Рисунок 4.1 – Схема навігації між екранами системи

4.1.3 Робота з дашбордом

Після авторизації користувач потрапляє на головну сторінку – дашборд. Він відображає зведені показники поточного місяця: загальні доходи, загальні витрати, місячний баланс та загальний баланс за весь час роботи в системі. Нижче розміщено стовпчасту діаграму доходів і витрат за останні 6 місяців та кругову діаграму розподілу витрат за категоріями. На основі співвідношення доходів і витрат система генерує кольорові аналітичні рекомендації (зелені – норма, жовті – попередження, червоні – перевищення).

4.1.4 Робота з журналом транзакцій

Розділ «Транзакції» (/transactions/) відображає повний журнал фінансових операцій з пагінацією (15 записів на сторінку). Над таблицею розміщено форму фільтрації з полями: тип операції, категорія, дата «від» та «до», текстовий пошук за описом. Усі фільтри можна комбінувати між собою. Кнопки сортування дозволяють упорядкувати список за датою або сумою (зростаючий та спадаючий порядок). Підсумкові рядки над таблицею відображають загальні суми доходів та витрат для відфільтрованої вибірки.

4.1.5 Управління категоріями

Розділ «Категорії» (/categories/) відображає всі категорії користувача, згруповані за типом: доходи та витрати. При реєстрації автоматично створюються 14 стандартних категорій (5 для доходів, 9 для витрат) із визначеними іконками та кольорами. Користувач може редагувати будь-яку категорію або створити нову, вказавши назву, тип, emoji-іконку та колір (через color-picker). Видалення категорії не видаляє пов'язані транзакції – вони зберігаються без категорії.

4.1.6 Бюджетування

Розділ «Бюджети» (/budgets/) дозволяє встановити щомісячний ліміт витрат для кожної категорії. Для кожного бюджету відображається: картка з назвою та іконкою категорії; прогрес-бар, що показує відсоток використання (зелений – до 70 %, жовтий – 70–90 %, червоний – понад 90 %); числові показники «витрачено / ліміт». Бюджети прив'язані до конкретного місяця та року, що дозволяє вести окремий облік для кожного облікового період.

4.2 Тестування системи

4.2.1 Методологія тестування

Тестування системи проводилось на трьох рівнях: функціональне тестування (перевірка відповідності вимогам ФВ-01 – ФВ-15), тестування безпеки (перевірка

захисту від вразливостей) та тестування інтерфейсу (перевірка коректності відображення).

4.2.2 Функціональне тестування

В таблиці 4.2 наведено результати функціонального тестування вебсервісу.

Таблиця 4.2 – Результати функціонального тестування

ID тесту	Тестовий сценарій	Очікуваний результат	Фактичний результат	Статус
Т-01	Реєстрація з коректними даними	Редирект на /dashboard/, 14 категорій створено автоматично	Відповідає	Пройдено
Т-02	Реєстрація із зайнятим логіном	Відображення помилки валідації	Відповідає	Пройдено
Т-03	Авторизація з некоректним паролем	Повідомлення про помилку, форма залишається	Відповідає	Пройдено
Т-04	Додавання транзакції типу «дохід»	Транзакція збережена, статистика оновлена	Відповідає	Пройдено
Т-05	Додавання транзакції із сумою 0	Помилка валідації: сума повинна бути > 0	Відповідає	Пройдено
Т-06	Фільтрація за типом «витрата»	Відображаються лише витрати	Відповідає	Пройдено
Т-07	Пошук за словом у описі	Знаходяться транзакції з відповідним описом	Відповідає	Пройдено
Т-08	Видалення транзакції	Транзакція видалена, статистика оновлена	Відповідає	Пройдено
Т-09	Встановлення бюджету та витрата 90 %	Прогрес-бар жовтий (70–90 %)	Відповідає	Пройдено

Кінець таблиці 4.2

ID тесту	Тестовий сценарій	Очікуваний результат	Фактичний результат	Статус
Т-10	Встановлення бюджету та витрата 100 %	Прогрес-бар червоний (> 90 %)	Відповідає	Пройдено
Т-11	Перегляд статистики за різні місяці	Графіки та суми відповідають обраному місяцю	Відповідає	Пройдено
Т-12	Завантаження Excel-файлу	Файл .xlsx завантажується, дані коректні	Відповідає	Пройдено
Т-13	Завантаження PDF-звіту	Файл .pdf завантажується, кирилиця відображається	Відповідає	Пройдено
Т-14	Перегляд дашборду без транзакцій	Порожні графіки, підказка про додавання транзакцій	Відповідає	Пройдено
Т-15	Пагінація: перехід між сторінками	Правильно відображаються відповідні записи	Відповідає	Пройдено

4.2.3 Тестування безпеки

В таблиці 4.3 наведено результати тестування безпеки вебсервісу.

Таблиця 4.3 – Результати тестування безпеки

Тест безпеки	Метод перевірки	Результат
CSRF-захист	POST-запит без CSRF-токену через cURL	HTTP 403 Forbidden. Захист активний

Кінець таблиці 4.3

Тест безпеки	Метод перевірки	Результат
Несанкціонований доступ	Прямий GET-запит до /dashboard/ без авторизації	Редирект на /login/. Захист активний
Ізоляція даних між користувачами	Спроба GET /transactions/123/edit/ для чужої транзакції	HTTP 404 Not Found. Ізоляція активна
SQL-ін'єкція	Введення SQL-команд у поле пошуку	ORM екранує параметри. Захист активний
Зберігання паролів	Перевірка таблиці auth_user у БД	Паролі зберігаються у хешованому вигляді (PBKDF2-SHA256)
Мінімальна довжина пароля	Спроба реєстрації з паролем < 8 символів	Помилка валідації. Захист активний

4.2.4 Тестування інтерфейсу

В таблиці 4.4 наведено результати тестування інтерфейсу вебсервісу.

Таблиця 4.4 – Результати тестування інтерфейсу

Браузер / пристрій	Розмір екрану	Результат
Google Chrome	1920×1080 px	Коректне відображення всіх елементів, графіки працюють
Edge	1440×900 px	Коректне відображення, незначні відмінності у кольорах
Opera	1280×720 px	Коректне відображення всіх елементів
Мобільний (Chrome)	390×844 px (iPhone 14)	Адаптивний дизайн: бічна панель прихована, контент у 1 колонку
Планшет (Chrome)	768×1024 px (iPad)	Грід у 2 колонки, навігація коректна

4.2.5 Оцінка продуктивності

Тестування продуктивності проводилось на тестовій базі даних зі 100 транзакціями для одного акаунту. Вимірювання часу відгуку здійснювалось за допомогою вбудованих інструментів браузера (DevTools → Network). Результати тестування продуктивності вебсервісу (отриманий час відгуку системи при 100 транзакціях) наведено в таблиці 4.5.

Таблиця 4.5 – Час відгуку системи при 100 транзакціях

Сторінка	Середній час відгуку (мс)	Кількість запитів до БД	Відповідність вимогам
/dashboard/	150	8	Так
/transactions/ (без фільтру)	180	3	Так
/transactions/ (з фільтром)	160	4	Так
/statistics/	280	12	Так
/export/excel/	700	2	Так
/export/pdf/	850	2	Так

Усі сторінки завантажуються менш ніж за 2 секунди, що відповідає нефункціональній вимозі щодо продуктивності. Генерація файлів (Excel та PDF) займає дещо більше часу через операції введення/виведення, але залишається в прийнятних межах.

Висновки до розділу 4

У четвертому розділі описано процедуру встановлення та запуску системи (табл. 4.1), детальне керівництво по всіх розділах інтерфейсу та схему навігації (рис. 4.1). Проведено комплексне тестування: функціональне (15 сценаріїв, табл. 4.2 – всі пройдено), тестування безпеки (6 перевірок, табл. 4.3 – всі захисти

активні), тестування інтерфейсу у 3 браузерах та на пристроях різних розмірів (табл. 4.4). Виміряно продуктивність системи (табл. 4.5): усі сторінки відповідають вимозі часу відгуку < 2 секунди. Всі 15 функціональних та 6 нефункціональних вимог, визначених у розділі 1, виконано повністю.

ВИСНОВКИ

У результаті виконання кваліфікаційної роботи розроблено вебсервіс для керування особистими фінансами, що вирішує задачі автоматизованого обліку, аналізу та планування особистих фінансів. Усі задачі, поставлені у вступі, виконані в повному обсязі.

У першому розділі проведено аналіз предметної сфери та встановлено, що рівень охоплення населення України цифровими інструментами фінансового менеджменту становить лише 15 %. Огляд трьох аналогів (Monobank, Monefy, Spendee) виявив їх ключові обмеження. Сформульовано 15 функціональних вимог, згрупованих за пріоритетами, та 6 нефункціональних вимог.

У другому розділі обґрунтовано вибір технологічного стеку: Python 3.10+ та Django 5.0 як оптимальне поєднання для веброзробки на основі порівняльного аналізу мов (табл. 2.2) та СУБД (табл. 2.4). Описано методи роботи з даними через агрегаційні ORM-запити та алгоритм генерації аналітичних рекомендацій на основі розрахунку норми заощаджень.

У третьому розділі виконано повну програмну реалізацію: визначено файлову структуру проєкту (табл. 3.1). Розроблено систему маршрутизації URL для кожного функціонального розділу (табл. 3.2). Створено спрощену діаграму варіантів використання системи користувачем (рис. 3.1). Розроблено блок-схему загального алгоритму роботи системи (рис. 3.2). Реалізовано бізнес-логіку (Views та Forms) та модуль експорту даних (Excel та PDF). Розроблено шаблони інтерфейсу вебсервісу.

У четвертому розділі проведено комплексне тестування системи за 15 функціональними сценаріями, 6 перевірками безпеки та тестами інтерфейсу в 4 браузерах. Виміряно продуктивність: максимальний час відгуку для сторінок становить 280 мс, що значно нижче встановленої вимоги у 2000 мс. Усі вимоги (15 функціональних та 6 нефункціональних) виконані в повному обсязі.

Практичне значення результатів роботи полягає у тому, що розроблений веб-сервіс може бути безпосередньо використаний широким колом користувачів для управління особистими або сімейними фінансами. Система є готовою базою для подальшого розвитку.

Перспективи розвитку системи:

– інтеграція з банківськими API (Monobank API, PrivatBank API) для автоматичного імпорту транзакцій;

– розробка REST API та мобільного застосунку (React Native або Flutter);

– реалізація функцій фінансового прогнозування на основі методів машинного навчання;

– підтримка мультивалютності та автоматичне оновлення курсів;

– хмарне розгортання (Docker + Heroku або AWS) для мультикористувацького режиму.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Національний банк України. Звіт про фінансову грамотність населення України. Київ, 2021. 48 с.

2. Клименко І. В. Основи фінансового менеджменту: навч. посіб. Київ: КНУ ім. Тараса Шевченка, 2019. 280 с.

3. Денисенко М. П. Персональні фінанси: управління та планування. Харків: Видавництво ХНУ, 2020. 312 с.

4. S&P Global FinLit Survey: Measuring Financial Literacy Around the World. URL: https://gflec.org/initiatives/sp-global-finlit-survey/

5. Personal Finance Software Market Report 2024. Grand View Research. URL: https://www.grandviewresearch.com/industry-analysis/personal-finance-software-market

6. Percival H., Gregory B. Architecture Patterns with Python. O'Reilly Media, 2020. 586 p.

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

8. OWASP Top Ten. Open Web Application Security Project. URL: https://owasp.org/www-project-top-ten/

9. Monobank – мобільний банк. Офіційний сайт. URL: https://www.monobank.ua/

10. Monefy – Expense Manager App. Official website. URL: https://www.monefy.me/

11. Monefy interface. URL: htts://radar.ph/budgeting-apps-that-mean-business-for-the-no-nonsense-spender/

12. Spendee – Budget & Money Tracker. Official website. URL: https://www.spendee.com/

13. Spendee interface. URL: https://www.pcmag.com/news/spendee-offers-easy-mobile-financial-tracking-on-iphone

14. Visual Studio Code documentation. URL: https://code.visualstudio.com/docs

15. Python documentation. Version 3.10. URL: https://docs.python.org/3.10/

16. Lutz M. Learning Python. 5th ed. O'Reilly Media, 2013. 1540 p.

17. Vincent W. S. Django for Beginners: Build websites with Python & Django. 4th ed. 2023. 332 p.

18. Holovaty A., Kaplan-Moss J. The Definitive Guide to Django: Web Development Done Right. 2nd ed. Apress, 2009. 536 p.

19. SQLite Documentation. URL: https://www.sqlite.org/docs.html

20. ReportLab User Guide. Version 4.2. URL: https://docs.reportlab.com/reportlab/userguide/

21. OpenPyXL Documentation. Version 3.1. URL: https://openpyxl.readthedocs.io/en/stable/

22. Chart.js Documentation. Version 4.4. URL: https://www.chartjs.org/docs/latest/

23. Django security overview. URL: https://docs.djangoproject.com/en/5.0/topics/security/

24. Django documentation. Version 5.0. URL: https://docs.djangoproject.com/en/5.0/

25. Gamma E., Helm R., Johnson R., Vlissides J. Design Patterns: Elements of Reusable Object-Oriented Software. Addison-Wesley, 1994. 395 p.

26. Beazley D., Jones B. K. Python Cookbook. 3rd ed. O'Reilly Media, 2013. 706 p.

27. Mozilla MDN Web Docs. HTML, CSS, JavaScript Reference. URL: https://developer.mozilla.org/

28. Real Python. Web Development with Django. URL: https://realpython.com/tutorials/django/

29. Codd E. F. A Relational Model of Data for Large Shared Data Banks. Communications of the ACM. 1970. Vol. 13, No. 6. P. 377–387.

30. Nielsen J. Usability Engineering. Academic Press, 1993. 362 p.

31. Fielding R. T. Architectural Styles and the Design of Network-based Software Architectures: PhD thesis. University of California, Irvine, 2000. 162 p.

32. Git Documentation. Reference Manual. URL: https://git-scm.com/doc

ДОДАТОК А

Текст програми

Файл admin.py

```
from django.contrib import admin
from .models import Transaction, Category, Budget


@admin.register(Category)
class CategoryAdmin(admin.ModelAdmin):
    list_display = ['name', 'type', 'icon', 'user', 'created_at']
    list_filter = ['type', 'user']
    search_fields = ['name']


@admin.register(Transaction)
class TransactionAdmin(admin.ModelAdmin):
    list_display = ['user', 'type', 'amount', 'category', 'description', 'date']
    list_filter = ['type', 'user', 'date']
    search_fields = ['description']
    date_hierarchy = 'date'


@admin.register(Budget)
class BudgetAdmin(admin.ModelAdmin):
    list_display = ['user', 'category', 'limit', 'month', 'year']
    list_filter = ['user', 'year', 'month']
```

Файл forms.py

```
from django import forms
```

```
from django.contrib.auth.forms import UserCreationForm, AuthenticationForm
from django.contrib.auth.models import User
from .models import Transaction, Category, Budget


class RegisterForm(UserCreationForm):
    email = forms.EmailField(required=True, label='Email')
    first_name = forms.CharField(max_length=50, required=True, label="Ім'я")
    last_name = forms.CharField(max_length=50, required=True, label='Прізвище')

    class Meta:
        model = User
        fields = ['username', 'first_name', 'last_name', 'email', 'password1', 'password2']

    def __init__(self, *args, **kwargs):
        super().__init__(*args, **kwargs)
        placeholders = {
            'username': 'Логін',
            'first_name': "Ім'я",
            'last_name': 'Прізвище',
            'email': 'Email',
            'password1': 'Пароль',
            'password2': 'Підтвердження паролю',
        }
        for field_name, placeholder in placeholders.items():
            self.fields[field_name].widget.attrs.update({
                'class': 'form-input',
                'placeholder': placeholder,
            })
```

```
        self.fields['username'].label = 'Логін'
        self.fields['password1'].label = 'Пароль'
        self.fields['password2'].label = 'Підтвердження паролю'


class LoginForm(AuthenticationForm):
    def __init__(self, *args, **kwargs):
        super().__init__(*args, **kwargs)
        self.fields['username'].widget.attrs.update({'class': 'form-input', 'placeholder':
'Логін'})
        self.fields['password'].widget.attrs.update({'class': 'form-input', 'placeholder':
'Пароль'})
        self.fields['username'].label = 'Логін'
        self.fields['password'].label = 'Пароль'


class TransactionForm(forms.ModelForm):
    class Meta:
        model = Transaction
        fields = ['type', 'amount', 'category', 'description', 'date']
        widgets = {
            'date': forms.DateInput(attrs={'type': 'date', 'class': 'form-input'}),
            'amount': forms.NumberInput(attrs={'class': 'form-input', 'placeholder': '0.00',
'step': '0.01', 'min': '0.01'}),
            'description': forms.TextInput(attrs={'class': 'form-input', 'placeholder': 'Опис
(необов\'язково)'}),
            'type': forms.Select(attrs={'class': 'form-input', 'id': 'id_type'}),
            'category': forms.Select(attrs={'class': 'form-input', 'id': 'id_category'}),
        }
        labels = {
```

```
            'type': 'Тип',
            'amount': 'Сума (грн)',
            'category': 'Категорія',
            'description': 'Опис',
            'date': 'Дата',
        }

    def __init__(self, user=None, *args, **kwargs):
        super().__init__(*args, **kwargs)
        if user:
            self.fields['category'].queryset = Category.objects.filter(user=user)
        self.fields['category'].required = False
        self.fields['category'].empty_label = '— Без категорії —'


class CategoryForm(forms.ModelForm):
    class Meta:
        model = Category
        fields = ['name', 'type', 'icon', 'color']
        widgets = {
            'name': forms.TextInput(attrs={'class': 'form-input', 'placeholder': 'Назва
категорії'}),
            'type': forms.Select(attrs={'class': 'form-input'}),
            'icon': forms.TextInput(attrs={'class': 'form-input', 'placeholder': '💰'}),
            'color': forms.TextInput(attrs={'class': 'form-input', 'type': 'color'}),
        }
        labels = {
            'name': 'Назва',
            'type': 'Тип',
```

```
            'icon': 'Іконка (emoji)',
            'color': 'Колір',
        }


class BudgetForm(forms.ModelForm):
    class Meta:
        model = Budget
        fields = ['category', 'limit', 'month', 'year']
        widgets = {
            'category': forms.Select(attrs={'class': 'form-input'}),
            'limit': forms.NumberInput(attrs={'class': 'form-input', 'placeholder': '0.00', 'step':
'0.01'}),
            'month': forms.NumberInput(attrs={'class': 'form-input', 'min': 1, 'max': 12}),
            'year': forms.NumberInput(attrs={'class': 'form-input', 'min': 2020, 'max': 2100}),
        }
        labels = {
            'category': 'Категорія',
            'limit': 'Ліміт (грн)',
            'month': 'Місяць',
            'year': 'Рік',
        }

    def __init__(self, user=None, *args, **kwargs):
        super().__init__(*args, **kwargs)
        if user:
            self.fields['category'].queryset = Category.objects.filter(user=user,
type='expense')
```

```
class TransactionFilterForm(forms.Form):
    TYPE_CHOICES = [('', 'Усі'), ('income', 'Доходи'), ('expense', 'Витрати')]

    type = forms.ChoiceField(choices=TYPE_CHOICES, required=False, label='Тип',
                             widget=forms.Select(attrs={'class': 'form-input'}))
    category = forms.ModelChoiceField(queryset=Category.objects.none(),
required=False,
                                      label='Категорія', empty_label='Усі категорії',
                                      widget=forms.Select(attrs={'class': 'form-input'}))
    date_from = forms.DateField(required=False, label='Від',
                                widget=forms.DateInput(attrs={'type': 'date', 'class': 'form-input'}))
    date_to = forms.DateField(required=False, label='До',
                              widget=forms.DateInput(attrs={'type': 'date', 'class': 'form-input'}))
    search = forms.CharField(required=False, label='Пошук',
                             widget=forms.TextInput(attrs={'class': 'form-input', 'placeholder':
'Пошук за описом...'}))

    def __init__(self, user=None, *args, **kwargs):
        super().__init__(*args, **kwargs)
        if user:
            self.fields['category'].queryset = Category.objects.filter(user=user)
```

Файл models.py

```
from django.db import models
from django.contrib.auth.models import User
from django.utils import timezone


class Category(models.Model):
```

```
    TRANSACTION_TYPES = [
        ('income', 'Дохід'),
        ('expense', 'Витрата'),
    ]

    user = models.ForeignKey(User, on_delete=models.CASCADE,
related_name='categories')
    name = models.CharField(max_length=100, verbose_name='Назва')
    type = models.CharField(max_length=10, choices=TRANSACTION_TYPES,
verbose_name='Тип')
    icon = models.CharField(max_length=50, default='💰', verbose_name='Іконка')
    color = models.CharField(max_length=7, default='#3B82F6', verbose_name='Колір')
    created_at = models.DateTimeField(auto_now_add=True)

    class Meta:
        verbose_name = 'Категорія'
        verbose_name_plural = 'Категорії'
        ordering = ['name']
        unique_together = ['user', 'name', 'type']

    def __str__(self):
        return f"{self.icon} {self.name}"


class Transaction(models.Model):
    TRANSACTION_TYPES = [
        ('income', 'Дохід'),
        ('expense', 'Витрата'),
    ]
```

```
    user = models.ForeignKey(User, on_delete=models.CASCADE,
related_name='transactions')
    category = models.ForeignKey(Category, on_delete=models.SET_NULL, null=True,
blank=True, related_name='transactions')
    type = models.CharField(max_length=10, choices=TRANSACTION_TYPES,
verbose_name='Тип')
    amount = models.DecimalField(max_digits=12, decimal_places=2,
verbose_name='Сума')
    description = models.CharField(max_length=255, verbose_name='Опис',
blank=True)
    date = models.DateField(default=timezone.now, verbose_name='Дата')
    created_at = models.DateTimeField(auto_now_add=True)
    updated_at = models.DateTimeField(auto_now=True)

    class Meta:
        verbose_name = 'Транзакція'
        verbose_name_plural = 'Транзакції'
        ordering = ['-date', '-created_at']

    def __str__(self):
        sign = '+' if self.type == 'income' else '-'
        return f"{sign}{self.amount} грн — {self.description or self.category}"


class Budget(models.Model):
    user = models.ForeignKey(User, on_delete=models.CASCADE,
related_name='budgets')
```

```
    category = models.ForeignKey(Category, on_delete=models.CASCADE,
related_name='budgets')
    limit = models.DecimalField(max_digits=12, decimal_places=2,
verbose_name='Ліміт')
    month = models.IntegerField(verbose_name='Місяць')
    year = models.IntegerField(verbose_name='Рік')

    class Meta:
        verbose_name = 'Бюджет'
        verbose_name_plural = 'Бюджети'
        unique_together = ['user', 'category', 'month', 'year']

    def __str__(self):
        return f"{self.category.name} — {self.limit} грн ({self.month}/{self.year})"

    def spent(self):
        from django.db.models import Sum
        total = Transaction.objects.filter(
            user=self.user,
            category=self.category,
            type='expense',
            date__month=self.month,
            date__year=self.year
        ).aggregate(Sum('amount'))['amount__sum'] or 0
        return total

    def percentage(self):
        if self.limit == 0:
            return 0
```

```
        return min(int((self.spent() / self.limit) * 100), 100)
```

Файл urls.py

```
from django.urls import path
from . import views

urlpatterns = [
    # Auth
    path('', views.dashboard, name='home'),
    path('register/', views.register_view, name='register'),
    path('login/', views.login_view, name='login'),
    path('logout/', views.logout_view, name='logout'),

    # Dashboard
    path('dashboard/', views.dashboard, name='dashboard'),

    # Transactions
    path('transactions/', views.transaction_list, name='transaction_list'),
    path('transactions/add/', views.transaction_add, name='transaction_add'),
    path('transactions/<int:pk>/edit/', views.transaction_edit, name='transaction_edit'),
    path('transactions/<int:pk>/delete/', views.transaction_delete,
name='transaction_delete'),

    # Categories
    path('categories/', views.category_list, name='category_list'),
    path('categories/add/', views.category_add, name='category_add'),
    path('categories/<int:pk>/edit/', views.category_edit, name='category_edit'),
    path('categories/<int:pk>/delete/', views.category_delete, name='category_delete'),
```

```
    # Statistics
    path('statistics/', views.statistics, name='statistics'),

    # Budgets
    path('budgets/', views.budget_list, name='budget_list'),
    path('budgets/add/', views.budget_add, name='budget_add'),
    path('budgets/<int:pk>/delete/', views.budget_delete, name='budget_delete'),

    # Export
    path('export/excel/', views.export_excel, name='export_excel'),
    path('export/pdf/', views.export_pdf, name='export_pdf'),

    # API
    path('api/categories/', views.api_categories, name='api_categories'),
]
```

Файл views.py

```
from django.shortcuts import render, redirect, get_object_or_404
from django.contrib.auth import login, logout, authenticate
from django.contrib.auth.decorators import login_required
from django.contrib import messages
from django.db.models import Sum, Count, Q
from django.http import HttpResponse, JsonResponse
from django.utils import timezone
from django.core.paginator import Paginator
import json
import io
import datetime
```

```
from .models import Transaction, Category, Budget
from .forms import (RegisterForm, LoginForm, TransactionForm,
                    CategoryForm, BudgetForm, TransactionFilterForm)


def register_view(request):
    if request.user.is_authenticated:
        return redirect('dashboard')
    if request.method == 'POST':
        form = RegisterForm(request.POST)
        if form.is_valid():
            user = form.save()
            _create_default_categories(user)
            login(request, user)
            messages.success(request, f'Вітаємо, {user.first_name}! Акаунт успішно створено.')
            return redirect('dashboard')
    else:
        form = RegisterForm()
    return render(request, 'finance_app/auth/register.html', {'form': form})


def login_view(request):
    if request.user.is_authenticated:
        return redirect('dashboard')
    if request.method == 'POST':
        form = LoginForm(request, data=request.POST)
        if form.is_valid():
            user = form.get_user()
            login(request, user)
```

```
            messages.success(request, f'Ласкаво просимо, {user.first_name or
user.username}!')
            return redirect('dashboard')
    else:
        form = LoginForm()
    return render(request, 'finance_app/auth/login.html', {'form': form})


def logout_view(request):
    logout(request)
    return redirect('login')

@login_required
def dashboard(request):
    user = request.user
    now = timezone.now()
    month, year = now.month, now.year

    month_transactions = Transaction.objects.filter(
        user=user, date__month=month, date__year=year
    )
    total_income =
month_transactions.filter(type='income').aggregate(Sum('amount'))['amount__sum'] or
0
    total_expense =
month_transactions.filter(type='expense').aggregate(Sum('amount'))['amount__sum'] or
0
    balance = total_income - total_expense
```

```
    all_income = Transaction.objects.filter(user=user,
type='income').aggregate(Sum('amount'))['amount__sum'] or 0
    all_expense = Transaction.objects.filter(user=user,
type='expense').aggregate(Sum('amount'))['amount__sum'] or 0
    all_balance = all_income - all_expense

    recent = Transaction.objects.filter(user=user).select_related('category')[:8]

    expense_by_category = month_transactions.filter(type='expense').values(
        'category__name', 'category__color', 'category__icon'
    ).annotate(total=Sum('amount')).order_by('-total')[:6]

    chart_data = _get_monthly_chart_data(user, 6)

    budgets = Budget.objects.filter(user=user, month=month,
year=year).select_related('category')

    tips = _generate_tips(user, total_income, total_expense, month, year)

    context = {
        'total_income': total_income,
        'total_expense': total_expense,
        'balance': balance,
        'all_balance': all_balance,
        'recent': recent,
        'expense_by_category': expense_by_category,
        'chart_data': json.dumps(chart_data),
        'budgets': budgets,
        'tips': tips,
```

```
        'current_month': now.strftime('%B %Y'),
    }
    return render(request, 'finance_app/dashboard.html', context)


@login_required
def transaction_list(request):
    user = request.user
    filter_form = TransactionFilterForm(user=user, data=request.GET or None)
    transactions = Transaction.objects.filter(user=user).select_related('category')

    if filter_form.is_valid():
        data = filter_form.cleaned_data
        if data.get('type'):
            transactions = transactions.filter(type=data['type'])
        if data.get('category'):
            transactions = transactions.filter(category=data['category'])
        if data.get('date_from'):
            transactions = transactions.filter(date__gte=data['date_from'])
        if data.get('date_to'):
            transactions = transactions.filter(date__lte=data['date_to'])
        if data.get('search'):
            transactions = transactions.filter(description__icontains=data['search'])

    sort = request.GET.get('sort', '-date')
    if sort in ['date', '-date', 'amount', '-amount']:
        transactions = transactions.order_by(sort)

    total_income =
transactions.filter(type='income').aggregate(Sum('amount'))['amount__sum'] or 0
```

```
    total_expense =
transactions.filter(type='expense').aggregate(Sum('amount'))['amount__sum'] or 0

    paginator = Paginator(transactions, 15)
    page_obj = paginator.get_page(request.GET.get('page'))

    return render(request, 'finance_app/transactions/list.html', {
        'page_obj': page_obj,
        'filter_form': filter_form,
        'total_income': total_income,
        'total_expense': total_expense,
        'sort': sort,
    })


@login_required
def transaction_add(request):
    if request.method == 'POST':
        form = TransactionForm(user=request.user, data=request.POST)
        if form.is_valid():
            t = form.save(commit=False)
            t.user = request.user
            t.save()
            messages.success(request, 'Транзакцію додано успішно!')
            return redirect('transaction_list')
    else:
        form = TransactionForm(user=request.user)
    return render(request, 'finance_app/transactions/form.html', {'form': form, 'title':
'Нова транзакція'})
```

```
@login_required
def transaction_edit(request, pk):
    transaction = get_object_or_404(Transaction, pk=pk, user=request.user)
    if request.method == 'POST':
        form = TransactionForm(user=request.user, data=request.POST,
instance=transaction)
        if form.is_valid():
            form.save()
            messages.success(request, 'Транзакцію оновлено!')
            return redirect('transaction_list')
    else:
        form = TransactionForm(user=request.user, instance=transaction)
    return render(request, 'finance_app/transactions/form.html', {'form': form, 'title':
'Редагування транзакції'})


@login_required
def transaction_delete(request, pk):
    transaction = get_object_or_404(Transaction, pk=pk, user=request.user)
    if request.method == 'POST':
        transaction.delete()
        messages.success(request, 'Транзакцію видалено.')
        return redirect('transaction_list')
    return render(request, 'finance_app/transactions/confirm_delete.html', {'object':
transaction})

@login_required
def category_list(request):
```

```
    categories = Category.objects.filter(user=request.user).annotate(
        transaction_count=Count('transactions')
    )
    return render(request, 'finance_app/categories/list.html', {'categories': categories})


@login_required
def category_add(request):
    if request.method == 'POST':
        form = CategoryForm(request.POST)
        if form.is_valid():
            cat = form.save(commit=False)
            cat.user = request.user
            cat.save()
            messages.success(request, 'Категорію додано!')
            return redirect('category_list')
    else:
        form = CategoryForm()
    return render(request, 'finance_app/categories/form.html', {'form': form, 'title': 'Нова категорія'})


@login_required
def category_edit(request, pk):
    category = get_object_or_404(Category, pk=pk, user=request.user)
    if request.method == 'POST':
        form = CategoryForm(request.POST, instance=category)
        if form.is_valid():
            form.save()
            messages.success(request, 'Категорію оновлено!')
```

```
            return redirect('category_list')
    else:
        form = CategoryForm(instance=category)
    return render(request, 'finance_app/categories/form.html', {'form': form, 'title':
'Редагування категорії'})


@login_required
def category_delete(request, pk):
    category = get_object_or_404(Category, pk=pk, user=request.user)
    if request.method == 'POST':
        category.delete()
        messages.success(request, 'Категорію видалено.')
        return redirect('category_list')
    return render(request, 'finance_app/categories/confirm_delete.html', {'object':
category})

@login_required
def statistics(request):
    user = request.user
    now = timezone.now()
    year = int(request.GET.get('year', now.year))
    month = int(request.GET.get('month', now.month))

    month_qs = Transaction.objects.filter(user=user, date__month=month,
date__year=year)
    income = month_qs.filter(type='income').aggregate(Sum('amount'))['amount__sum']
or 0
```

```
    expense = month_qs.filter(type='expense').aggregate(Sum('amount'))['amount__sum']
or 0

    by_cat = month_qs.filter(type='expense').values(
        'category__name', 'category__color', 'category__icon'
    ).annotate(total=Sum('amount')).order_by('-total')

    by_cat_income = month_qs.filter(type='income').values(
        'category__name', 'category__color', 'category__icon'
    ).annotate(total=Sum('amount')).order_by('-total')

    year_data = _get_monthly_chart_data(user, 12, year=year)

    daily = {}
    for t in month_qs:
        day = t.date.day
        if day not in daily:
            daily[day] = {'income': 0, 'expense': 0}
        daily[day][t.type] += float(t.amount)

    daily_labels = sorted(daily.keys())
    daily_income = [daily[d]['income'] for d in daily_labels]
    daily_expense = [daily[d]['expense'] for d in daily_labels]

    months_list = [
        (1, 'Січень'), (2, 'Лютий'), (3, 'Березень'), (4, 'Квітень'),
        (5, 'Травень'), (6, 'Червень'), (7, 'Липень'), (8, 'Серпень'),
        (9, 'Вересень'), (10, 'Жовтень'), (11, 'Листопад'), (12, 'Грудень'),
    ]
```

```
    years_list = list(range(2020, now.year + 2))

    context = {
        'income': income,
        'expense': expense,
        'balance': income - expense,
        'by_cat': by_cat,
        'by_cat_income': by_cat_income,
        'year_data': json.dumps(year_data),
        'daily_labels': json.dumps(daily_labels),
        'daily_income': json.dumps(daily_income),
        'daily_expense': json.dumps(daily_expense),
        'selected_month': month,
        'selected_year': year,
        'months_list': months_list,
        'years_list': years_list,
    }
    return render(request, 'finance_app/statistics.html', context)

@login_required
def budget_list(request):
    now = timezone.now()
    month = int(request.GET.get('month', now.month))
    year = int(request.GET.get('year', now.year))
    budgets = Budget.objects.filter(user=request.user, month=month,
year=year).select_related('category')
    return render(request, 'finance_app/budgets/list.html', {
        'budgets': budgets, 'month': month, 'year': year,
    })
```

```
@login_required
def budget_add(request):
    now = timezone.now()
    if request.method == 'POST':
        form = BudgetForm(user=request.user, data=request.POST)
        if form.is_valid():
            b = form.save(commit=False)
            b.user = request.user
            b.save()
            messages.success(request, 'Бюджет встановлено!')
            return redirect('budget_list')
    else:
        form = BudgetForm(user=request.user, initial={'month': now.month, 'year':
now.year})
    return render(request, 'finance_app/budgets/form.html', {'form': form, 'title': 'Новий
бюджет'})


@login_required
def budget_delete(request, pk):
    budget = get_object_or_404(Budget, pk=pk, user=request.user)
    if request.method == 'POST':
        budget.delete()
        messages.success(request, 'Бюджет видалено.')
        return redirect('budget_list')
    return render(request, 'finance_app/budgets/confirm_delete.html', {'object': budget})

@login_required
```

```
def export_excel(request):
    import openpyxl
    from openpyxl.styles import Font, PatternFill, Alignment, Border, Side

    user = request.user
    transactions =
Transaction.objects.filter(user=user).select_related('category').order_by('-date')

    wb = openpyxl.Workbook()
    ws = wb.active
    ws.title = 'Транзакції'

    header_font = Font(bold=True, color='FFFFFF', size=12)
    header_fill = PatternFill(start_color='1E40AF', end_color='1E40AF',
fill_type='solid')
    income_fill = PatternFill(start_color='D1FAE5', end_color='D1FAE5',
fill_type='solid')
    expense_fill = PatternFill(start_color='FEE2E2', end_color='FEE2E2',
fill_type='solid')
    border = Border(
        left=Side(style='thin'), right=Side(style='thin'),
        top=Side(style='thin'), bottom=Side(style='thin')
    )
    center = Alignment(horizontal='center')

    headers = ['№', 'Дата', 'Тип', 'Категорія', 'Опис', 'Сума (грн)']
    for col, header in enumerate(headers, 1):
        cell = ws.cell(row=1, column=col, value=header)
        cell.font = header_font
```

```
        cell.fill = header_fill
        cell.alignment = center
        cell.border = border

    ws.column_dimensions['A'].width = 6
    ws.column_dimensions['B'].width = 14
    ws.column_dimensions['C'].width = 12
    ws.column_dimensions['D'].width = 20
    ws.column_dimensions['E'].width = 30
    ws.column_dimensions['F'].width = 16

    for i, t in enumerate(transactions, 1):
        row = i + 1
        fill = income_fill if t.type == 'income' else expense_fill
        values = [
            i, str(t.date), 'Дохід' if t.type == 'income' else 'Витрата',
            str(t.category) if t.category else '—',
            t.description or '—', float(t.amount)
        ]
        for col, value in enumerate(values, 1):
            cell = ws.cell(row=row, column=col, value=value)
            cell.fill = fill
            cell.border = border
            if col in [1, 2, 3]:
                cell.alignment = center

    ws2 = wb.create_sheet('Зведення')
    total_income =
transactions.filter(type='income').aggregate(Sum('amount'))['amount__sum'] or 0
```

```
    total_expense =
transactions.filter(type='expense').aggregate(Sum('amount'))['amount__sum'] or 0
    summary_data = [
        ('Загальний дохід:', float(total_income)),
        ('Загальні витрати:', float(total_expense)),
        ('Баланс:', float(total_income - total_expense)),
    ]
    for row_num, (label, value) in enumerate(summary_data, 1):
        ws2.cell(row=row_num, column=1, value=label).font = Font(bold=True)
        ws2.cell(row=row_num, column=2, value=value)
    ws2.column_dimensions['A'].width = 25
    ws2.column_dimensions['B'].width = 18

    response = HttpResponse(
        content_type='application/vnd.openxmlformats-
officedocument.spreadsheetml.sheet'
    )
    response['Content-Disposition'] = 'attachment; filename="finances.xlsx"'
    wb.save(response)
    return response


@login_required
def export_pdf(request):
    from reportlab.lib.pagesizes import A4
    from reportlab.lib.styles import getSampleStyleSheet, ParagraphStyle
    from reportlab.lib import colors
    from reportlab.platypus import SimpleDocTemplate, Table, TableStyle, Paragraph,
Spacer
```

```
    from reportlab.lib.units import cm
    from reportlab.pdfbase import pdfmetrics
    from reportlab.pdfbase.ttfonts import TTFont
    import os

    user = request.user
    transactions =
Transaction.objects.filter(user=user).select_related('category').order_by('-date')[:100]

    font_paths = [
        'C:/Windows/Fonts/arial.ttf',
        'C:/Windows/Fonts/Arial.ttf',
        '/usr/share/fonts/truetype/dejavu/DejaVuSans.ttf',
        '/usr/share/fonts/dejavu/DejaVuSans.ttf',
        '/Library/Fonts/Arial.ttf',
        '/System/Library/Fonts/Supplemental/Arial.ttf',
    ]

    font_registered = False
    for path in font_paths:
        if os.path.exists(path):
            try:
                pdfmetrics.registerFont(TTFont('CyrFont', path))
                pdfmetrics.registerFont(TTFont('CyrFont-Bold', path.replace('arial',
'arialbd').replace('Arial.ttf', 'Arialbd.ttf').replace('DejaVuSans.ttf', 'DejaVuSans-
Bold.ttf')))
                font_registered = True
                break
            except Exception:
```

```
            continue

    if not font_registered:
        try:
            from reportlab.pdfbase.ttfonts import TTFont
            import urllib.request
            font_name = 'Helvetica'
            font_bold = 'Helvetica-Bold'
        except Exception:
            font_name = 'Helvetica'
            font_bold = 'Helvetica-Bold'
    else:
        font_name = 'CyrFont'
        font_bold = 'CyrFont-Bold' if font_registered else 'CyrFont'

    buffer = io.BytesIO()
    doc = SimpleDocTemplate(buffer, pagesize=A4, rightMargin=2*cm,
leftMargin=2*cm,
                            topMargin=2*cm, bottomMargin=2*cm)
    elements = []

    title_style = ParagraphStyle('Title', fontName=font_bold, fontSize=18,
                                 textColor=colors.HexColor('#1E40AF'), spaceAfter=6)
    sub_style = ParagraphStyle('Sub', fontName=font_name, fontSize=10,
                               textColor=colors.grey, spaceAfter=12)
    styles = getSampleStyleSheet()

    elements.append(Paragraph('Звіт про фінансові операції', title_style))
    elements.append(Paragraph(
```

```
        f'Користувач: {user.get_full_name() or user.username}  |  Дата:
{timezone.now().strftime("%d.%m.%Y")}',
        sub_style
    ))
    elements.append(Spacer(1, 0.5*cm))

    total_income = Transaction.objects.filter(user=user,
type='income').aggregate(Sum('amount'))['amount__sum'] or 0
    total_expense = Transaction.objects.filter(user=user,
type='expense').aggregate(Sum('amount'))['amount__sum'] or 0
    balance = total_income - total_expense

    summary_data = [
        ['Загальний дохід', f'{total_income:,.2f} грн'],
        ['Загальні витрати', f'{total_expense:,.2f} грн'],
        ['Баланс', f'{balance:,.2f} грн'],
    ]
    summary_table = Table(summary_data, colWidths=[9*cm, 7*cm])
    summary_table.setStyle(TableStyle([
        ('BACKGROUND', (0, 0), (-1, -1), colors.HexColor('#EFF6FF')),
        ('TEXTCOLOR', (0, 0), (0, -1), colors.HexColor('#1E40AF')),
        ('FONTNAME', (0, 0), (-1, -1), font_name),
        ('FONTNAME', (0, 0), (0, -1), font_bold),
        ('FONTSIZE', (0, 0), (-1, -1), 11),
        ('GRID', (0, 0), (-1, -1), 0.5, colors.HexColor('#BFDBFE')),
        ('ROWBACKGROUNDS', (0, 0), (-1, -1), [colors.HexColor('#EFF6FF'),
colors.HexColor('#DBEAFE')]),
        ('PADDING', (0, 0), (-1, -1), 8),
    ]))
```

```
    elements.append(summary_table)
    elements.append(Spacer(1, 0.7*cm))

    heading_style = ParagraphStyle('H2', fontName=font_bold, fontSize=13,
                                   textColor=colors.HexColor('#1E293B'), spaceAfter=8)
    elements.append(Paragraph('Останні транзакції (до 100)', heading_style))
    elements.append(Spacer(1, 0.3*cm))

    data = [['Дата', 'Тип', 'Категорія', 'Опис', 'Сума']]
    for t in transactions:
        cat_name = t.category.name[:20] if t.category else '—'
        data.append([
            str(t.date),
            'Дохід' if t.type == 'income' else 'Витрата',
            cat_name,
            (t.description or '—')[:30],
            f"{'+' if t.type == 'income' else '-'}{t.amount:.2f}"
        ])

    table = Table(data, colWidths=[3*cm, 2.5*cm, 4*cm, 5.5*cm, 3*cm])
    table.setStyle(TableStyle([
        ('BACKGROUND', (0, 0), (-1, 0), colors.HexColor('#1E40AF')),
        ('TEXTCOLOR', (0, 0), (-1, 0), colors.white),
        ('FONTNAME', (0, 0), (-1, 0), font_bold),
        ('FONTNAME', (0, 1), (-1, -1), font_name),
        ('FONTSIZE', (0, 0), (-1, -1), 9),
        ('GRID', (0, 0), (-1, -1), 0.3, colors.HexColor('#E5E7EB')),
        ('ROWBACKGROUNDS', (0, 1), (-1, -1), [colors.white,
colors.HexColor('#F9FAFB')]),
```

```
        ('PADDING', (0, 0), (-1, -1), 5),
        ('ALIGN', (4, 1), (4, -1), 'RIGHT'),
    ]))
    elements.append(table)

    doc.build(elements)
    buffer.seek(0)
    response = HttpResponse(buffer, content_type='application/pdf')
    response['Content-Disposition'] = 'attachment; filename="finances.pdf"'
    return response

@login_required
def api_categories(request):
    type_ = request.GET.get('type', '')
    cats = Category.objects.filter(user=request.user, type=type_).values('id', 'name',
'icon')
    return JsonResponse({'categories': list(cats)})

def _create_default_categories(user):
    defaults = [
        {'name': 'Зарплата', 'type': 'income', 'icon': '💼', 'color': '#10B981'},
        {'name': 'Фріланс', 'type': 'income', 'icon': '💻', 'color': '#059669'},
        {'name': 'Інвестиції', 'type': 'income', 'icon': '📈', 'color': '#047857'},
        {'name': 'Подарунки', 'type': 'income', 'icon': '🎁', 'color': '#065F46'},
        {'name': 'Інше (дохід)', 'type': 'income', 'icon': '✨', 'color': '#6EE7B7'},

        {'name': 'Продукти', 'type': 'expense', 'icon': '🛒', 'color': '#EF4444'},
        {'name': 'Транспорт', 'type': 'expense', 'icon': '🚌', 'color': '#F97316'},
```

```
        {'name': 'Житло', 'type': 'expense', 'icon': '🏠', 'color': '#F59E0B'},
        {'name': 'Здоров\'я', 'type': 'expense', 'icon': '💊', 'color': '#EF4444'},
        {'name': 'Розваги', 'type': 'expense', 'icon': '🎮', 'color': '#8B5CF6'},
        {'name': 'Одяг', 'type': 'expense', 'icon': '👗', 'color': '#EC4899'},
        {'name': 'Освіта', 'type': 'expense', 'icon': '📚', 'color': '#3B82F6'},
        {'name': 'Кафе/Ресторани', 'type': 'expense', 'icon': '🍕', 'color': '#F97316'},
        {'name': 'Інше (витрата)', 'type': 'expense', 'icon': '📦', 'color': '#6B7280'},
    ]
    Category.objects.bulk_create([Category(user=user, **d) for d in defaults])


def _get_monthly_chart_data(user, months_count=6, year=None):
    now = timezone.now()
    labels, income_data, expense_data = [], [], []

    month_names = ['', 'Січ', 'Лют', 'Бер', 'Кві', 'Тра', 'Чер',
                   'Лип', 'Сер', 'Вер', 'Жов', 'Лис', 'Гру']

    if year:
        for m in range(1, 13):
            qs = Transaction.objects.filter(user=user, date__month=m, date__year=year)
            inc = qs.filter(type='income').aggregate(Sum('amount'))['amount__sum'] or 0
            exp = qs.filter(type='expense').aggregate(Sum('amount'))['amount__sum'] or 0
            labels.append(month_names[m])
            income_data.append(float(inc))
            expense_data.append(float(exp))
    else:
        for i in range(months_count - 1, -1, -1):
```

```
            dt = now - datetime.timedelta(days=i * 30)
            m, y = dt.month, dt.year
            qs = Transaction.objects.filter(user=user, date__month=m, date__year=y)
            inc = qs.filter(type='income').aggregate(Sum('amount'))['amount__sum'] or 0
            exp = qs.filter(type='expense').aggregate(Sum('amount'))['amount__sum'] or 0
            labels.append(f"{month_names[m]} {y}")
            income_data.append(float(inc))
            expense_data.append(float(exp))

    return {'labels': labels, 'income': income_data, 'expense': expense_data}


def _generate_tips(user, income, expense, month, year):
    tips = []
    if income == 0 and expense == 0:
        tips.append({'type': 'info', 'text': 'Додайте першу транзакцію, щоб отримати аналітику!'})
        return tips

    if income > 0:
        savings_rate = (income - expense) / income * 100
        if savings_rate < 0:
            tips.append({'type': 'danger', 'text': f'Цього місяця витрати перевищують доходи на {abs(expense - income):.0f} грн. Зверніть увагу на бюджет!'})
        elif savings_rate < 20:
            tips.append({'type': 'warning', 'text': f'Рівень заощаджень {savings_rate:.0f}%. Спробуйте зберігати від 20% доходу.'})
        else:
```

```
            tips.append({'type': 'success', 'text': f'Чудово! Ви заощаджуєте
{savings_rate:.0f}% доходу цього місяця.'})

    top_cat = Transaction.objects.filter(
        user=user, type='expense', date__month=month, date__year=year
    ).values('category__name').annotate(total=Sum('amount')).order_by('-total').first()

    if top_cat and top_cat['category__name']:
        tips.append({'type': 'info', 'text': f'Найбільші витрати: категорія
«{top_cat["category__name"]}» — {top_cat["total"]:.0f} грн.'})

    return tips
```

Файл settings.py

```
from pathlib import Path
import os

BASE_DIR = Path(__file__).resolve().parent.parent

SECRET_KEY = 'django-insecure-change-this-key-in-production-xyz123'

DEBUG = True

ALLOWED_HOSTS = ['*']

INSTALLED_APPS = [
    'django.contrib.admin',
    'django.contrib.auth',
    'django.contrib.contenttypes',
```

```
    'django.contrib.sessions',
    'django.contrib.messages',
    'django.contrib.staticfiles',
    'finance_app',
]

MIDDLEWARE = [
    'django.middleware.security.SecurityMiddleware',
    'django.contrib.sessions.middleware.SessionMiddleware',
    'django.middleware.common.CommonMiddleware',
    'django.middleware.csrf.CsrfViewMiddleware',
    'django.contrib.auth.middleware.AuthenticationMiddleware',
    'django.contrib.messages.middleware.MessageMiddleware',
    'django.middleware.clickjacking.XFrameOptionsMiddleware',
]

ROOT_URLCONF = 'finance_project.urls'

TEMPLATES = [
    {
        'BACKEND': 'django.template.backends.django.DjangoTemplates',
        'DIRS': [BASE_DIR / 'templates'],
        'APP_DIRS': True,
        'OPTIONS': {
            'context_processors': [
                'django.template.context_processors.debug',
                'django.template.context_processors.request',
                'django.contrib.auth.context_processors.auth',
                'django.contrib.messages.context_processors.messages',
```

```
            ],
        },
    },
]

WSGI_APPLICATION = 'finance_project.wsgi.application'

DATABASES = {
    'default': {
        'ENGINE': 'django.db.backends.sqlite3',
        'NAME': BASE_DIR / 'db.sqlite3',
    }
}

AUTH_PASSWORD_VALIDATORS = [
    {'NAME':
'django.contrib.auth.password_validation.UserAttributeSimilarityValidator'},
    {'NAME': 'django.contrib.auth.password_validation.MinimumLengthValidator'},
    {'NAME': 'django.contrib.auth.password_validation.CommonPasswordValidator'},
    {'NAME': 'django.contrib.auth.password_validation.NumericPasswordValidator'},
]

LANGUAGE_CODE = 'uk'
TIME_ZONE = 'Europe/Kyiv'
USE_I18N = True
USE_TZ = True

STATIC_URL = '/static/'
STATICFILES_DIRS = [BASE_DIR / 'static']
```

```
STATIC_ROOT = BASE_DIR / 'staticfiles'

MEDIA_URL = '/media/'
MEDIA_ROOT = BASE_DIR / 'media'

DEFAULT_AUTO_FIELD = 'django.db.models.BigAutoField'

LOGIN_URL = '/login/'
LOGIN_REDIRECT_URL = '/dashboard/'
LOGOUT_REDIRECT_URL = '/login/'
```

ДОДАТОК Б

Інтерфейс вебсервісу

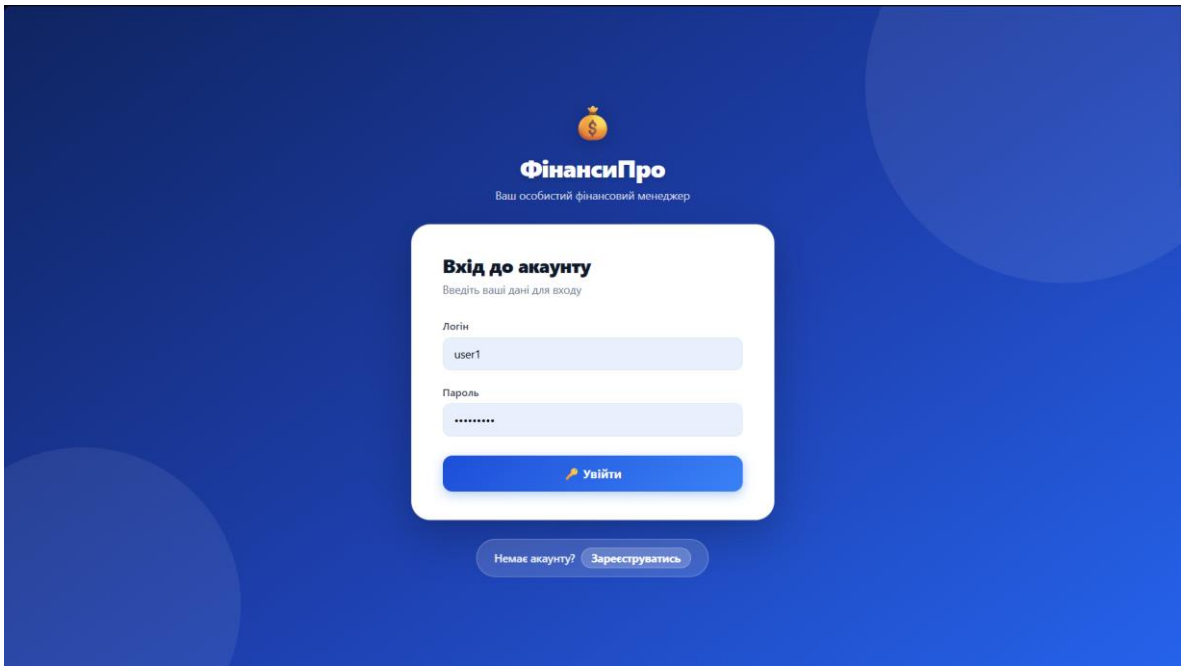


Рисунок Б.1 – Сторінка з формою для входу в акаунт

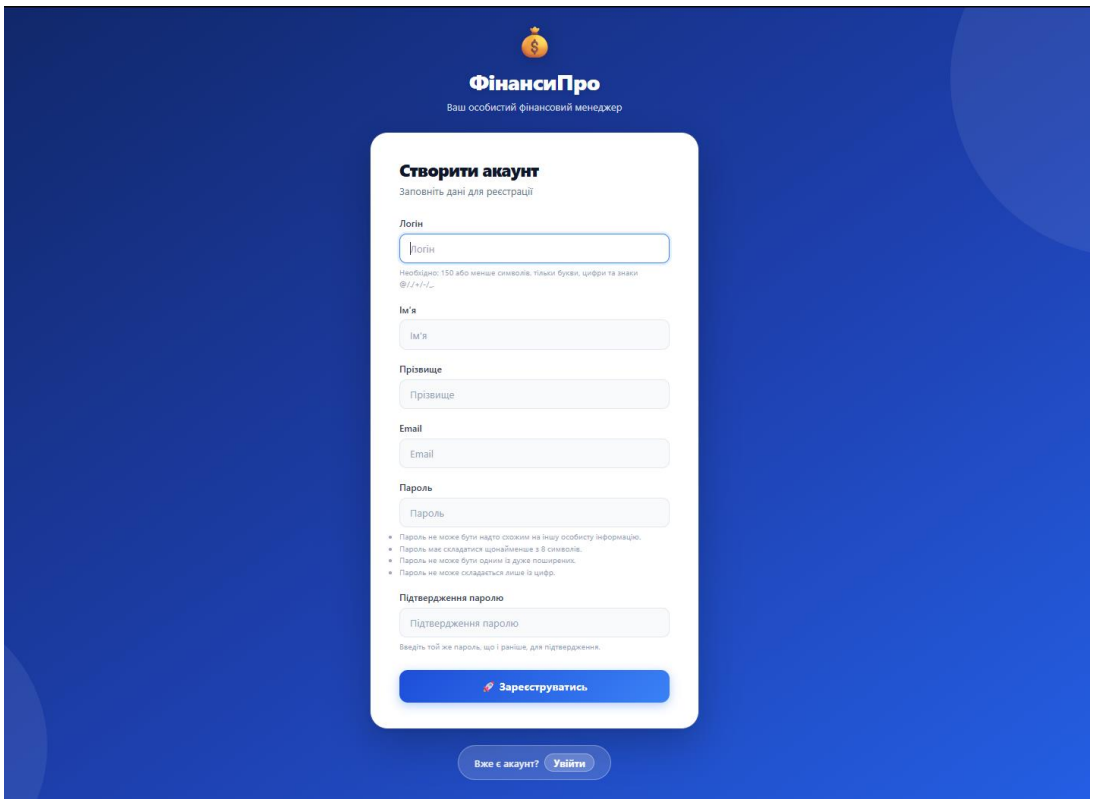


Рисунок Б.2 – Сторінка з формою для створення акаунту

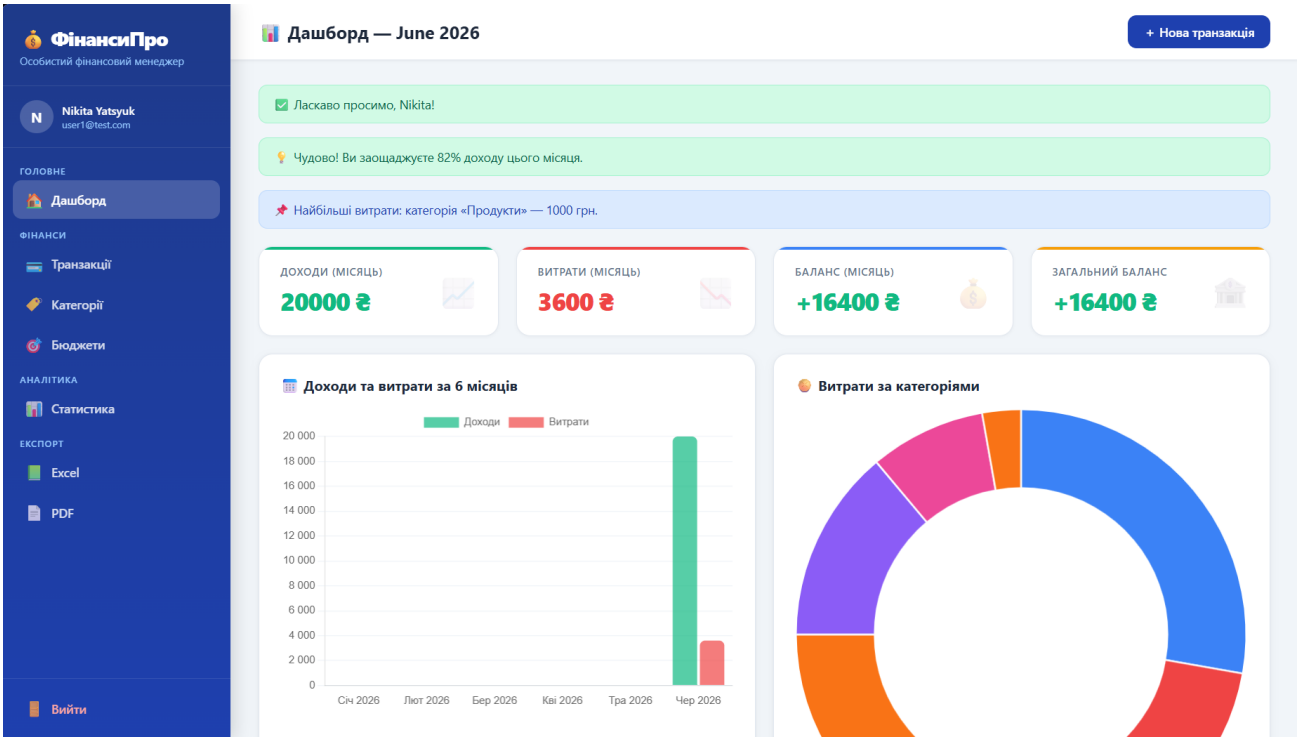


Рисунок Б.3 – Сторінка дашборду (перша частина)

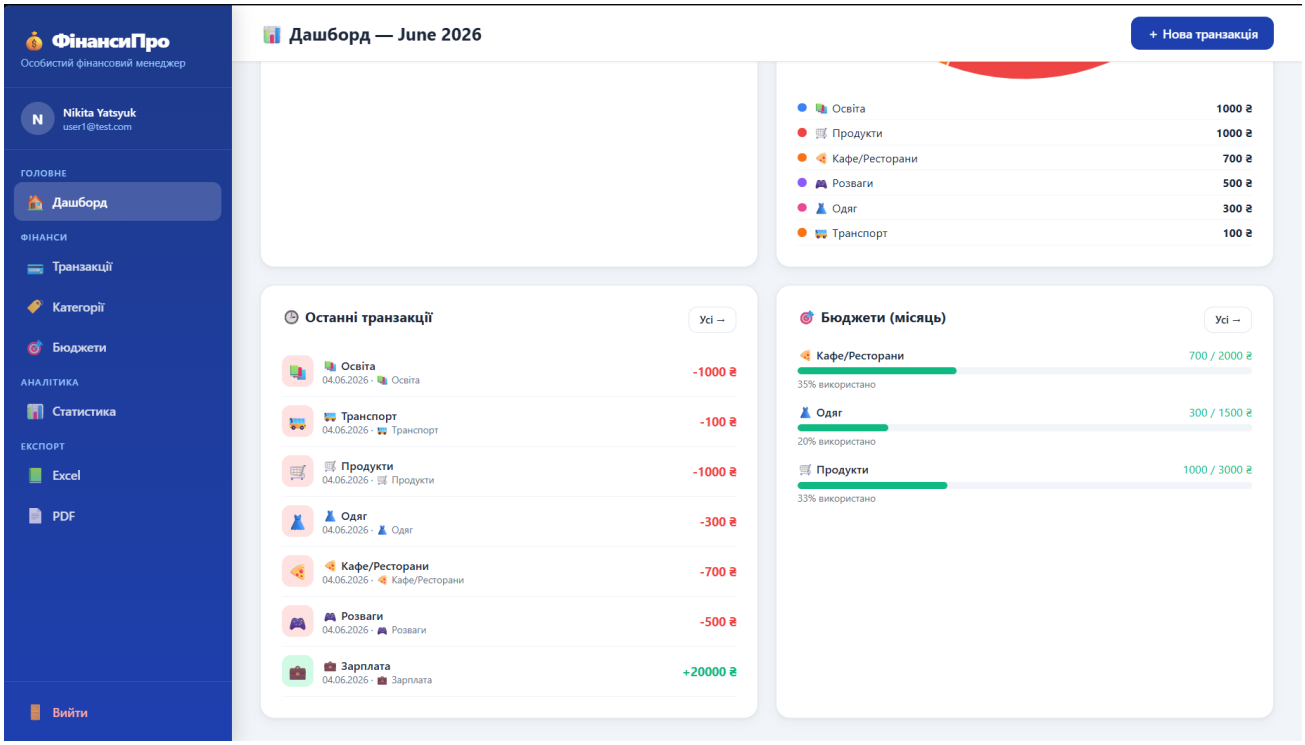


Рисунок Б.4 – Дашборд (друга частина)

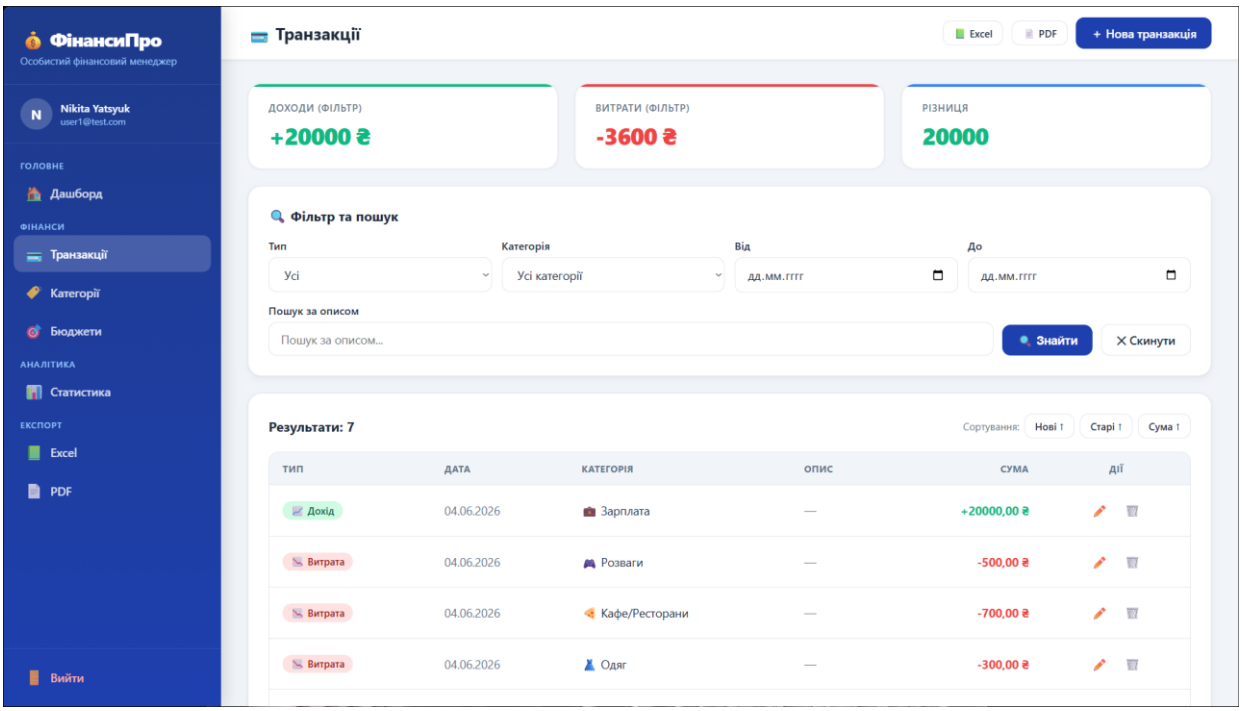


Рисунок Б.5 – Сторінка керування транзакціями (редагування, видалення, пошук за фільтрами або за описом)

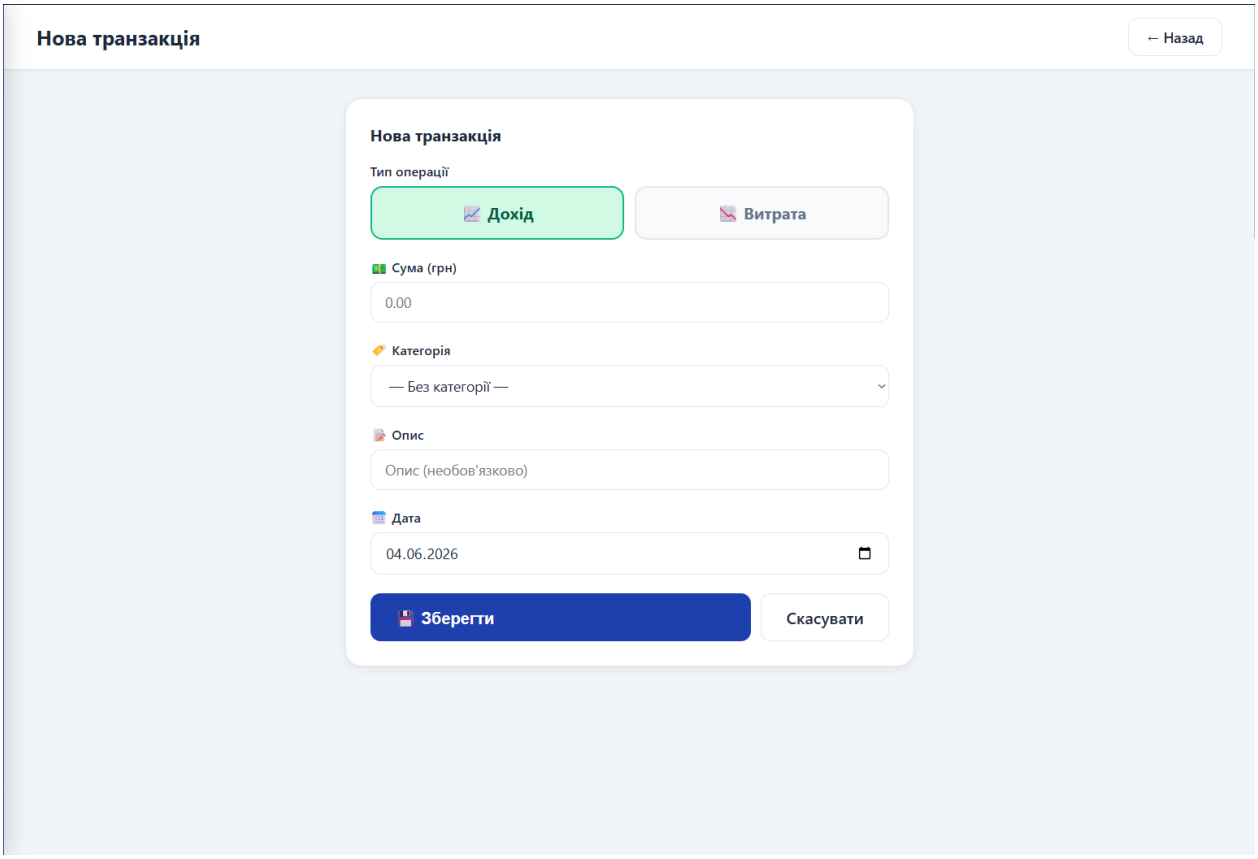


Рисунок Б.6 – Форма додавання нової транзакції (дохід / витрата)

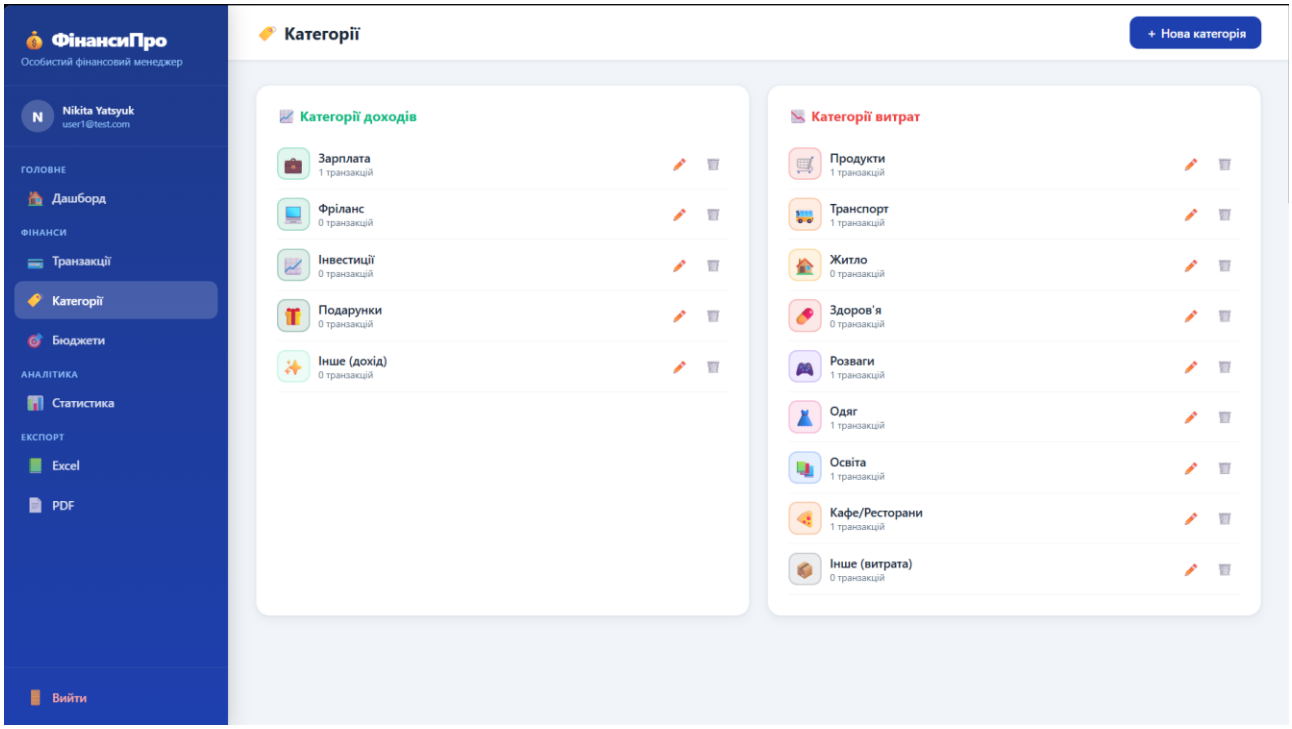


Рисунок Б.7 – Сторінка керування категоріями транзакцій (редагування / видалення)

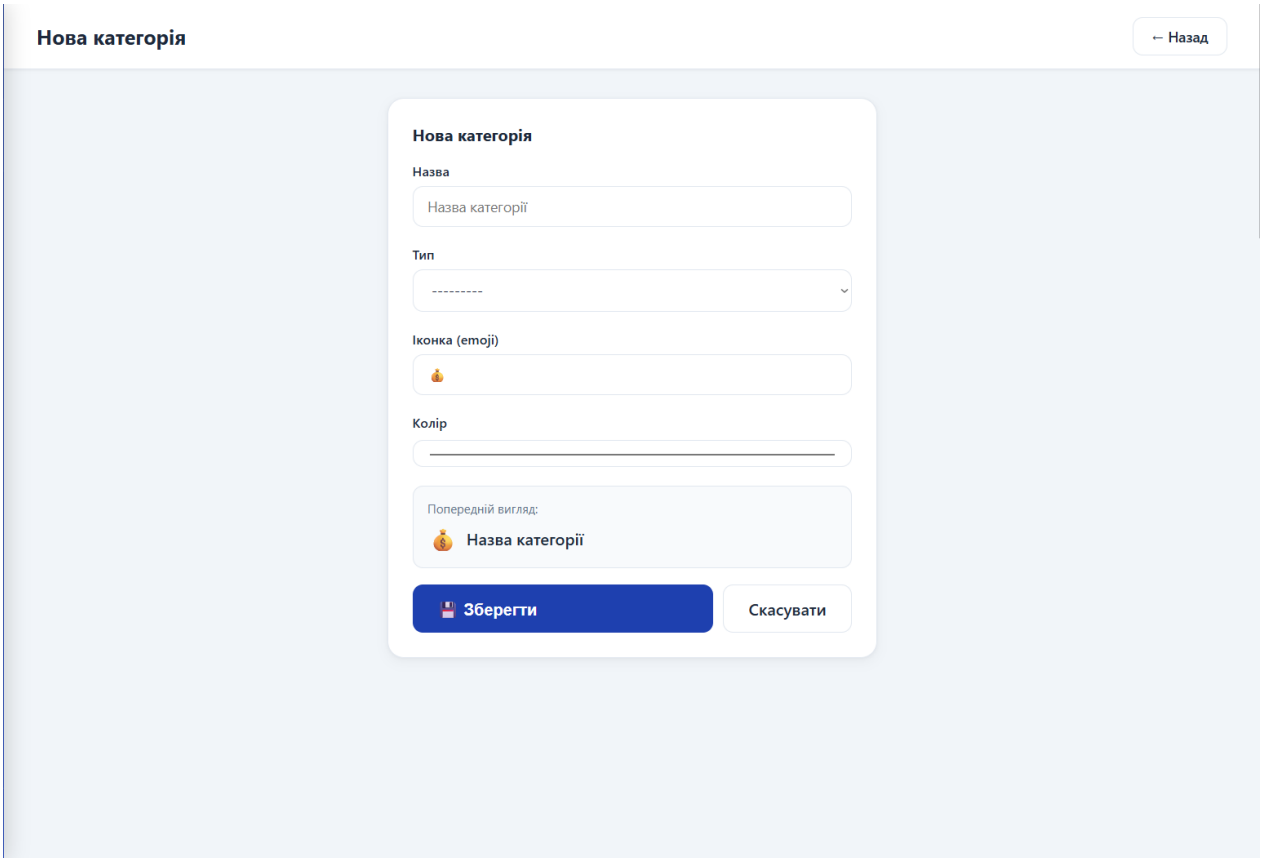


Рисунок Б.8 – Форма додавання нової категорії

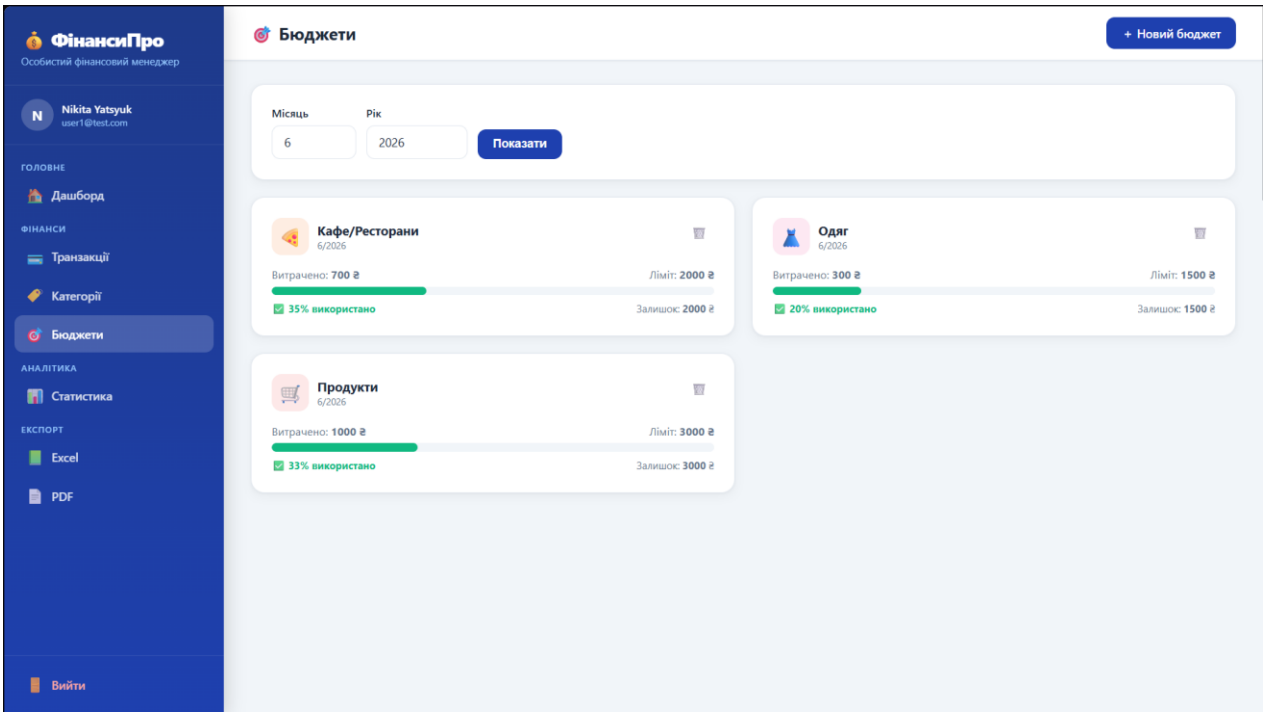


Рисунок Б.9 – Сторінка керування бюджетами

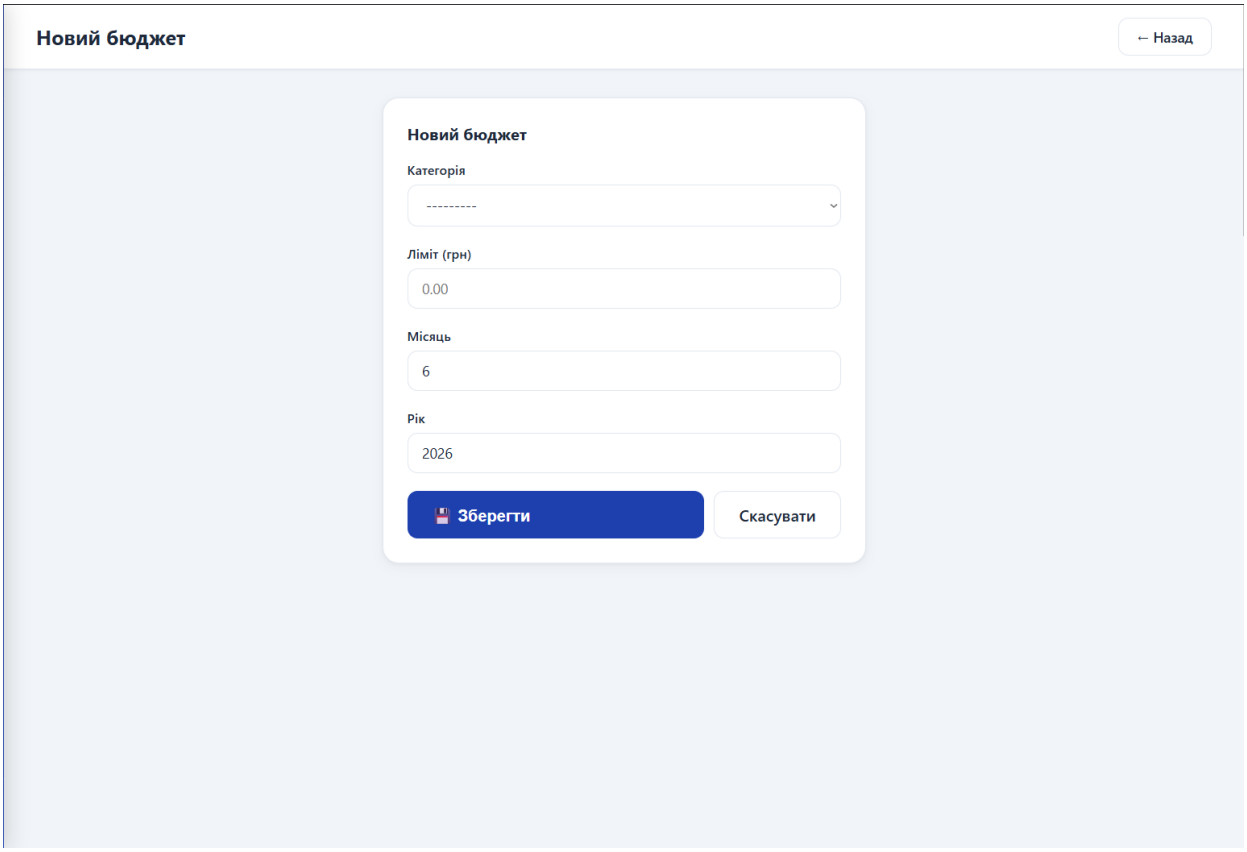


Рисунок Б.10 – Форма додавання нового бюджету

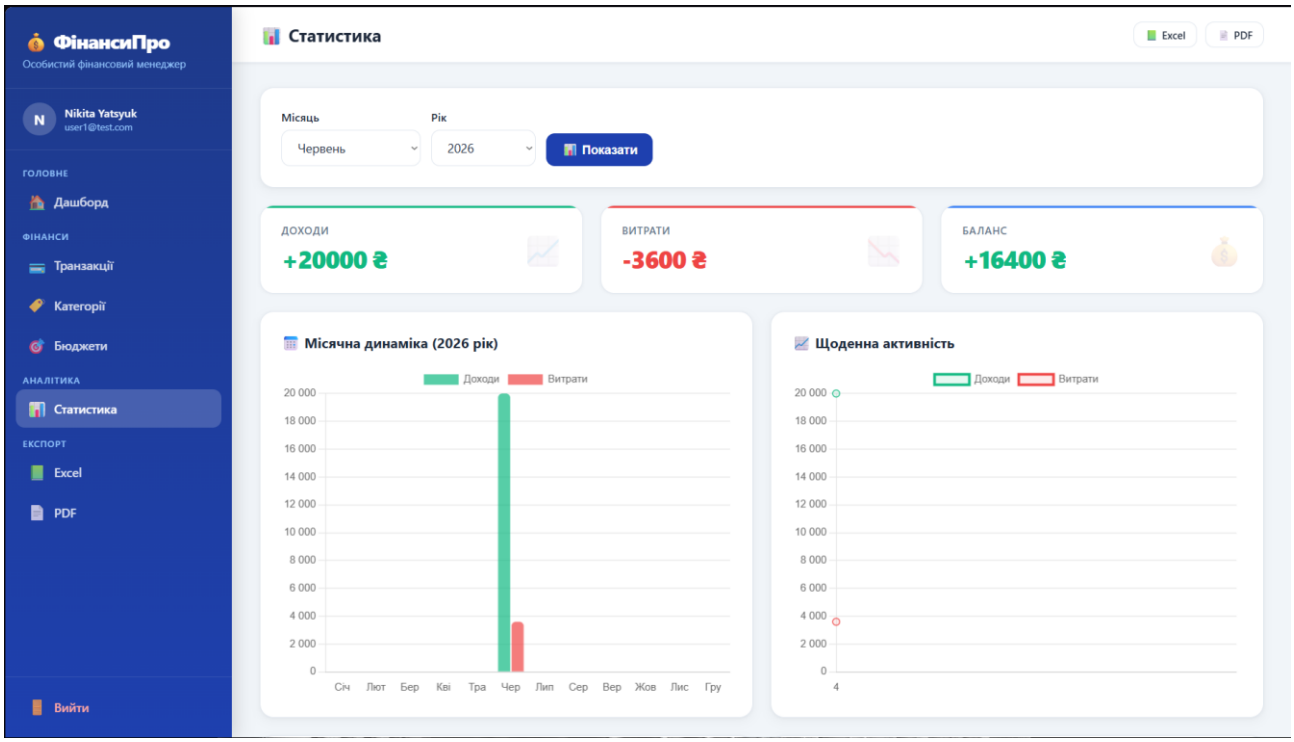


Рисунок Б.11 – Загальна статистика активності за місяць (перша частина)

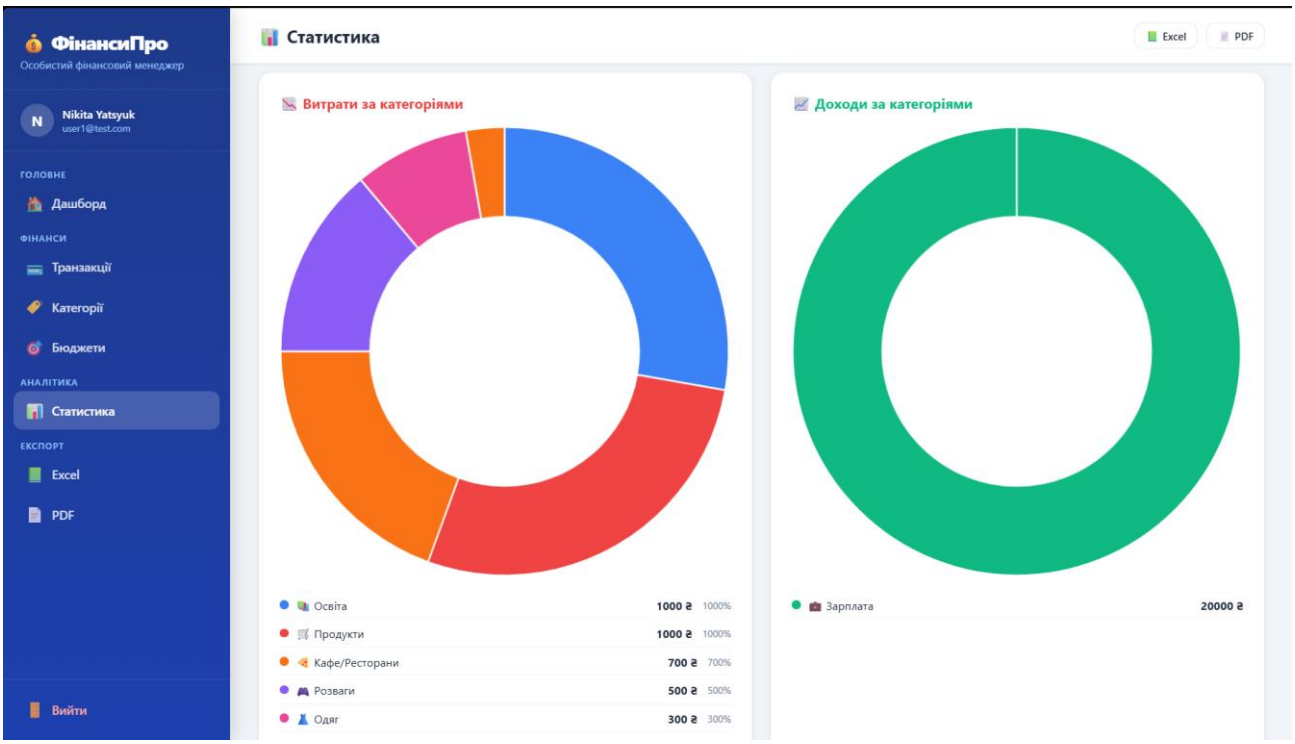
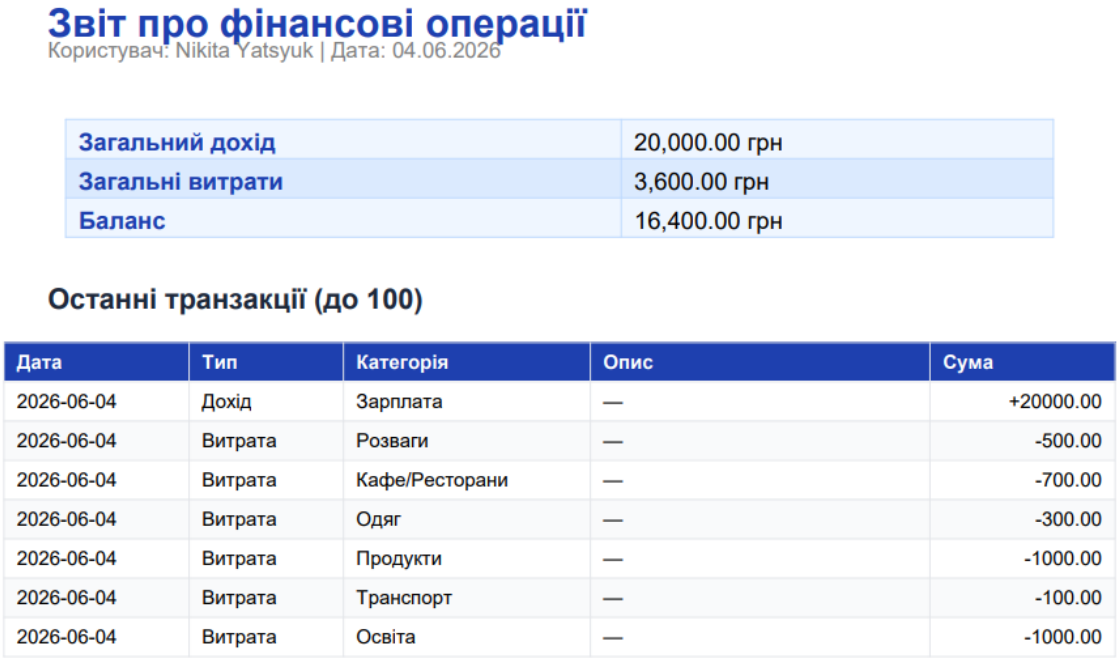


Рисунок Б.12 – Статистика (друга частина)

	A	B	C	D	E	F	G	H
1	№	Дата	Тип	Категорія	Опис	Сума (грн)		
2	1	2026-06-04	Дохід	Зарплата	—	20000		
3	2	2026-06-04	Витрата	Розваги	—	500		
4	3	2026-06-04	Витрата	Кафе/Ресторани	—	700		
5	4	2026-06-04	Витрата	Одяг	—	300		
6	5	2026-06-04	Витрата	Продукти	—	1000		
7	6	2026-06-04	Витрата	Транспорт	—	100		
8	7	2026-06-04	Витрата	Освіта	—	1000		
9								
10								
11								
12								
13								
14								
15								
16								
17								

Рисунок Б.13 – Вигляд таблиці з експортованими даними в форматі Excel



Звіт про фінансові операції

Користувач: Nikita Yatsyuk | Дата: 04.06.2026

Загальний дохід	20,000.00 грн
Загальні витрати	3,600.00 грн
Баланс	16,400.00 грн

Останні транзакції (до 100)

Дата	Тип	Категорія	Опис	Сума
2026-06-04	Дохід	Зарплата	—	+20000.00
2026-06-04	Витрата	Розваги	—	-500.00
2026-06-04	Витрата	Кафе/Ресторани	—	-700.00
2026-06-04	Витрата	Одяг	—	-300.00
2026-06-04	Витрата	Продукти	—	-1000.00
2026-06-04	Витрата	Транспорт	—	-100.00
2026-06-04	Витрата	Освіта	—	-1000.00

Рисунок Б.14 – Вигляд PDF файлу з експортованими даними