ДЕРЖАВНИЙ УНІВЕРСИТЕТ
ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ
НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ
ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ
КАФЕДРА ТЕХНОЛОГІЇ ЦИФРОВОГО РОЗВИТКУ

КВАЛІФІКАЦІЙНА РОБОТА

на тему: «Цифровий асистент управління особистими фінансами»

на здобуття освітнього ступеня бакалавра
зі спеціальності 121 Інженерія програмного забезпечення
освітньо-професійної програми «Технології цифрового розвитку»

Кваліфікаційна робота містить результати власних досліджень. Використання ідей, результатів і текстів інших авторів мають посилання на відповідне джерело

_____ Андрій ФЕСЕНКО
(підпис)

Виконав: здобувач вищої освіти групи ТЦР-43

Андрій ФЕСЕНКО

Керівник: Світлана ПОПЕРЕШНЯК
ст. викладач

Рецензент: _____

Київ 2026

ДЕРЖАВНИЙ УНІВЕРСИТЕТ
ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ

Навчально-науковий інститут інформаційних технологій

Кафедра Технологій цифрового розвитку

Ступінь вищої освіти Бакалавр

Спеціальність 121 Інженерія програмного забезпечення

Освітньо-професійна програма «Технології цифрового розвитку»

ЗАТВЕРДЖУЮ
Завідувач кафедри
Технологій цифрового розвитку
_____ Вікторія ЖЕБКА
« ____ » _____ 2026 р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ

Фесенку Андрію Олеговичу

1. Тема кваліфікаційної роботи: «Цифровий асистент управління особистими фінансами» керівник кваліфікаційної роботи, затверджені наказом Державного університету інформаційно-комунікаційних технологій від «20» лютого 2026 р. № 51.

2. Строк подання кваліфікаційної роботи «18» травня 2026 р.

3. Вихідні дані до кваліфікаційної роботи: офіційна документація фреймворку Flask, бібліотек SQLAlchemy, Flask-Login, Werkzeug та Chart.js, документація API exchangerate-api.com для отримання актуальних курсів валют, наукові джерела з управління особистими фінансами, фінансової аналітики та візуалізації даних, матеріали з проєктування баз даних та архітектури web-застосунків, приклади реалізації систем обліку фінансів з відкритим кодом.

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити)

1) Аналіз предметної галузі та аналіз існуючих рішень для управління особистими фінансами.

2) Аналіз інструментів реалізації та архітектурного проєктування web-застосунку.

3) Програмна реалізація застосунку для автоматизованого обліку фінансів з багатовалютністю.

4) Візуалізація інтерфейсу та тестування функціональності web-застосунку.

5. Перелік графічного матеріалу: *презентація*

1) Аналіз аналогів.

2) Вимоги до програмного забезпечення.
3) Програмні засоби реалізації.
4) Діаграма варіантів використання (гість та користувач).
5) Діаграма варіантів використання (адміністратор).
6) Діаграма класів підбору товарів за типом фігури та кольоротипом.
7) Діаграма структури бази даних.
8) Екранні форми.
9) Апробація результатів дослідження.
6. Дата видачі завдання «20» лютого 2026 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1	Підбір та аналіз науково-технічної літератури	20.02.2026 - 10.03.2026	
2	Аналіз та дослідження існуючих аналогів	11.03.2026 - 15.03.2026	
3	Вибір та обґрунтування технологічного стеку	16.03.2026 - 28.03.2026	
4	Проєктування мікросервісної архітектури та бази даних	01.04.2026 - 08.04.2026	
5	Програмна реалізація застосунку	09.04.2026 - 20.04.2026	
6	Тестування застосунку	21.04.2026 - 29.04.2026	
7	Оформлення роботи: вступ, висновки, реферат	30.04.2026 - 11.05.2026	
8	Розробка демонстраційних матеріалів	12.05.2026 - 17.05.2026	
9	Підготовка презентації та попередній захист	15.05.2026 - 18.05.2026	

Здобувач(ка) вищої освіти _____ Андрій ФЕСЕНКО
(підпис)

Керівник
кваліфікаційної роботи _____ Світлана ПОПЕРЕШНЯК
(підпис)

РЕФЕРАТ

Текстова частина кваліфікаційної роботи на здобуття освітнього ступеня бакалавра: 75 стор., 8 табл., 29 рис., 21 джерел.

Мета роботи – покращити контроль користування особистими фінансами для користувачів та розробити веб-застосунок для обліку та аналізу особистих фінансів з підтримкою багатовалютності та автоматичною конвертацією валют у єдину базову валюту.

Об'єкт дослідження – процес управління особистими фінансами користувача в умовах використання кількох валют.

Предмет дослідження – програмне забезпечення для автоматизованого обліку доходів, витрат та бюджетів з автоматичною конвертацією валют.

Короткий зміст роботи: В роботі проаналізовано сучасні підходи до управління особистими фінансами в умовах багатовалютності. Досліджено наявні системи фінансового обліку, виявлено їх переваги та обмеження. Розроблено архітектуру web-застосунку для управління особистими фінансами із реалізацією функціоналу автоматичної конвертації валют на основі актуальних курсів з відкритого API. Побудовано базу даних для зберігання користувачів, транзакцій, категорій та бюджетів, реалізовано контролери та модулі безпеки за допомогою Flask, SQLAlchemy, Flask-Login та Werkzeug. Система забезпечує категоризацію транзакцій, візуалізацію статистики, систему бюджетів, конвертер валют та персоналізований профіль користувача з можливістю зміни базової валюти. Проведено функціональне та кросбраузерне тестування, перевірено точність конвертації та зручність використання інтерфейсу.

Сферою використання застосунку є автоматизація обліку особистих фінансів для фізичних осіб, зокрема тих, хто отримує доходи в іноземній валюті при витратах у національній валюті.

ЗМІСТ

ВСТУП

Актуальність: у сучасних умовах глобалізації економіки та розвитку безготівкових розрахунків особливої популярності набувають інструменти для систематизації та аналізу особистих фінансів. Значна частина населення України отримує доходи в іноземній валюті (доларах США, євро) при витратах у національній валюті – українській гривні. Це створює потребу в автоматизованих рішеннях, які забезпечують уніфікований облік транзакцій у єдиній базовій валюті з автоматичною конвертацією за актуальними курсами.

Об'єкт дослідження – процес управління особистими фінансами користувача в умовах використання кількох валют.

Предмет дослідження – програмне забезпечення для обліку доходів, витрат, бюджетів з автоматичною конвертацією валют та візуалізації фінансової статистики.

Мета роботи – покращити контроль користування особистими фінансами для користувачів та розробити веб-застосунок для обліку та аналізу особистих фінансів з підтримкою багатовалютності та автоматичною конвертацією валют у єдину базову валюту.

Для досягнення поставленої мети в бакалаврській роботі виконано наступні задачі:

1. Провести аналіз існуючих рішень у сфері управління особистими фінансами, зокрема систем з підтримкою багатовалютності та автоматичною конвертацією.

2. Визначити ключові функціональні вимоги до системи фінансового обліку та дослідити технології їх реалізації.

3. Розробити алгоритм автоматичної конвертації транзакцій у базову валюту користувача на основі актуальних курсів валют.

4. Реалізувати web-застосунок для управління особистими фінансами із функціоналом категоризації, візуалізації статистики, бюджетування, конвертації валют та управління профілем користувача.

5. Провести тестування системи, перевірити точність конвертації валют, зручність використання та загальну ефективність розробленого рішення.

Методи дослідження: аналіз наукових джерел у галузі фінансових технологій та веб-розробки; порівняльний аналіз існуючих програмних платформ; методи системного аналізу, проєктування програмного забезпечення та побудови інформаційних систем; практичні методи розробки та тестування web-застосунків на базі фреймворку Flask.

Наукова новизна роботи полягає у реалізації механізму автоматичної конвертації будь-якої валюти транзакції у вибрану користувачем базову валюту з використанням відкритого API актуальних курсів, що забезпечує уніфіковану аналітику фінансових потоків незалежно від валюти введення. У межах кваліфікаційної роботи запропоновано архітектуру системи, яка поєднує класичний фінансовий облік з інтелектуальною обробкою валютних операцій, забезпечуючи при цьому повну безкоштовність та відкритість рішення.

Практична значущість результатів полягає у створенні повнофункціонального web-застосунку, який може використовуватися фізичними особами для систематизації особистих фінансів. Застосунок дозволяє зменшити час на ручний перерахунок валют, підвищити точність фінансової звітності та забезпечити користувачам зручний, інформативний і персоналізований досвід управління бюджетом.

Робота пройшла апробацію. За її результатами було опубліковано наступні тези доповідей:

1. Фесенко А.О., Поперешняк С.В. Використання штучного інтелекту у ВЕБ-додатках для автоматизації фінансового обліку//VII Всеукраїнська науково-технічна конференція «Застосування програмного забезпечення в інформаційно-комунікаційних технологіях». Збірник тез. К.: ДУІКТ, 2026.

2. Фесенко А.О., Поперешняк С.В. Інтелектуальний цифровий асистент як інструмент розвитку цифрової економіки та електронного врядування //Всеукраїнська науково-технічна конференція «Технології цифрового розвитку: програмні системи та децентралізація». Збірник тез. К.: ДУІКТ, 2026.

3. Фесенко А.О., Поперешняк С.В. БЕЗПЕКА IoT-мереж //VII Науково-технічна конференція «Сучасний стан та перспективи розвитку IoT». Збірник тез. К.: ДУІКТ, 2026

1 АНАЛІЗ ПРЕДМЕТНОЇ ГАЛУЗІ

1.1 Характеристика предметної галузі

Сфера фінансових технологій (fintech) та особистого фінансового менеджменту активно розвивається в умовах цифровізації економіки та зростання популярності безготівкових розрахунків. За даними Національного банку України, частка безготівкових операцій у загальному обсязі платежів фізичних осіб перевищує 60%, при цьому значна частка населення отримує доходи в іноземній валюті або використовує декілька валютних рахунків одночасно [1].

Розвиток фінансових технологій (fintech) є одним із найдинамічніших сегментів цифрової економіки. За даними консалтингової компанії KPMG, глобальні інвестиції у fintech-стартапи у 2024 році склали 113 мільярдів доларів США, що на 31% більше порівняно з 2023 роком. Основними драйверами зростання є: розвиток штучного інтелекту для персоналізації фінансових послуг, впровадження відкритого банкінгу (Open Banking), розширення екосистем цифрових платежів та зростання попиту на інструменти фінансової грамотності.

Сегмент Personal Finance Management (PFM) займає особливе місце у структурі fintech-ринку. За прогнозами Juniper Research, кількість активних користувачів PFM-застосунків у світі досягне 1,8 мільярда до 2027 року. Це зумовлено не лише технічною доступністю, а й зміною фінансової поведінки населення: сучасні користувачі прагнуть не просто фіксувати витрати, а отримувати аналітичну інформацію для прийняття рішень, прогнозувати фінансові потоки та оптимізувати структуру бюджету.

В українському контексті розвиток PFM-систем тісно пов'язаний із процесами діджиталізації банківського сектору. За даними НБУ, станом на 1 січня 2025 року в Україні функціонує 63 банки, з яких 48 пропонують мобільні застосунки з елементами фінансової аналітики. Проте ці інструменти мають внутрішньобанківську спрямованість та не вирішують завдання комплексного обліку фінансів з різних джерел.

Особливістю українського ринку є висока доля населення з мультивалютними доходами. За результатами дослідження Міжнародної організації міграції (IOM), близько 4,5 мільйона українців працюють за кордоном, при цьому значна частка трудових мігрантів отримує доходи в іноземній валюті та переказує кошти в Україну. Внутрішній ринок праці також трансформується: за даними Держстату, частка зайнятих у сфері інформаційних технологій та наданні послуг за контрактом з іноземними компаніями зросла з 3,2% у 2019 році до 8,7% у 2024 році.

Така структура доходів створює специфічні виклики для фінансового обліку. Користувачеві необхідно: відстежувати операції в різних валютах, враховувати коливання обмінних курсів, консолідувати дані для отримання цілісної картини бюджету, планувати витрати з урахуванням валютних ризиків. Існуючі інструменти або не підтримують ці функції, або реалізують їх фрагментарно та несистемно.

Предметна галузь даної кваліфікаційної роботи охоплює перетин кількох дисциплін: фінансового менеджменту (методології обліку та бюджетування), програмної інженерії (архітектура та розробка веб-застосунків), валютного ринку (механізми конвертації та курсоутворення) та дизайну інтерфейсів (UX/UI принципи побудови зручних інструментів). Такий міждисциплінарний характер визначає складність та комплексність поставленого завдання.

Особливу актуальність має проблема ведення обліку в умовах мультивалютності. В Україні, де значна частка населення отримує доходи в доларах США або євро при витратах у національній валюті (гривні), виникає необхідність автоматичної конвертації та уніфікації фінансової звітності. Існуючі рішення часто не забезпечують належного рівня автоматизації цього процесу, вимагають ручного перерахунку або є комерційними продуктами з обмеженим функціоналом у безкоштовній версії.

Предметна галузь охоплює міждисциплінарну область, яка включає елементи fintech, веб-розробки, обробки фінансових даних, візуалізації інформації та проєктування баз даних. Особливу значимість мають технології, що дозволяють на

основі вхідних параметрів (сум транзакцій, валют, категорій) автоматично формувати персоналізовану аналітику, рекомендації щодо бюджетування та забезпечувати конвертацію фінансових потоків у єдину базову валюту [2; 3].

1.2 Аналіз існуючих аналогів

У сфері управління особистими фінансами існує значна кількість програмних рішень — від мобільних додатків до самостійно розгортуваних веб-платформ. У цьому розділі проведено аналіз існуючих програмних продуктів, які частково або повністю реалізують функціональність, аналогічну до тієї, що розробляється у межах даної бакалаврської роботи. Зокрема, розглянуто платформи Spendee [5], Firefly III [6] та Homemoney[7].

1.2.1 Аналіз платформи Spendee

Spendee — популярний фінансовий трекер, розроблений чеською компанією Cleevio s.r.o., який доступний як у вигляді мобільних додатків для iOS та Android, так і у веб-версії через браузер. Платформа орієнтована на візуалізацію фінансових даних та спрощення процесу обліку для широкого кола користувачів.

Основні можливості Spendee включають створення гаманців у різних валютах, ручне додавання транзакцій з категоризацією, створення бюджетів на період, візуалізацію витрат у вигляді кругових діаграм та списків. Платформа підтримує синхронізацію між пристроями, спільні гаманці для сімейного обліку, а також базовий експорт даних. Веб-версія Spendee доступна за адресою app.spendee.com та повторює основний функціонал мобільних додатків. Інтерфейс виконаний у сучасному плоскому стилі з акцентом на візуальну складову. Головний екран містить загальний баланс, останні транзакції та швидкий доступ до основних функцій.

Проте для вирішення завдань, пов'язаних із автоматичною конвертацією транзакцій у базову валюту, Spendee має суттєві обмеження. Платформа підтримує створення гаманців у різних валютах, але кожен гаманець існує ізольовано — загальна

аналітика витрат ведеться окремо за кожною валютою без консолідації. Користувач не може встановити єдину базову валюту для автоматичного перерахунку всіх операцій. При перегляді загального балансу суми з різних гаманців відображаються без конвертації, що ускладнює отримання цілісної картини фінансового стану.

Автоматична конвертація в Spendee відсутня повністю. Користувач може лише вручну додати транзакцію у валюті гаманця, але система не пропонує інструментів для перерахунку за актуальним курсом. Відсутній також вбудований конвертер валют з актуальними курсами.

Особистий профіль користувача в Spendee має базовий функціонал — можливість зміни імені, аватара, мови інтерфейсу. Проте відсутня можливість зміни базової валюти обліку, що є критично важливим для користувачів з мультивалютними доходами.

До переваг Spendee можна віднести:

1) сучасний інтуїтивний інтерфейс з якісною візуалізацією даних;

2) підтримка веб-версії для роботи через браузер на комп'ютері;

3) можливість створення спільних гаманців для сімейного бюджету;

4) синхронізація між мобільними пристроями та веб-версією;

5) базова безкоштовна версія з можливістю розширення функціоналу через підписку.

Недоліки, що впливають на ефективність для завдань, що розглядаються в межах бакалаврської роботи:

1) відсутність автоматичної конвертації транзакцій у базову валюту;

2) ізольований облік за валютами без консолідації аналітики;

3) відсутність вибору базової валюти користувачем;

4) відсутність вбудованого конвертера валют з актуальними курсами;

5) обмежений функціонал безкоштовної версії (лише 1 спільний гаманець, обмежена кількість бюджетів);

6) базовий профіль користувача без можливості зміни ключових параметрів обліку.

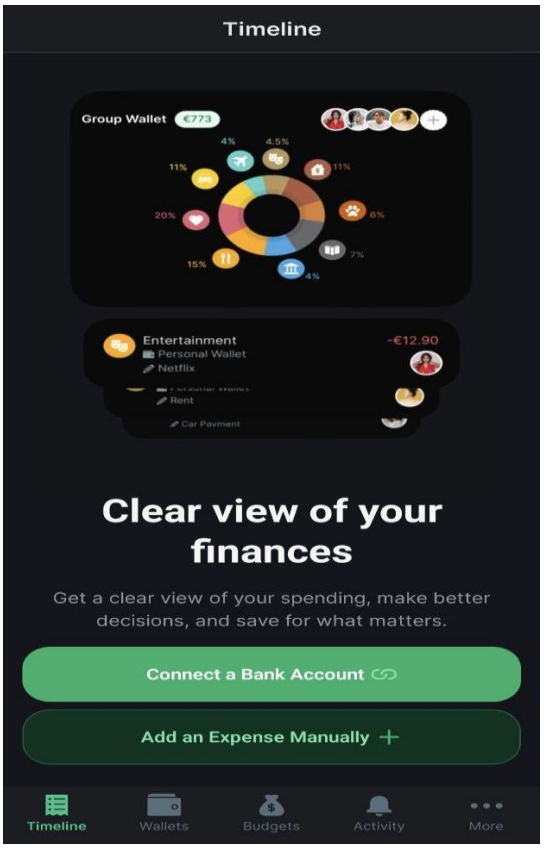

Рис. 1.1 Головний екран Spendee

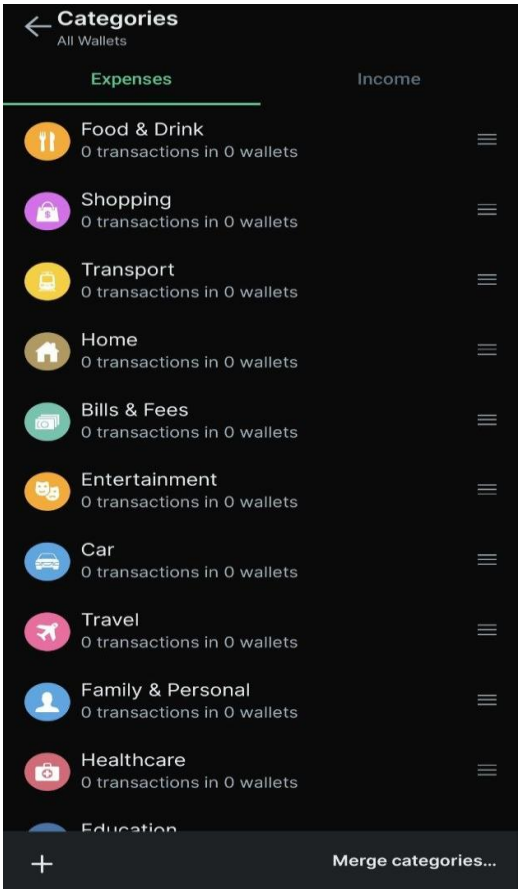

Рис. 1.2 Вкладка з категоріями

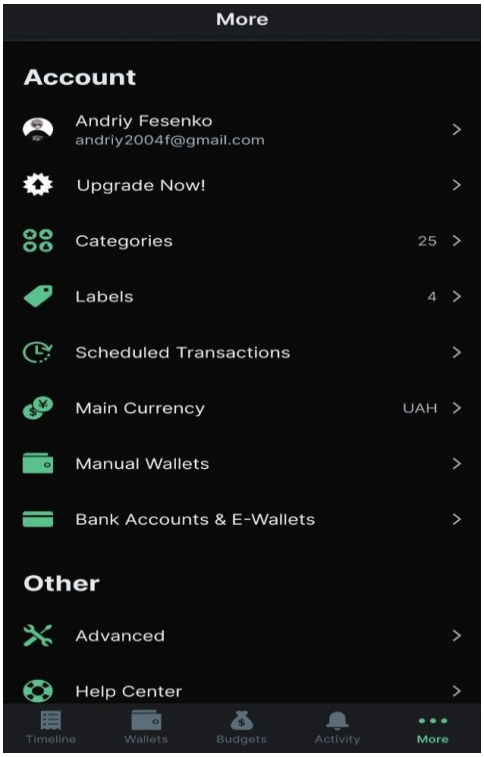

Рис. 1.3 Розширний інтерфейс

1.2.2 Аналіз платформи Firefly III

Firefly III — самостійно розгортувана (self-hosted) система управління особистими фінансами з відкритим вихідним кодом, розроблена голландським розробником James Cole. Платформа розповсюджується за ліцензією AGPL-3.0 та може бути розгорнута на власному сервері користувача. Доступна веб-версія для роботи через браузер, а також існують сторонні мобільні додатки-клієнти для Android та iOS.

Основні можливості Firefly III включають подвійний облік (double-entry bookkeeping), категоризацію транзакцій, створення бюджетів, цілей накопичення, управління рахунками в різних валютах, візуалізацію даних у вигляді графіків та діаграм, а також експорт у різні формати. Платформа підтримує автоматичне отримання курсів валют з відкритих джерел та конвертацію сум для відображення.

Веб-інтерфейс Firefly III виконаний у функціональному стилі з акцентом на повноту даних та гнучкість налаштувань. Головна сторінка містить оглядову

панель з балансами рахунків, графіками витрат, інформацією про бюджети та цілі. Платформа підтримує необмежену кількість рахунків у різних валютах з можливістю встановлення "рідної" (native) валюти для кожного рахунку окремо. Однією з важливих функцій Firefly III є підтримка конвертації валют. Система дозволяє створювати транзакції з вказанням валюти та отримує актуальні курси з відкритих API (fixer.io, exchangerate.host). Проте конвертація використовується переважно для відображення загальних балансів, а не для уніфікації аналітики за категоріями. Користувач не може встановити єдину базову валюту для всієї системи та отримати аналітику витрат за категоріями в цій валюті — звіти формуються роздільно за валютами рахунків.

Профіль користувача в Firefly III має розширений функціонал налаштувань, включаючи вибір мови, часового поясу, формату дат та чисел. Проте відсутня концепція "базової валюти користувача" як глобального параметра системи. Кожен рахунок має власну валюту, і користувач повинен самостійно відстежувати співвідношення між ними.

Також варто зазначити, що Firefly III орієнтований на технічно підкованих користувачів, оскільки вимагає самостійного розгортання на сервері (Docker, VPS або локальному комп'ютері). Це створює бар'єр для широкого кола користувачів, які не мають навичок системного адміністрування.

Переваги Firefly III:

1) повністю відкритий вихідний код та безкоштовне використання;
2) підтримка подвійного обліку для точного бухгалтерського обліку;
3) можливість самостійного розгортання на власному сервері;
4) підтримка рахунків у різних валютах з отриманням актуальних курсів;
5) розширена система бюджетів, цілей та правил автоматизації;
6) наявність API для інтеграції зі сторонніми сервісами.

Недоліки:

1) складність розгортання для нетехнічних користувачів (потрібен сервер, Docker);
2) відсутність єдиної базової валюти для консолідації аналітики;

3) аналітика за категоріями ведеться роздільно за валютами без автоматичної конвертації;

4) функціональний, але не сучасний інтерфейс користувача;

5) відсутність офіційної мобільної версії (лише сторонні клієнти);

6) базовий профіль користувача без гнучкого управління валютними налаштуваннями.

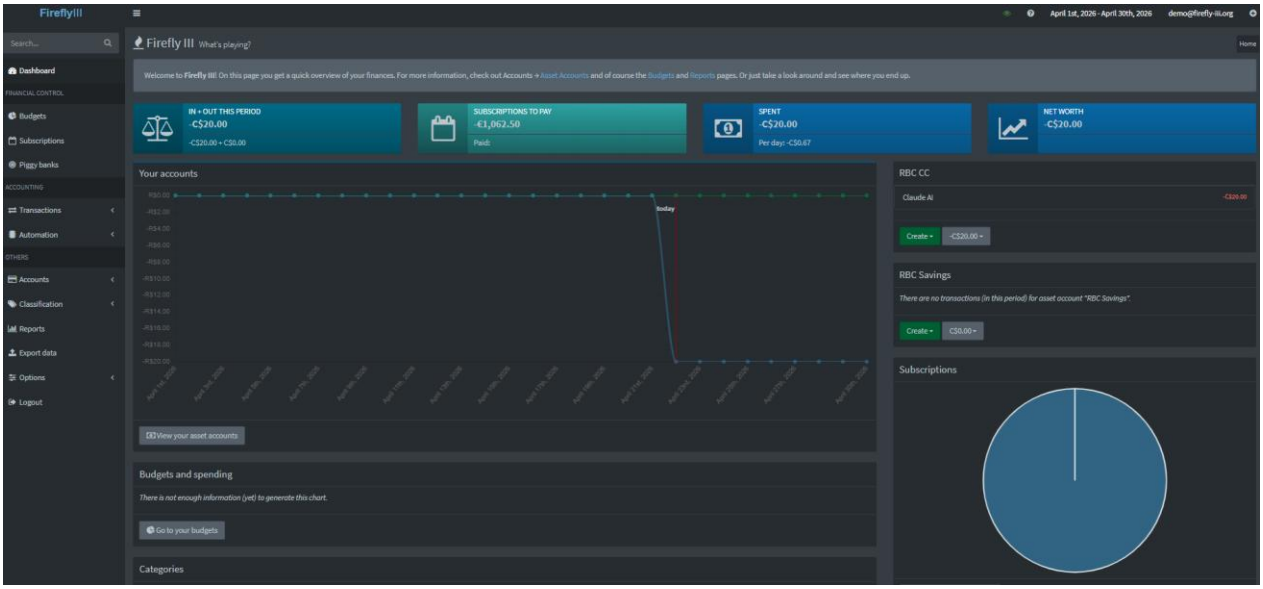

Рис. 1.4 Оглядова панель Firefly III з балансами та графіками

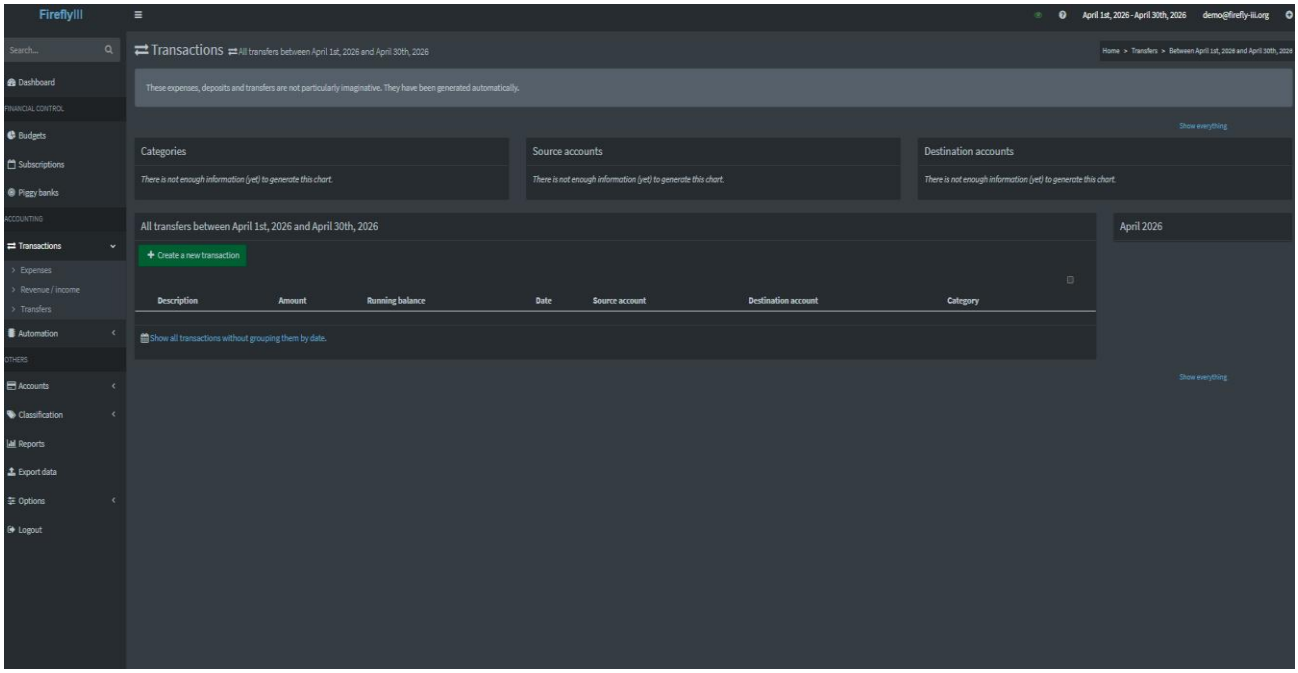

Рис. 1.5 Сторінка транзакцій з деталізацією в Firefly III

1.2.3 Аналіз платформи Homemoney

Для оцінки стану та тенденцій розвитку вітчизняного ринку програмного забезпечення у сфері персонального фінансового менеджменту було досліджено український веб-сервіс Homemoney. Дана система є одним із перших успішних українських стартапів у цьому сегменті та розроблялася з урахуванням специфіки національної банківської системи та особливостей ведення бюджету українськими домогосподарствами.

Homemoney реалізовано у вигляді кросплатформеного рішення, що включає веб-версію та мобільні клієнти. Основний фокус системи спрямовано на максимальну простоту введення даних та наочність звітів. Продукт підтримує роботу з основними валютами, що мають обіг в Україні (UAH, USD, EUR, PLN), а також дозволяє користувачеві самостійно додавати рідкісні валюти.

Конвертація валют у Homemoney здійснюється автоматично на основі щоденних курсів, які імпортуються з офіційних даних Національного банку України (НБУ). Користувач визначає базову валюту профілю (найчастіше — гривню), і всі мультивалютні рахунки, вклади чи кредити автоматично консолідуються в інтегральний показник «Чиста вартість» (Net Worth).

Візуалізація даних у Homemoney базується на класичних аналітичних інструментах:

1) Календар платежів — унікальний інтерфейсний елемент, який дозволяє планувати майбутні регулярні витрати (комунальні послуги, кредити) та бачити прогнозований баланс на будь-який день місяця.

2) Структура витрат — кругові діаграми з можливістю деталізації (drilling down) від загальної категорії (наприклад, «Транспорт») до конкретного підпункту («Бензин» або «Таксі»).

Переваги системи Homemoney:

1) Орієнтація на український ринок: інтеграція офіційного курсу НБУ, що забезпечує точність розрахунків для українських реалій.

2) Простота планування: наявність інтерактивного фінансового календаря для запобігання «касовим розривам» особистого бюджету.

3) Гнучка система боргів: розвинений модуль обліку позик, кредитів та взаєморозрахунків з іншими людьми.

Недоліки системи Hometmoney:

1) Застарілий технологічний стек: веб-інтерфейс системи тривалий час не отримував кардинальних оновлень, що негативно впливає на швидкість завантаження сторінок (Performance) порівняно з сучасними SPA (Single Page Applications).

2) Обмеження безкоштовної версії: ліміт на кількість створюваних рахунків та категорій у безкоштовному тарифі, а також наявність реклами у веб-інтерфейсі.

3) Проблеми з безпекою (Cloud-залежність): дані користувачів зберігаються на віддалених серверах компанії, що створює ризики витоку конфіденційної фінансової інформації у разі кібератак.

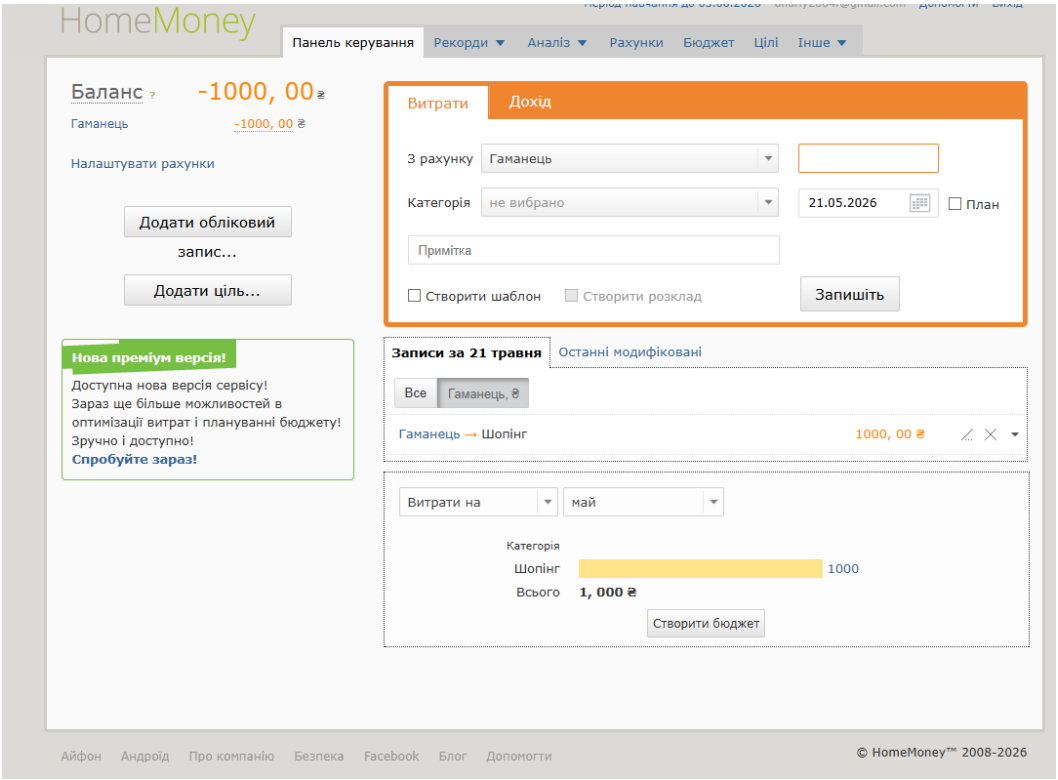

Рис. 1.6 Сторінка панелі керування

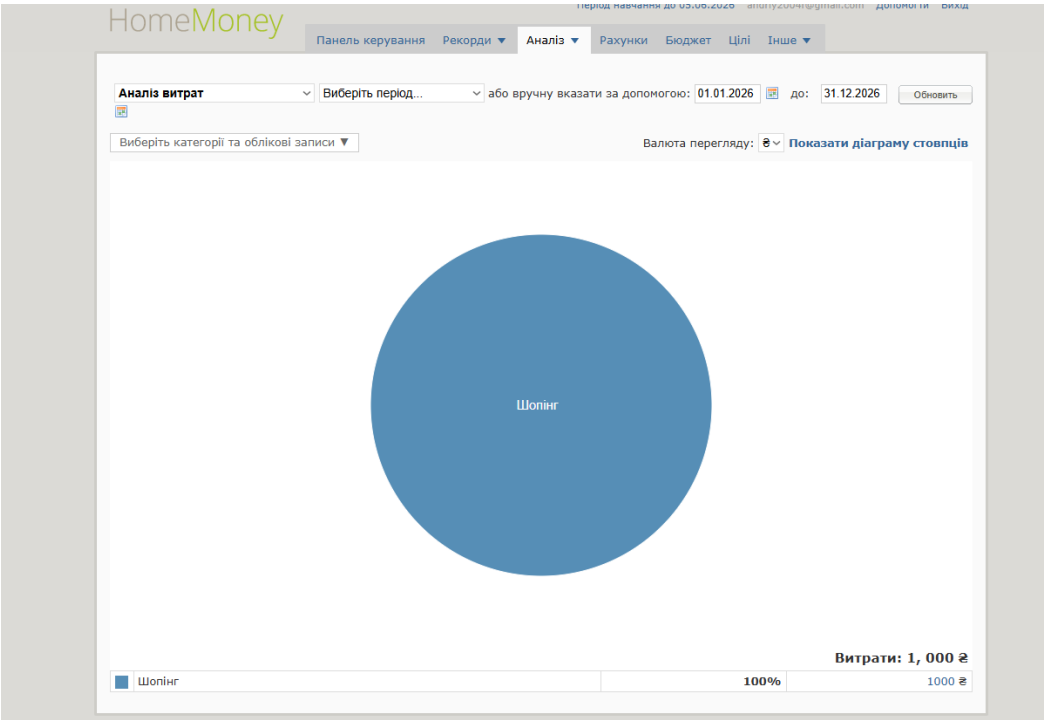

Рис. 1.7 Сторінка аналізу витрат

Таблиця 1.1

Зведена оцінка можливостей персоналізації та функціоналу web-додатків

Функціональність/Показник	**Spendee**	**Firefly III**	Hometone y
Наявність web-платформи	+	+	+
Доступ до Android	+	-	+
Облік транзакцій за категоріями	+	+	+
Візуалізація статистики(графіки)	+	+	+
Підтримка багатовалютності	+	+	+
Автоматична конвертація в базову валюту	-	+	+
Вибір базової валюти користувачем	-	-	-
Конвертер валют з актуальними курсами	-	-	+
Система бюджетів	+	+	-
Експорт даних	+	+	-
Простота розгортання	+	-	+

1.3 Визначення проблем предметної галузі

У ході дослідження сучасного ринку програмного забезпечення для персонального фінансового менеджменту було проведено аналіз існуючих аналогів, що дозволило визначити переваги та недоліки сучасних рішень, а також сформувати вимоги до проектованого цифрового асистента. До розряду аналогів увійшли: комерційний мобільний застосунок Spendee, open-source веб-платформа Firefly III, а також вітчизняний хмарний веб-сервіс Homemoney.

1. Пропрієтарна хмарна система Spendee.

Популярний мобільний трекер витрат, орієнтований на кінцевого користувача.

Переваги: сучасний та ергономічний користувацький інтерфейс, наявність механізмів синхронізації з банківськими рахунками через відкриті API (PSD2), можливість створення спільних гаманців для ведення сімейного бюджету.

Недоліки: обмежені можливості мультивалютного обліку у безкоштовній версії, що вимагає або ручного введення курсів, або фіксації лише однієї базової валюти. Зберігання фінансових транзакцій на серверах третіх сторін знижує рівень конфіденційності даних. Наявність реклами та платної підписки для розблокування базових аналітичних графіків.

2. Відкрита self-hosted система Firefly III.

Спеціалізоване веб-орієнтоване open-source рішення, що вимагає самостійного розгортання користувачем на власному сервері.

Переваги: абсолютна автономність і конфіденційність даних (база даних знаходиться під повним контролем користувача), відсутність платного функціоналу чи реклами, потужний двигун автоматизації за допомогою правил для категоризації транзакцій.

Недоліки: висока складність розгортання та адміністрування для пересічного користувача (необхідність знання Docker та налаштування веб-серверів). Повна відсутність офіційного україномовного інтерфейсу та складна архітектура додавання мультивалютних записів.

3. Вітчизняна хмарна система Homemoney.

Один із перших українських PFM-сервісів, створений вітчизняними розробниками з урахуванням специфіки національної фінансової системи та особливостей ведення бюджету українськими домогосподарствами.

Переваги: глибока інтеграція з українським фінансовим простором, автоматична конвертація мультивалютних рахунків на основі офіційних щоденних даних Національного банку України (НБУ). Наявність унікального елемента — «Календаря платежів» для планування майбутніх регулярних витрат і прогнозування балансу, а також повна україномовна локалізація.

Недоліки: застарілий технологічний стек веб-інтерфейсу, який тривалий час не оновлювався та не оптимізований під сучасні шаблони швидких веб-додатків. Безкоштовна версія має суворі ліміти на кількість категорій і рахунків та містить сторонні рекламні банери. Дані зберігаються в хмарі провайдера, що створює ризики несанкціонованого доступу.

1.4 Постановка завдання на розробку системи

На основі проведеного аналізу предметної галузі, вивчення наукових джерел [3;4; 8;9], а також результатів порівняльного аналізу існуючих аналогів (платформ Spendee, Firefly III, Homemoney), було виявлено низку проблем, які на сьогодні не вирішуються в межах безкоштовних програмних рішень з українською локалізацією. Ці проблеми стосуються насамперед відсутності автоматичної конвертації валют у базову валюту користувача, обмеженої багатовалютності, фрагментації обліку між банківськими та загальними системами, а також відсутності доступних веб-застосунків для широкого кола користувачів.

Деталізація завдання на розробку системи потребує чіткого визначення меж проєкту, цільової аудиторії та критеріїв успіху. Це забезпечує фокусування ресурсів на ключовій функціональності та запобігає розмиванню архітектури непріоритетними функціями.

Цільова аудиторія. Первинною аудиторією розроблюваного застосунку є

фізичні особи віком 22-45 років, які:

1) отримують доходи в іноземній валюті (трудові мігранти, фрілансери, віддалені працівники іноземних компаній);

1) ведуть облік сімейного бюджету з кількома джерелами доходу;

2) потребують консолідованої аналітики витрат незалежно від валюти операцій;

3) цінують конфіденційність фінансових даних та контроль над їх зберіганням;

4) мають базові навички роботи з комп'ютером та веб-браузером.

Вторинною аудиторією є малий бізнес (ФОП), яким потрібен простий інструмент обліку доходів та витрат з можливістю експорту даних для податкової звітності.

Межі проєкту (Scope). У межах кваліфікаційної роботи реалізовується MVP (Minimum Viable Product) — мінімально життєздатний продукт з фокусом на ключову цінність: автоматичну конвертацію валют та уніфіковану аналітику. До складу MVP входять:

1) реєстрація/авторизація з вибором базової валюти;

2) CRUD-операції з транзакціями з автоматичною конвертацією;

3) система категорій з кастомізацією;

4) візуалізація статистики (doughnut chart, line chart);

5) система бюджетів з прогрес-барами;

6) конвертер валют з актуальними курсами;

7) профіль користувача з можливістю зміни налаштувань;

8) експорт даних у CSV.

За межами проєкту (Out of Scope). Наступні функції не входять до MVP, проти визначені як перспективні для подальшого розвитку:

1) інтеграція з банківськими API для автоматичного імпорту;

2) мобільний застосунок (iOS/Android);

3) спільні бюджети для кількох користувачів;

4) прогнозування витрат на основі ML;

5) підтримка криптовалют;

6) інтеграція з податковою звітністю.

Критерії успіху. Проєкт вважається успішним за виконання наступних умов:

1) Точність конвертації валют відповідає курсу API з похибкою не більше 0,5%;

2) Час завантаження сторінки не перевищує 2 секунд при швидкості з'єднання 10 Мбіт/с;

3) Оцінка зручності використання (SUS) перевищує 68 балів;

4) Система коректно функціонує у 4+ сучасних браузерах;

5) Код покритий базовими тестами з покриттям не менше 60%.

У межах цієї кваліфікаційної роботи ставиться завдання створити web-застосунок для управління особистими фінансами, який би забезпечував автоматичну конвертацію транзакцій у базову валюту користувача з підтримкою мультивалютного обліку. Система повинна поєднати переваги банківських додатків (автоматизація, сучасний інтерфейс) з гнучкістю самостійно розгортуваних рішень (контроль даних, кастомізація), уникаючи при цьому їхніх недоліків.

Для реалізації поставленої задачі передбачається:

1) реалізація модуля автоматичної конвертації валют на основі актуальних курсів з відкритого API exchangerate-api.com з механізмом кешування та резервними курсами;

2) розробка системи вибору базової валюти користувачем під час реєстрації з можливістю подальшої зміни в профілі;

3) формування блоку персоналізованої статистики, яка відображатиме всі операції у єдиній базовій валюті незалежно від валюти введення;

4) інтеграція функціоналу класичного фінансового трекера: категоризація доходів/витрат, візуалізація статистики, система бюджетів, конвертер валют;

5) створення особистого кабінету користувача з можливістю редагування профілю, зміни базової валюти, імені, email та пароля;

6) реалізація адміністративних функцій (управління категоріями, експорт

даних у CSV);

7) використання сучасного технологічного стеку: Python, Flask, SQLAlchemy, SQLite, Chart.js, HTML/CSS/JavaScript;

8) забезпечення захисту персональних даних користувача через хешування паролів (bcrypt-алгоритм через werkzeug.security);

9) реалізація адаптивного дизайну для коректного відображення на десктопах, планшетах та мобільних пристроях.

Поставлене завдання передбачає комплексний підхід до реалізації системи, яка поєднує інструменти фінансового обліку з інтелектуальною обробкою валютних операцій. Особлива увага буде приділена користувацькому досвіду (UX), зручності взаємодії, швидкості відповіді системи та точності конвертації валют. У результаті реалізації проєкту очікується створення повнофункціонального web-застосунку, здатного якісно підвищити рівень автоматизації фінансового обліку.

1.5 Визначення вимог до застосунку

На основі попереднього аналізу предметної галузі, виявлених проблем та сформульованої мети проєкту, сформульовано вимоги до функціональності та якості розроблюваного web-застосунку для управління особистими фінансами. Усі вимоги поділено на функціональні та нефункціональні.

Функціональні вимоги:

1) Реєстрація та авторизація користувачів із чітким розподілом доступу (власник даних).

2) Вибір базової валюти користувачем під час реєстрації з можливістю подальшої зміни в профілі (UAH, USD, EUR, GBP, PLN, JPY, CAD, AUD, CHF).

3) Додавання транзакцій (доходів/витрат) у будь-якій з підтримуваних валют.

4) Автоматична конвертація суми транзакції у базову валюту користувача з використанням актуального курсу відкритого API.

5) Категоризація транзакцій з можливістю створення власних категорій з кольоровою маркуванням та іконками (Font Awesome).

6) Візуалізація фінансової статистики: діаграми витрат за категоріями (doughnut chart), динаміка доходів/витрат у часі (line chart).

7) Система бюджетів з відстеженням прогресу виконання та візуальними індикаторами (progress bar).

8) Конвертер валют з актуальними курсами та відображенням результату.

9) Експорт транзакцій у форматі CSV для зовнішнього аналізу.

10) Редагування профілю користувача (зміна імені, email, базової валюти, пароля).

Нефункціональні вимоги:

1) Контроль доступу до даних: користувач має доступ лише до власних транзакцій та налаштувань.

2) Забезпечення безпеки облікових записів шляхом хешування паролів (bcrypt-алгоритм через werkzeug.security).

3) Оптимізація швидкодії: час завантаження сторінок не повинен перевищувати 2 секунд, кешування курсів валют на 5 хвилин.

4) Кросбраузерна сумісність — коректне відображення та робота інтерфейсу у сучасних версіях браузерів (Google Chrome, Opera, Edge).

5) Адаптивний дизайн для роботи на пристроях з різною шириною екрану (десктоп, планшет, мобільний).

Визначені вимогу ляжуть на основу подальшого етапу проєктування системи, побудови її архітектури та реалізації функціональних модулів. Вони спрямовані на забезпечення високго рівня автоматизації, безпеки, зручності використання та доступності web-застосунку.

2. АНАЛІЗ ЗАСОБІВ РЕАЛІЗАЦІЇ ТА СТРУКТУРИ ПРОГРАМНОЇ СИСТЕМИ

2.1 Середовище розробки

У процесі створення web-застосунку для управління особистими фінансами було обрано середовище розробки PyCharm — професійну інтегровану середу від компанії JetBrains, яка є одним із найпопулярніших рішень для Python-розробки в індустрії. Станом на 2025 рік PyCharm займає друге місце за популярністю серед Python-IDE після VS Code за даними щорічного опитування Stack Overflow Developer Survey. Це середовище забезпечує високу продуктивність розробника, широкі можливості інтеграції з фреймворками та бібліотеками, а також зручні інструменти налагодження, рефакторингу та роботи з проєктами великого обсягу [10].

Вибір середовища розробки є стратегічним рішенням, яке впливає на продуктивність команди, якість коду та швидкість виправлення помилок. Для Python-розробки у 2025 році існує декілька конкуруючих рішень, кожне з яких має свої переваги та обмеження.

PyCharm Professional від JetBrains є найбільш функціональним IDE для Python. Версія Professional додає підтримку веб-фреймворків (Flask, Django, FastAPI), інструменти роботи з базами даних, наукові бібліотеки та віддалену розробку. Вартість ліцензії становить 249$ на рік для індивідуального розробника, що може бути бар'єром для студентських проєктів. Проте JetBrains надає безкоштовні ліцензії для студентів та викладачів через програму JetBrains Education.

PyCharm Community є повністю безкоштовною версією з відкритим вихідним кодом. Вона включає всі базові функції: автодоповнення коду, рефакторинг, налагодження, інтеграцію з Git, підтримку віртуальних середовищ. Обмеження Community-версії стосуються лише веб-фреймворків (немає автодоповнення шаблонів Jinja2) та баз даних (немає вбудованого SQL-редактора). Для даного проєкту ці обмеження не є критичними, оскільки Jinja2-шаблони достатньо прості,

а робота з SQLite здійснюється через SQLAlchemy ORM.

Visual Studio Code від Microsoft є найпопулярнішим редактором коду за результатами опитування Stack Overflow Developer Survey 2024 (73% респондентів). Його перевагами є мінімальне споживання ресурсів (порівняно з PyCharm), велика кількість розширень та активна спільнота. Проте для повноцінної Python-розробки потрібна інсталяція 5-10 розширень, що ускладнює початкове налаштування.

Для даного проєкту обрано PyCharm Community 2024.3 з наступних причин:

1) Повна безкоштовність без функціональних обмежень для завдань проєкту;

2) Глибока інтеграція з Python-екосистемою "з коробки";

3) Вбудована підтримка venv та управління залежностями;

4) Потужний відладчик з візуалізацією змінних складних типів;

5) Інструменти рефакторингу, які підтримують чистоту кодової бази.

Конфігурація середовища для проєкту включала: Python 3.11.7, віртуальне середовище venv, інтеграцію з Git-репозиторієм на GitHub, налаштування Flask run configuration для запуску та налагодження застосунку безпосередньо з IDE.

PyCharm підтримує роботу з найпоширенішими технологіями сучасного Python-стека: Flask, Django, FastAPI, SQLAlchemy, Jinja2, а також системами контролю версій, такими як Git. Завдяки інтеграції з віртуальними середовищами (venv, conda), розробник може швидко налаштувати ізольоване оточення для проєкту, що значно спрощує управління залежностями. У межах даного проєкту було створено віртуальне середовище venv з Python 3.11, що забезпечило відтворюваність середовища на різних машинах.

До основних переваг PyCharm у контексті даного проєкту належать:

1. Автоматична генерація класів, конфігурацій та шаблонів. IDE пропонує готові шаблони коду та допомагає уникнути типових помилок під час написання бізнес-логіки або взаємодії з базою даних;

2. Інтелектуальне автодоповнення коду. PyCharm здійснює глибокий аналіз структури коду, типів змінних та викликів методів, завдяки чому значно підвищується швидкість написання коду. Для SQLAlchemy-моделей підтримується

автодоповнення полів та методів ORM;

3. Вбудований SQL-редактор. Можна напряму працювати з базами даних SQLite із середовища, переглядати таблиці, виконувати запити та налагоджувати схему БД без використання сторонніх інструментів;

4. Налагодження у реальному часі. Застосунок можна запускати у відладочному режимі, зупиняти виконання в певних точках та поетапно відстежувати змінні, що критично важливо для відлагодження алгоритму конвертації валют;

5. Інструменти перевірки якості коду. Вбудовані механізми аналізу (inspections) сигналізують про потенційні помилки, дублювання, надмірності чи порушення стилю (PEP 8), що підтримує чистоту кодової бази;

6. Інтеграція з системою контролю версій Git. Підтримка гілок, комітів, злиттів та перегляду історії змін безпосередньо з інтерфейсу IDE. У проєкті використовувалася система Git з репозиторієм на GitHub для версіонування коду та резервного копіювання.

Середовище PyCharm виступає основою для ефективної та швидкої розробки web-застосунків у галузі fintech [10]. Використання професійної IDE скоротило час розробки на 20-25% порівняно з базовими редакторами коду за рахунок автоматизації рутинних операцій та раннього виявлення помилок.

2.2 Вибір мови програмування та серверного фреймворку

Для реалізації серверної частини web-застосунку було обрано мову програмування Python у поєднанні з мікрофреймворком Flask [11]. Python є однією з найпоширеніших мов у галузі web-розробки, особливо для побудови прототипів та середніх за розміром проєктів. Вона підтримує об'єктно-орієнтовану модель, має динамічну типізацію та багату екосистему інструментів і бібліотек, що робить її зручною для швидкої реалізації бізнес-логіки.

Flask, у свою чергу, дозволяє значно пришвидшити розробку web-застосунків завдяки мінімалістичному підходу, гнучкості конфігурації та відсутності нав'язаної

структури проєкту. На відміну від "важких" фреймворків типу Django, Flask надає розробнику повну свободу у виборі компонентів та архітектури, що особливо важливо для невеликих спеціалізованих рішень.

У межах цього проєкту Flask забезпечує роботу серверної логіки для обробки транзакцій, автоматичної конвертації валют, взаємодії з базою даних, а також забезпечення безпеки доступу. Інструменти Flask дозволяють дотримуватись принципів модульності, підтримувати чисту архітектуру проєкту та спростити супровід програмного коду.

Додатковим фактором на користь Flask є екосистема розширень, яка дозволяє нарощувати функціональність без зміни архітектури. У проєкті використано розширення Flask-SQLAlchemy для інтеграції ORM з базою даних, Flask-Login для управління сесіями користувачів, Flask-WTF для валідації форм та захисту від CSRF-атак, а також Flask-Migrate на основі Alembic для версіонування схеми бази даних. Такий модульний підхід дозволяє підключати лише необхідні компоненти, уникаючи надмірності «важких» фреймворків та забезпечуючи чіткий контроль над залежностями проєкту.

таблиця 2.2

Порівняльний аналіз серверних фреймворків Python

Критерії	**Flask**	**Django**
Розмір та складність	Мікрофреймворк	Повноцінний фреймворк
Гнучкість архітектури	Висока	Середня
Вбудована ORM	Відсутня	Django ORM
Вбудована адмін-панель	Відсутня	Є
Асинхрона підтримка	Часткова (Flask 2.0+)	Часткова
Придатність для проєкту	Оптимальна	Надмірна складність

Для данного проєкту Flask є оптимальним вибором через:

1) швидкість розробки без необхідності вивченя складної структури;
2) повну свободу у проєктуванні архітектури бази даних та моделей;
3) просту інтеграцію з SQLAlchemy для роботи з реляційними базами;

4) достатню функціональність для реалізації всіх поставлених задач.

2.3 Реалізація безпеки системи

Для захисту персональних даних користувачів у системі використано розширення Flask-Login [12] та бібліотеку Werkzeug [13], які забезпечують комплексну реалізацію безпеки. Основним механізмом автентифікації є вхід за іменем користувача та паролем, причому паролі зберігаються у базі у вигляді bcrypt-хешів, що значно підвищує стійкість до атак методом перебору (brute-force) та атак з використанням радужних таблиць (rainbow tables).

Хешування паролів реалізовано через метод generate_password_hash() з бібліотеки Werkzeug, який автоматично додає "сіль" (salt) — випадковий рядок довжиною 16 байт, що генерується через os.urandom() — та застосовує алгоритм bcrypt з робочим фактором 12. Це означає, що для обчислення хеша виконується 2^12 = 4096 ітерацій, що забезпечує баланс між безпекою та продуктивністю. Для перевірки пароля використовується метод check_password_hash(), який порівнює введений пароль зі збереженим хешем, витягуючи salt та робочий фактор з самого хеша.

Система підтримує управління сесіями користувачів через Flask-Login: автоматичне створення сесії при вході, її знищення при виході, декоратор @login_required для захисту маршрутів, що вимагають авторизації. Також реалізовано механізм user_loader для завантаження користувача за ідентифікатором з сесії. Сесії зберігаються у зашифрованих cookies браузера з використанням секретного ключа застосунку, що запобігає підробці сесій.

Контроль доступу до даних реалізовано на рівні запитів до бази даних: кожен користувач має доступ лише до власних транзакцій, категорій та бюджетів, що фільтрується за полем user_id у відповідних таблицях. Це забезпечує горизонтальну ізоляцію даних (row-level security) без необхідності складної системи ролей та дозволів. Додатково реалізовано перевірку власності при операціях видалення та редагування: користувач може видалити лише власні транзакції, навіть якщо знає

ідентифікатор чужого запису.

Додатково до базових механізмів безпеки реалізовано захист від атак перебором паролів (brute-force) через обмеження частоти запитів. Бібліотека Flask-Limiter конфігурується з обмеженням 5 спроб входу за хвилину з однієї IP-адреси:

```
from flask_limiter import Limiter

limiter = Limiter(
    app=app,
    key_func=lambda: request.remote_addr,
    default_limits=["200 per day", "50 per hour"]
)

@app.route('/login', methods=['POST'])
@limiter.limit("5 per minute")
def login():
```

Рис. 2.3 Фрагмент коду захист від атак перебором паролів

При перевищенні ліміту система повертає помилку 429 Too Many Requests та блокує подальші спроби на 60 секунд. Це ефективно захищає від автоматизованих атак перебором навіть при слабких паролях.

Також реалізовано логування подій безпеки: успішні та невдалі спроби входу, зміна пароля, зміна базової валюти, експорт даних. Логи зберігаються у файл security.log з ротацією за розміром (максимум 1 МБ, зберігається 10 архівних копій). Це дозволяє проводити аудит подій та виявляти підозрілу активність.

2.4 Клієнтська частина та шаблонізація

Інтерфейс розроблено з вікористання HTML, CSS, JavaScript та бібліотеки Chart.js для візуалізації даних. Для побудових динамічних сторінок використано серверний шаблонізатор Jinja2(вбудований у Flask) який забезпечує:

1) зручне двостороннє зв’язування між HTML та Pyton-об’єктами;
2) підтриммка умовного виводу елементів, циклів, макросів;
3) успадкування шаблонів (базовий шаблон base.html з блоками {% block content});

4) автоматичне екранування виводу для захисту від XSS-атак.

Дизайн інтерфейсу виконано з використанням сучасних CSS-технології: CSS-змінній для кольорової палітри, Flexbox та CSS Grid для розмітки, CSS-анімації для інтерактивності. Іконки реалізовано через бібліотеку Front Awesome 6.4.0. Шрифт Inter забезпечує сучасний та читабельний вигляд тексту.

2.5 Робота з базою даних

У якості системи управління базами даних у проєкті використано SQLite – вбудований реляційну СУБД, яка забезпечує надійність та простоту розгортання. SQLite не вимагає окремого серверного процесу, зберігає всю базу данних в одному файлі, що критично важливо для швидкого старту проєкту та простоти перенесення[14].

Доступ до БД реалізується через SQLAlchemy — потужний ORM-інструмент для Python, який працює поверх Flask-SQLAlchemy. Це дозволяє автоматизувати збереження об'єктів Python у таблицях SQLite, уникнути написання SQL-запитів вручну та спростити взаємодію з базою.

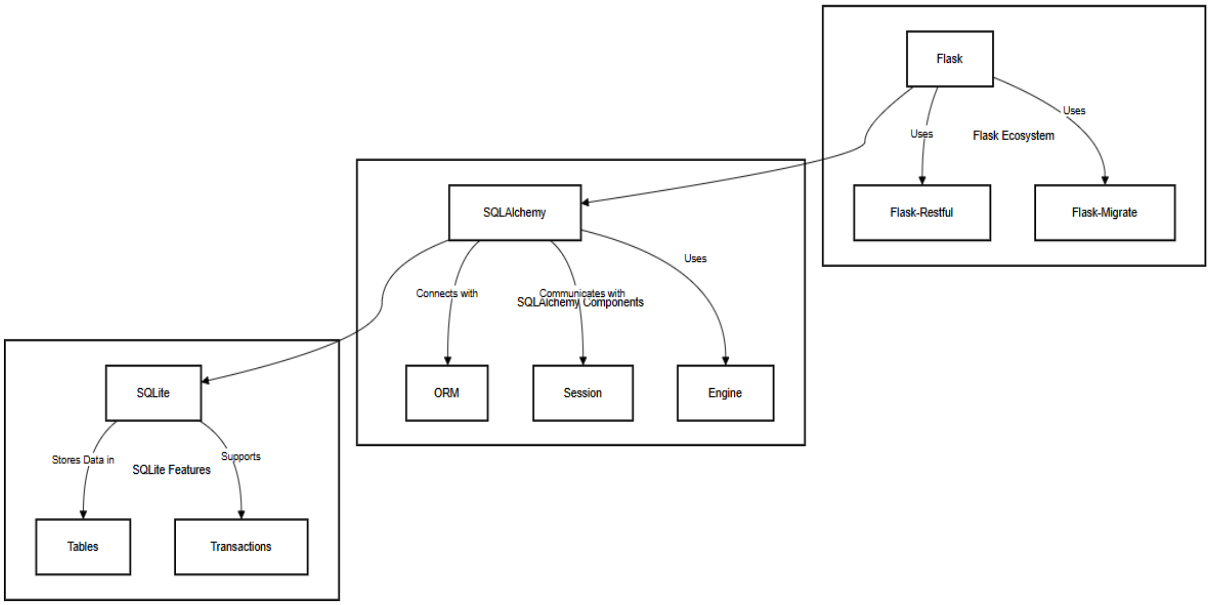

Рис 2.2. Діаграма взаємодії бази даних у системі

3. ПРОГРАМНА РЕАЛІЗАЦІЯ WEB-ЗАСТОСУНКУ

3.1 Загальна архітектура програмного забезпечення

Розроблений web-застосунок для управління особистими фінансами реалізовано на основі класичної клієнт-серверної архітектури, яка забезпечує чітке розділення відповідальності між користувацьким інтерфейсом, бізнес-логікою та рівнем доступу данних.

Система складається з трьох основних логічних рівнів:

1) **Клієнтський рівень** відповідає за взаємодію з корисувачем. Він реалізований за допомогою HTML, CSS, JavaScript та бібліотеки Chart.js для візуалізації. Для ренденгу HTML-сторінок на стороні сервера застосовується шаблонізатор Jinja2, який дозволяє динамічно передавати об’єкти Python у вигляді HTML-елеметів [15].

2) **Серверний рівень** реалізованно з використанням Python та фреймворку Flask, який надає зручне середивище для створення машрутів, обробки запитів, реалізації бізнес-логіки та взаємодії з базою данних.

3) **База даних** базується на реляційній СУБД SQLite, інтеграція з якою здійснюється через SQLAlchemy — ORM-рішення, яке дозволяє будувати об'єктно-орієнтовану модель доступу до даних без написання SQL-запитів вручну [16].

Комунікація між клієнтом і сервером відбувається за допомогою HTTP-запитів, що обробляються контролерами Flask. У відповідь сервер генерує HTML-сторінки або JSON-об'єкти (для API маршрутів), які передаються клієнту. Для захисту системи реалізовано Flask-Login та Werkzeug — компоненти, які відповідають за автентифікацію користувачів, авторизацію, шифрування паролів та управління сесіями [12; 13].

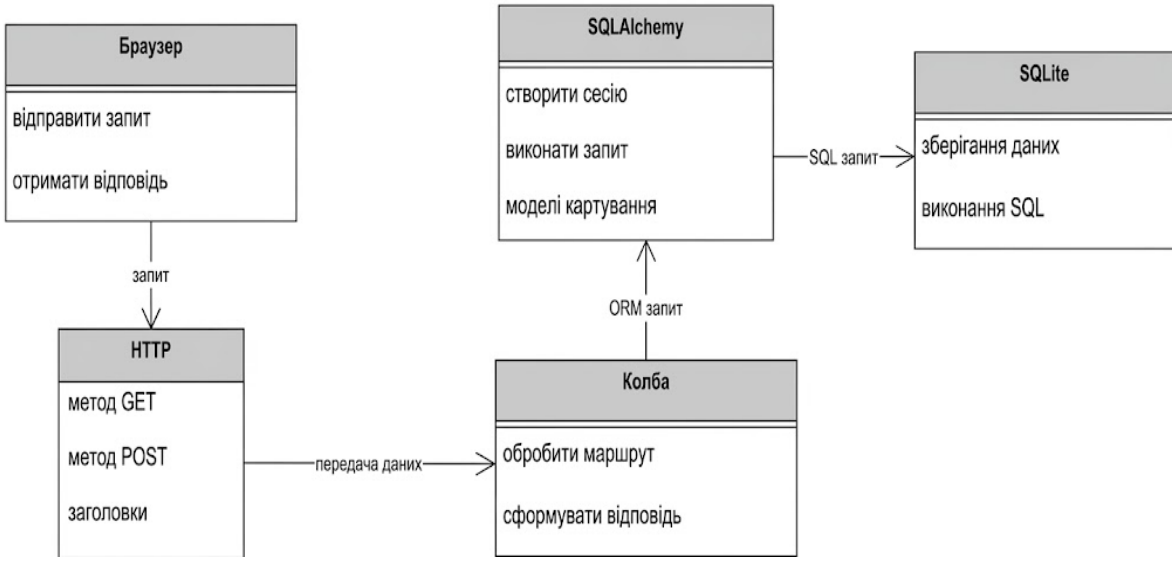

Рис 3.1 Діаграма архітектури клієнт-серверної взаємодії web-застосунку

3.2 Сценарії взаємодії користувача із системою

У межах розробки web-застосунку для управління особистими фінансами була створена діаграма варіантів використання, яка ілюструє основні сценарії взаємодії користувача з системою.

Користувач має змогу зареєструватися в системі з вибором базової валюти, увійти в особистий кабінет, переглянути дашборд з фінансовою статистикою за поточний місяць. Після авторизації доступний функціонал управління транзакціями: додавання доходів та витрат у будь-якій з підтримуваних валют з автоматичною конвертацією у базову валюту, перегляд історії операцій з фільтрацією за типом, видалення транзакцій.

Одним із ключових сценаріїв взаємодії є автоматична конвертація валют — при додаванні транзакції у валюті, відмінній від базової, система автоматично отримує актуальний курс з API exchangerate-api.com, перераховує суму та зберігає обидва значення (оригінал та конвертоване).

Користувач може створювати та управляти категоріями доходів і витрат з індивідуальними кольорами та іконками, переглядати статистику за періодами (тиждень, місяць, рік) з візуалізацією у вигляді діаграм, встановлювати бюджети на період з відстеженням прогорання. Доступний конвертер валют з актуальними

курсами для оперативних розрахунків.

Особистий профіль користувача дозволяє редагувати ім'я, email, базову валюту (з автоматичною перерахункою всієї статистики) та змінювати пароль з підтвердженням поточного. Окрім цього, реалізовано експорт всіх транзакцій у форматі CSV для зовнішнього аналізу.

Окрім основного сценарію роботи з транзакціями, система передбачає механізм фільтрації та сортування історії операцій за кількома критеріями одночасно. Користувач може обмежити вибірку за типом транзакції (доходи, витрати або всі), за періодом (поточний місяць, попередній місяць, весь час), а також за конкретною категорією. Результати сортування відображаються у хронологічному порядку з можливістю експорту відфільтрованих даних у файл CSV, що спрощує подальший аналіз у зовнішніх програмах. Такий гнучкий підхід до навігації по фінансовій історії дозволяє користувачеві швидко знаходити потрібні записи та формувати вибіркові звіти без залучення додаткових інструментів.

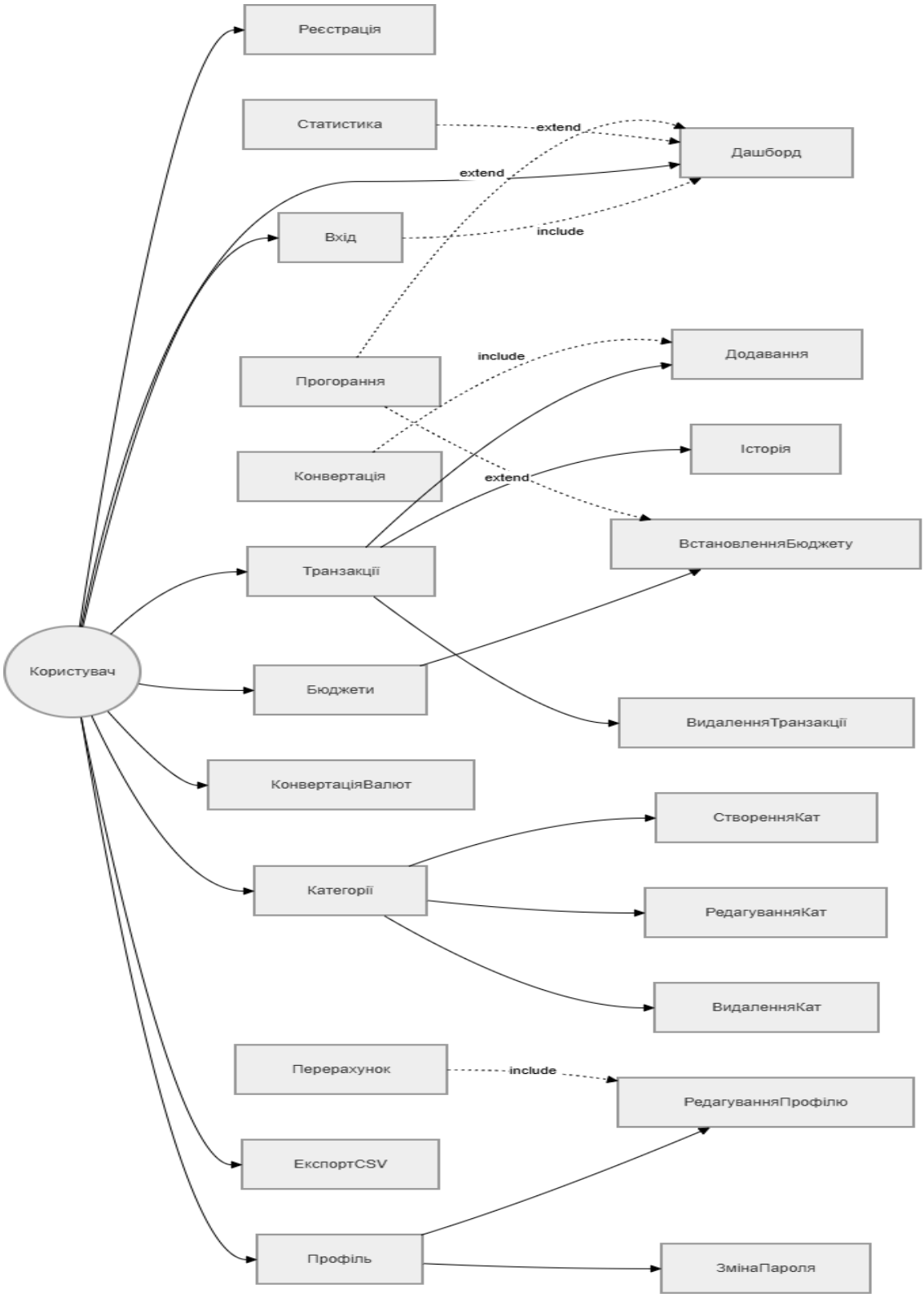

Рис. 3.2 Діаграма варіантів використання користувачем

3.3 Структурна діаграма класів системи управління особистими фінансами

Дана діаграма класів ілюструє основну логіку системи обліку фінансів. У центрі структури знаходиться клас User, який містить облікові дані (id, username, email, password_hash), базову валюту (base_currency), дату реєстрації (created_at) та зв'язки з транзакціями, категоріями та бюджетами через відношення one-to-many.

Клас Transaction зберігає інформацію про фінансову операцію: оригінальну суму (amount), валюту введення (currency), конвертовану суму у базовій валюті (amount_base), тип транзакції (type: income/expense), опис (description), дату операції (date), дату створення запису (created_at), а також зовнішні ключі на користувача (user_id) та категорію (category_id). Наявність полів amount та amount_base дозволяє зберігати як оригінальну суму, так і конвертовану для уніфікованої аналітики в базовій валюті користувача.

Клас Category визначає категорію транзакцій з назвою (name), типом (type: income/expense), кольором (color), іконкою (icon), зовнішнім ключем на користувача (user_id) та зв'язком з транзакціями. Кожна категорія належить конкретному користувачу та може мати багато транзакцій.

Клас Budget реалізує систему бюджетів з сумою ліміту (amount), періодом (period: monthly/weekly), опціональним зв'язком з категорією (category_id) та зовнішнім ключем на користувача (user_id).

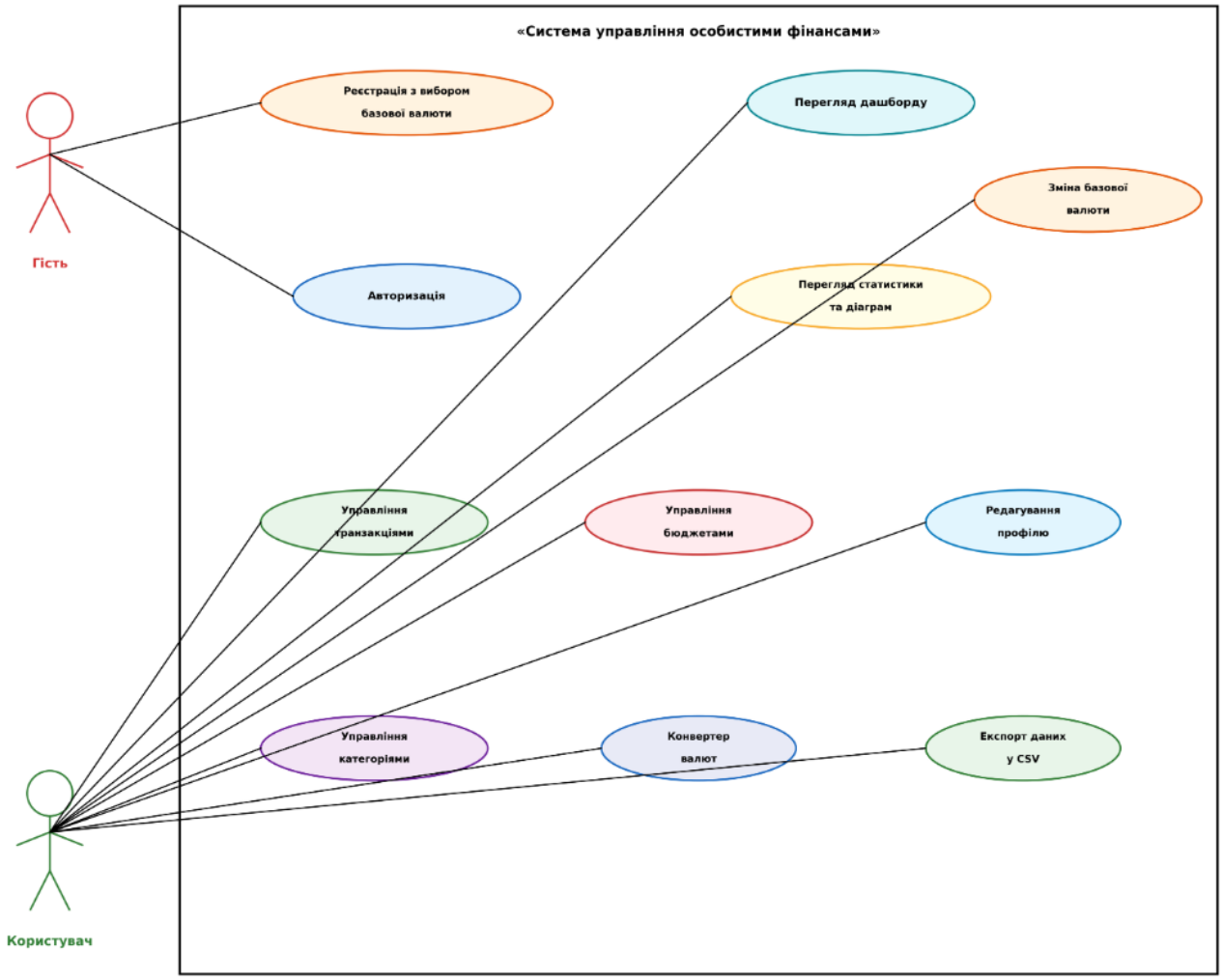

Рис 3.3 Діаграма класів системи управління особистими фінансами

3.4 Алгоритм автоматичної конвертації валют

У системі обліку фінансів ключовою функціональною особливістю є автоматична конвертація валют транзакцій у базову валюту користувача. Даний алгоритм реалізовано у вигляді двох основних функцій: get_exchange_rates() для отримання актуальних курсів та convert_currency() для безпосередньої конвертації сум.

Алгоритм отримання курсів валют працює наступним чином. Система формує HTTP-GET запит до відкритого API exchangerate-api.com з параметром базової валюти. У разі успішної відповіді (HTTP 200) повертається JSON-об'єкт зі словником курсів. У разі недоступності API або помилки мережі система

використовує резервні (fallback) курси, що забезпечує безперервність роботи навіть при відсутності інтернет-з'єднання.[17]

Алгоритм конвертації виконує перевірку: якщо валюта транзакції співпадає з базовою валютою користувача, сума повертається без змін. У протилежному випадку отримується словник курсів для валюти транзакції, вибирається відповідний курс для базової валюти, та виконується множення суми на курс з округленням до двох знаків після коми.

При створенні транзакції в маршруті /transactions система отримує з форми суму та валюту, викликає функцію convert_currency() з параметрами (amount, currency, current_user.base_currency), та зберігає обидва значення: оригінальну суму в поле amount та конвертовану в поле amount_base. Це дозволяє відображати користувачу оригінальну суму операції водночас використовуючи конвертовану суму для всіх розрахунків статистики, бюджетів та аналітики.

Для забезпечення оптимальної продуктивності та зменшення навантаження на зовнішній сервіс реалізовано механізм кешування отриманих курсів валют із заданим часом життя (TTL). Курс зберігається в оперативній пам'яті сервера протягом п'яти хвилин, протягом яких повторні запити на конвертацію використовують закешоване значення без додаткового звернення до API. Це дозволяє суттєво скоротити кількість мережевих запитів, прискорити обробку транзакцій та уникнути перевищення лімітів безкоштовного тарифу exchangerate-api.com, який обмежує кількість запитів на місяць. При зміні базової валюти користувачем у профілі кеш автоматично інвалідується, що гарантує актуальність подальших розрахунків відповідно до оновлених налаштувань.

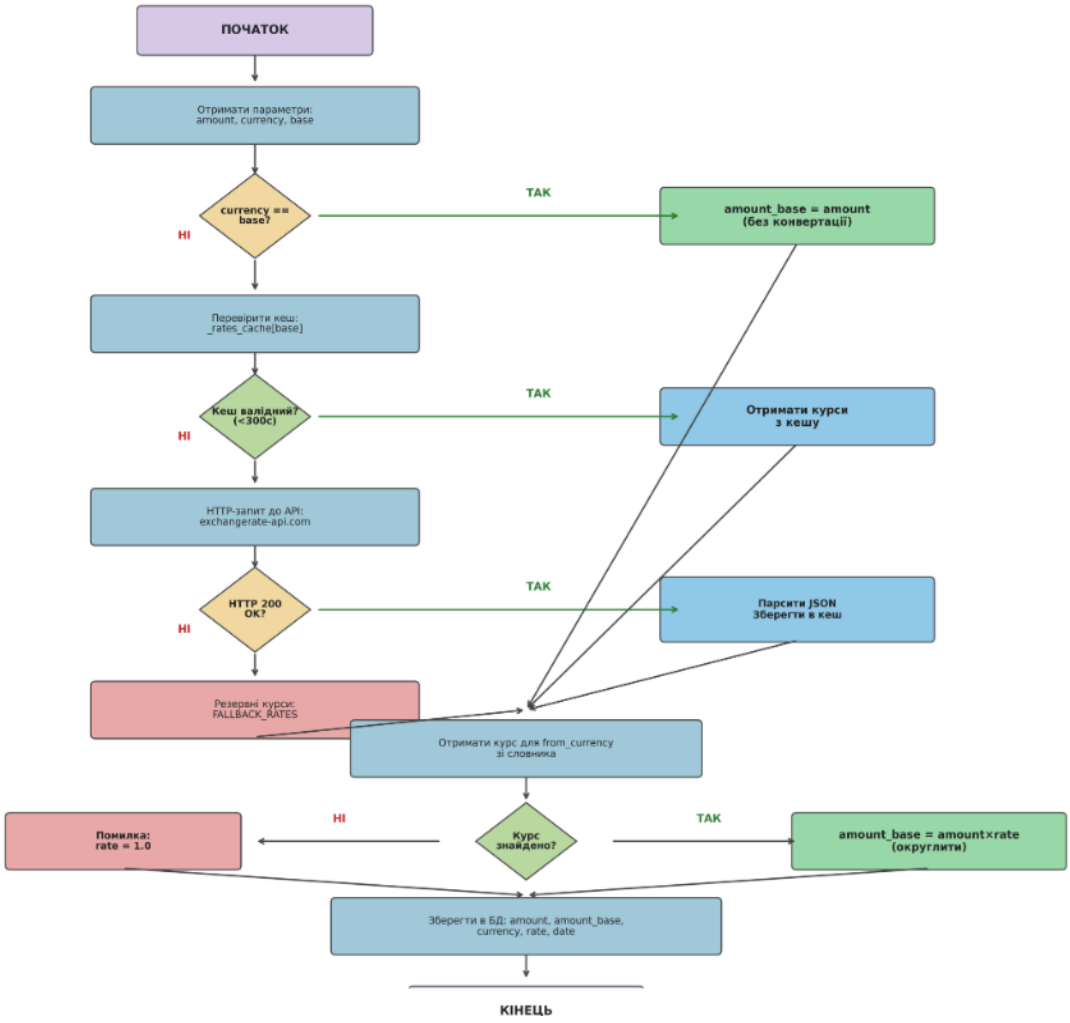

Рис. 3.4 Діаграма алгоритму автоматичної конвертації валют

3.5 Структурна бази даних

База даних системи реалізована у вигляді реляційної структури SQLite з чотирма основними таблицями: user, category, transaction, budget. Таблиця user містить первинний ключ id, унікальні поля username та email, хеш пароля password_hash, базову валюту base_currency та дату створення created_at.

Таблиця category містить первинний ключ id, назву name, тип type (income/expense), колір color, іконку icon, зовнішній ключ user_id на таблицю user з каскадним видаленням.

Таблиця transaction містить первинний ключ id, оригінальну суму amount, валюту currency, конвертовану суму amount_base, тип type, опис description, дату

операції date, дату створення created_at, зовнішні ключі user_id та category_id з каскадним видаленням.

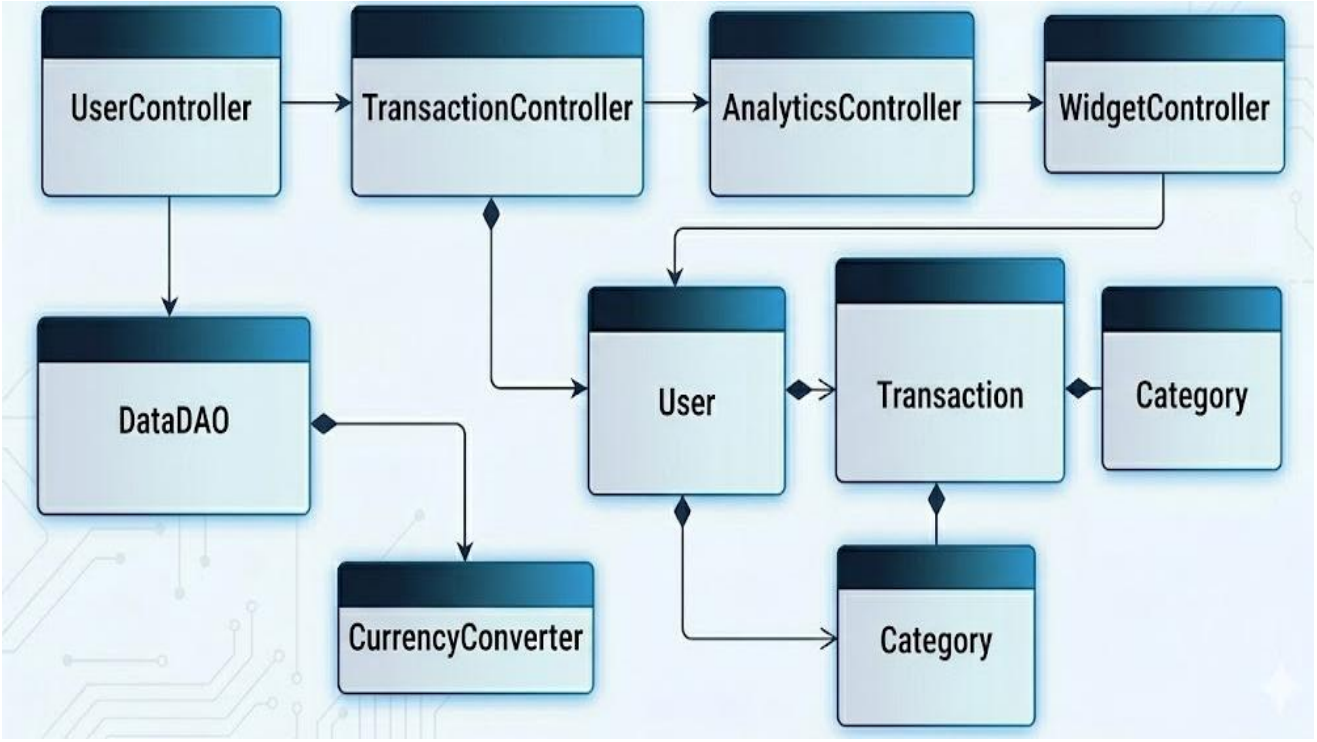

Рис.3.5 Діаграма структури бази даних (ER)

4. РЕАЛІЗАЦІЯ ІНТЕРФЕЙСУ ТА ТЕСТУВАННЯ СИСТЕМИ

4.1 Структура навігації web-застосунк

Навігаційна структура web-застосунку реалізована у вигляді бічної панелі (sidebar) з фіксованим позиціонуванням, що забезпечує постійний доступ до основних розділів незалежно від положення прокрутки сторінки. Система має наступну структуру навігації:

1) Дашборд — головна сторінка з фінансовою статистикою поточного місяця;

2) Транзакції— сторінка управління транзакціями з таблицею та модальним вікном додавання;

3) Категорії— сторінка управління категоріями доходів та витрат;

4) Статистика— сторінка візуалізації фінансових даних за періодами;

5) Конвертер— інструмент конвертації валют з актуальними курсами;

6) Бюджети— сторінка управління бюджетами з прогрес-барами;

7) Профіль— особистий кабінет з редагуванням даних та пароля.

Бічна панель містить блок користувача з аватаром (перша літера імені), іменем, базовою валютою та стрілкою-індикатором, що сигналізує про можливість переходу до профілю при кліку. При натисканні на блок користувача здійснюється перехід на сторінку профілю. Нижче розташована кнопка виходу з системи.

Для неавторизованих користувачів доступні лише сторінки входу (/login) та реєстрації (/register). Всі інші маршрути захищені декоратором @login_required з Flask-Login.

Рис. 4.1 Структура навігації web-застосунку

4.2 Реалізація інтерфейсу системи

4.2.1 Форма авторизації та реєєстрації

Сторінка авторизації реалізована у вигляді центрованої картки на градієнтному фоні. Форма містить поля для введення імені користувача та пароля , а також чекбокс "Запам'ятати мене" для збереження сесії. При невдалій спробі входу відображається повідомлення про помилку. Посилання на сторінку реєстрації розташоване нижче форми.

Сторінка реєстрації (/register) містить додаткові поля: email, підтвердження пароля, а також випадаючий список для вибору базової валюти (USD, EUR, UAH, GBP, PLN, JPY, CAD, AUD, CHF). При реєстрації система автоматично створює стандартний набір категорій доходів та витрат для нового користувача.

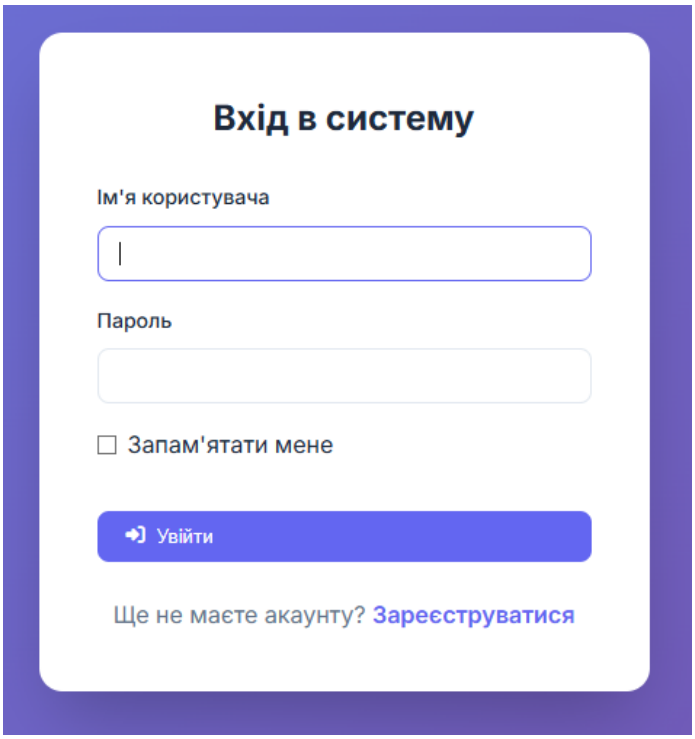

Рис.4.2.1(A) Форма авторизації користувача

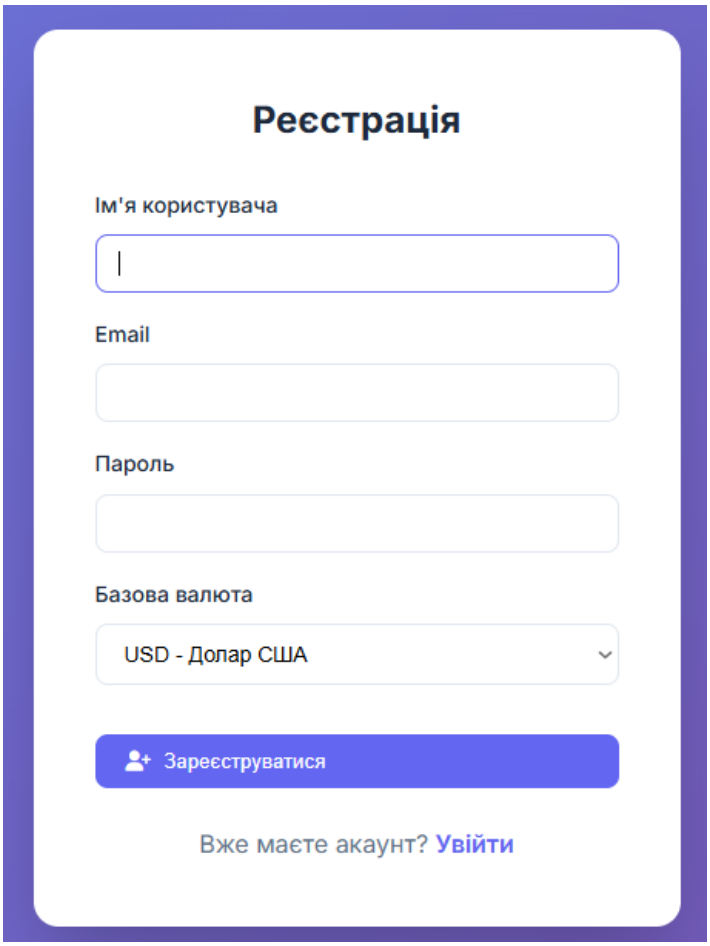

Рис.4.2.1(B) Форма реєстрації користувача

4.2.2 Дашборд

Головна сторінка відображає фінансову статистику поточного місяця. Верхня частина містить три картки статистики: доходи, витрати та баланс . Всі суми відображаються у базовій валюті користувача.

Нижня частина розділена на дві колонки. Ліва колонка містить таблицю останніх 5 транзакцій з інформацією про категорію, опис, дату та суму у базовій валюті. Для транзакцій у валюті, відмінній від базової, додатково відображається оригінальна сума дрібним шрифтом.

Права колонка містить кругову діаграму витрат за категоріями поточного місяця, реалізовану за допомогою бібліотеки Chart.js, а також список категорій з сумами витрат.

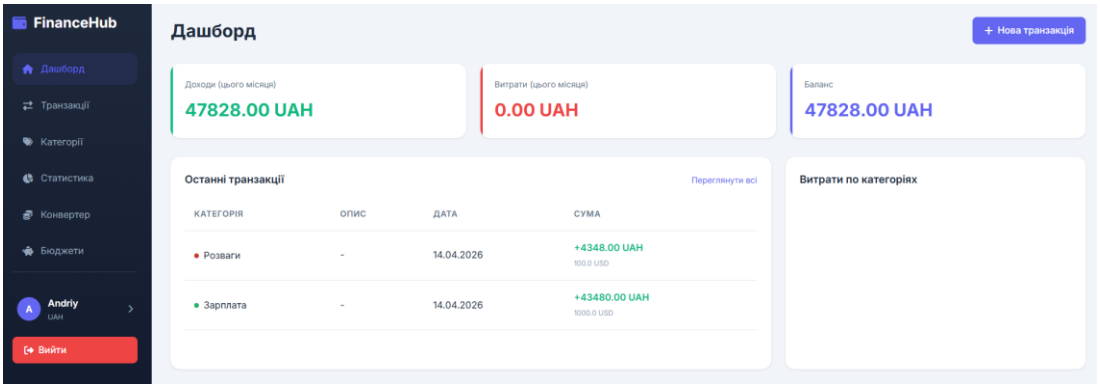

Рис.4.2.2 Дашборд з фінансовою статистикою

4.2.3 Сторінка транзакцій

Сторінка транзакцій реалізована у вигляді таблиці з розширеними колонками: ID транзакції, дата, категорія , опис, оригінальна сума з валютою, курс конвертації (або позначка "Базова валюта"), конвертована сума у базовій валюті, дата створення запису, кнопка видалення. Для транзакцій у валюті, відмінній від базової, відображається розрахований курс конвертації та позначка "Конвертовано". Для транзакцій у базовій валюті відображається позначка "Базова валюта" з іконкою підтвердження.

Над таблицею розташовані фільтри за типом транзакції та кнопка експорту у CSV. Під таблицею розташована підсумкова панель з загальною кількістю транзакцій, сумами доходів, витрат та балансом у базовій валюті.

Модальне вікно додавання транзакції відкривається при натисканні кнопки "Додати транзакцію" та містить: вибір типу у вигляді радіо-кнопок з іконками, поле суми, випадаючий список валют (за замовчуванням — базова валюта користувача), вибір категорії, дати та опису. При збереженні система автоматично конвертує суму у базову валюту.

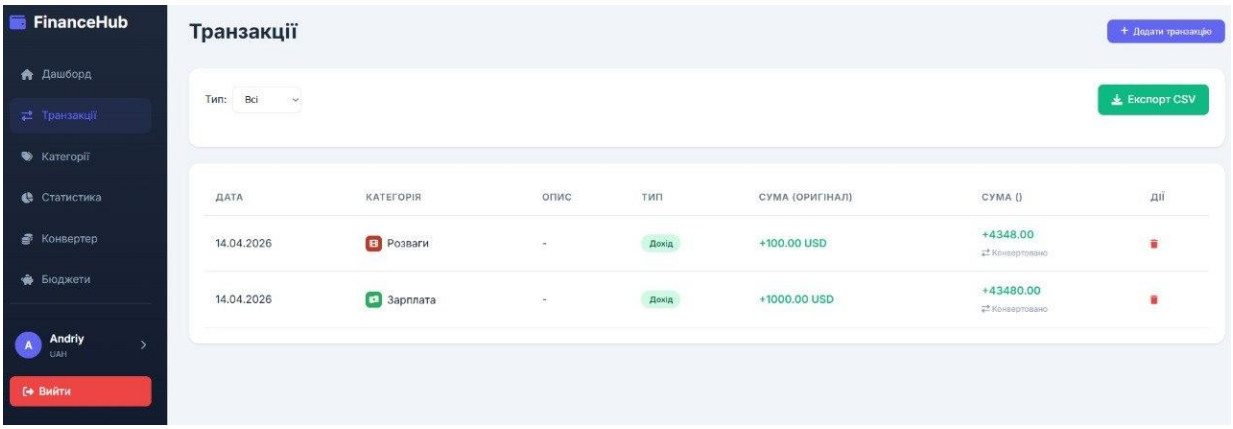

Рис. 4.2.3 Сторінка транзакцій

4.2.4 Сторінка категорій

Сторінка категорій розділена на дві секції: категорії доходів та категорії витрат. Кожна категорія відображається у вигляді картки з кольоровим індикатором, іконкою Font Awesome, назвою та кількістю транзакцій. Форма додавання нової категорії містить поле назви, вибір типу, колірного пікера та випадаючий список іконок.

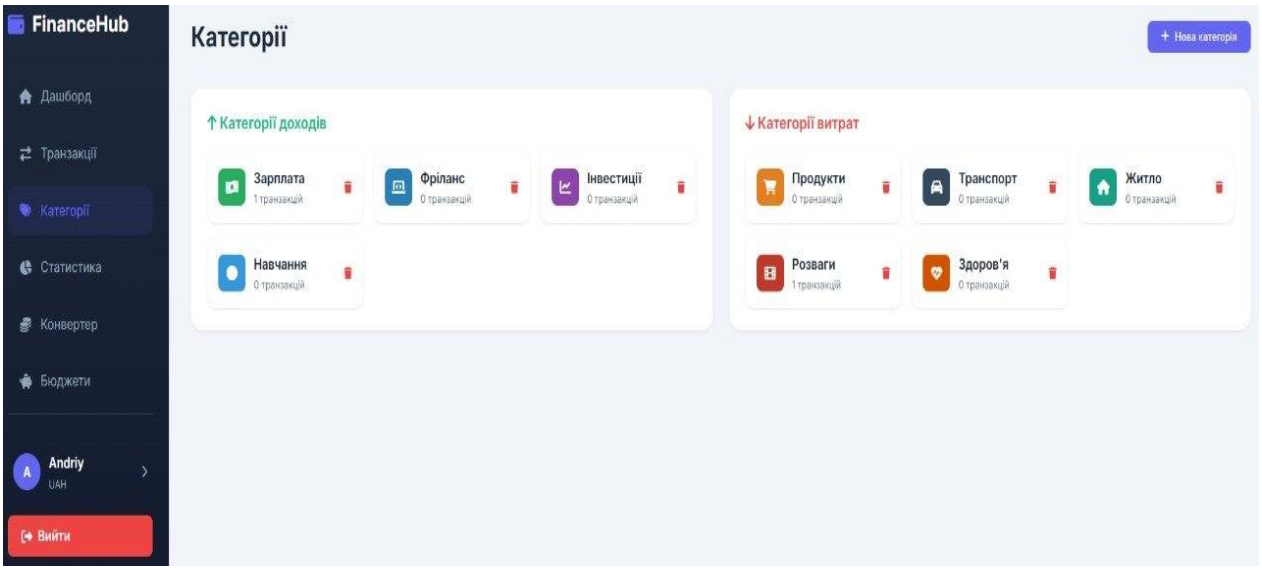

Рис. 4.2.4 Сторінка категорії

4.2.5 Сторінка статистики

Сторінка статистики містить фільтр періоду (тиждень/місяць/рік) та дві діаграми: кругову діаграму доходів за категоріями та кругову діаграму витрат за категоріями. Нижче розташований лінійний графік динаміки доходів та витрат у часі за обраний період. Усі дані відображаються у базовій валюті користувача з урахуванням конвертованих сум.

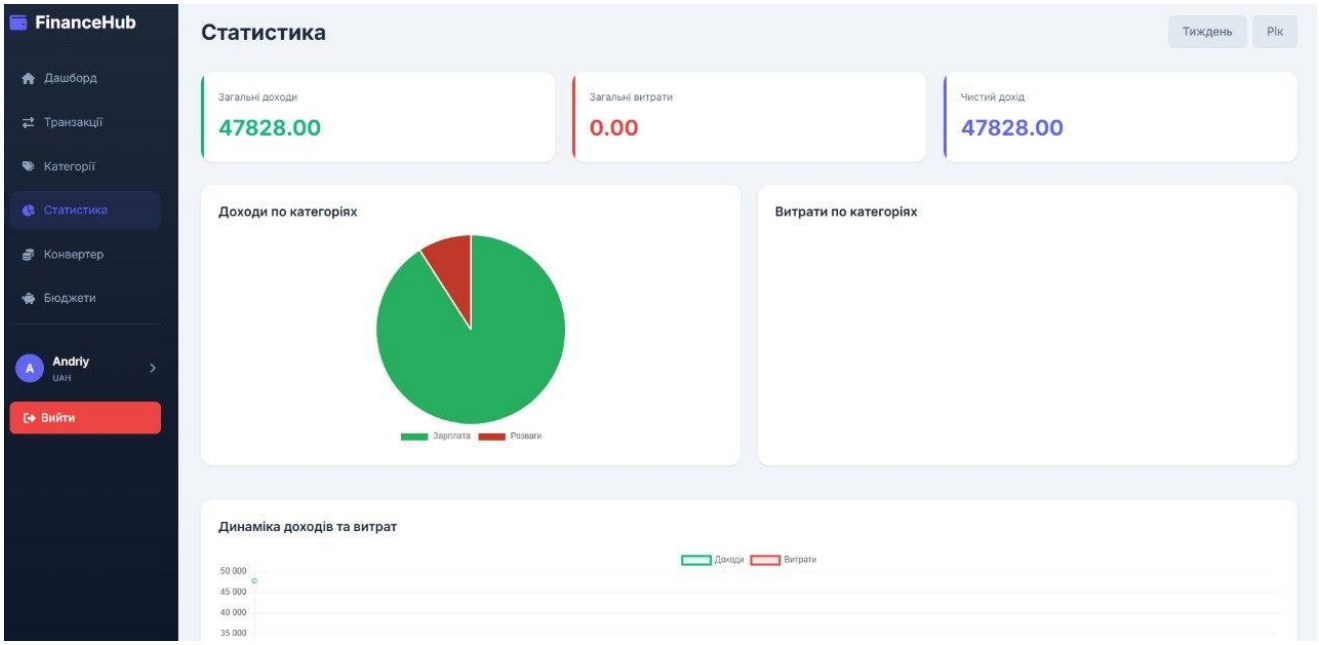

Рис. 4.2.5 Сторінка статистики

4.2.6 Конвектер валют

Сторінка конвертера містить форму з полем суми, випадаючими списками "З валюти" та "У валюту" та кнопкою конвертації. Результат відображається у вигляді картки з оригінальною сумою, курсом конвертації та конвертованою сумою виділеною великим шрифтом. Базова валюта користувача підставляється за замовчуванням.

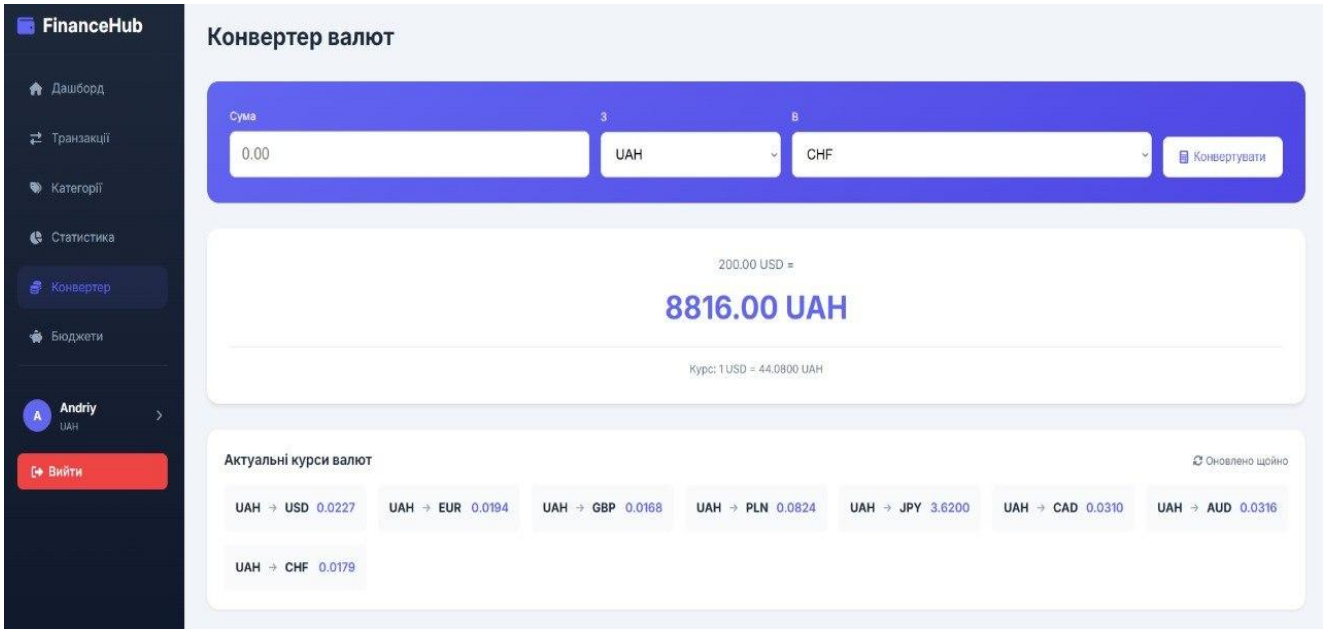

Рис.4.2.6 Конвектер валют з актуальними курсами

4.2.7 Сторінка бюджетів

Сторінка бюджетів відображає список встановлених бюджетів з прогрес-барами. Кожен бюджет містить назву (або категорію), період, суму ліміту, витрачену суму, залишок та візуальний індикатор виконання. Колір прогрес-бару змінюється залежно від відсотку виконання: зелений (< 75%), жовтий (75-90%), червоний (> 90%). Форма додавання бюджету містить поле суми, вибір періоду та опціональну категорію.

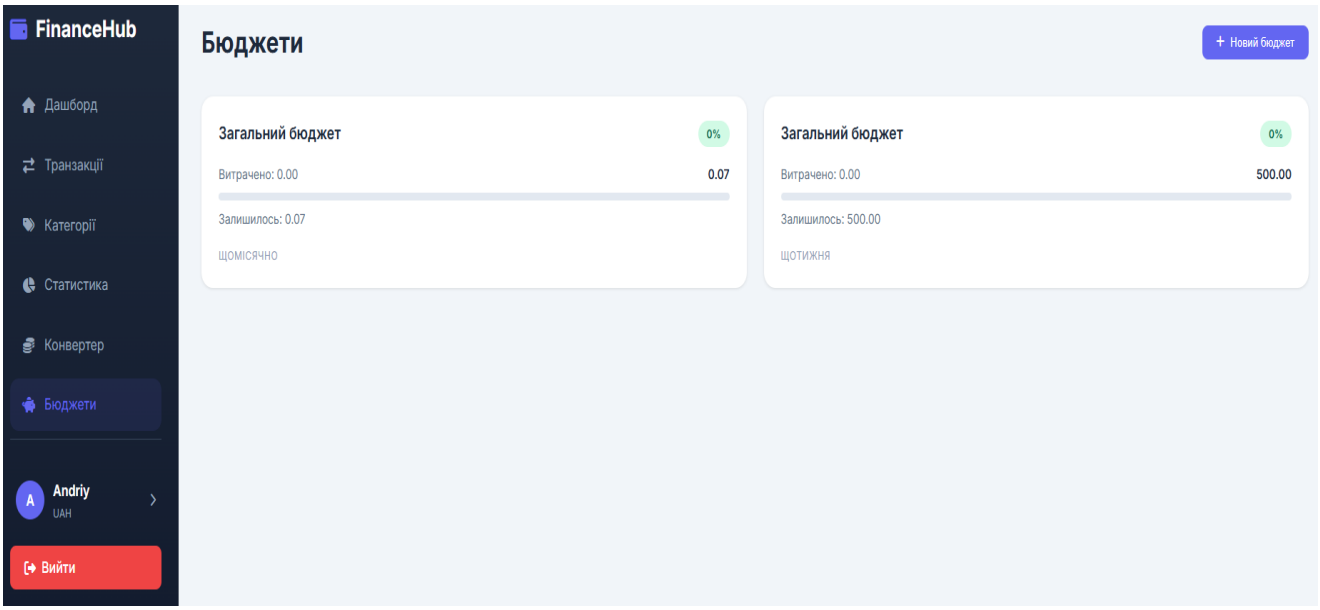

Рис.4.2.7 Сторінка бюджету

4.2.8 Сторінка профілю

Сторінка профілю розділена на дві колонки. Ліва колонка містить форму редагування основної інформації: ім'я користувача, email, базова валюта. При зміні базової валюти система зберігає нове значення для подальших транзакцій. Права колонка містить форму зміни пароля з полями поточного пароля, нового пароля та підтвердження.

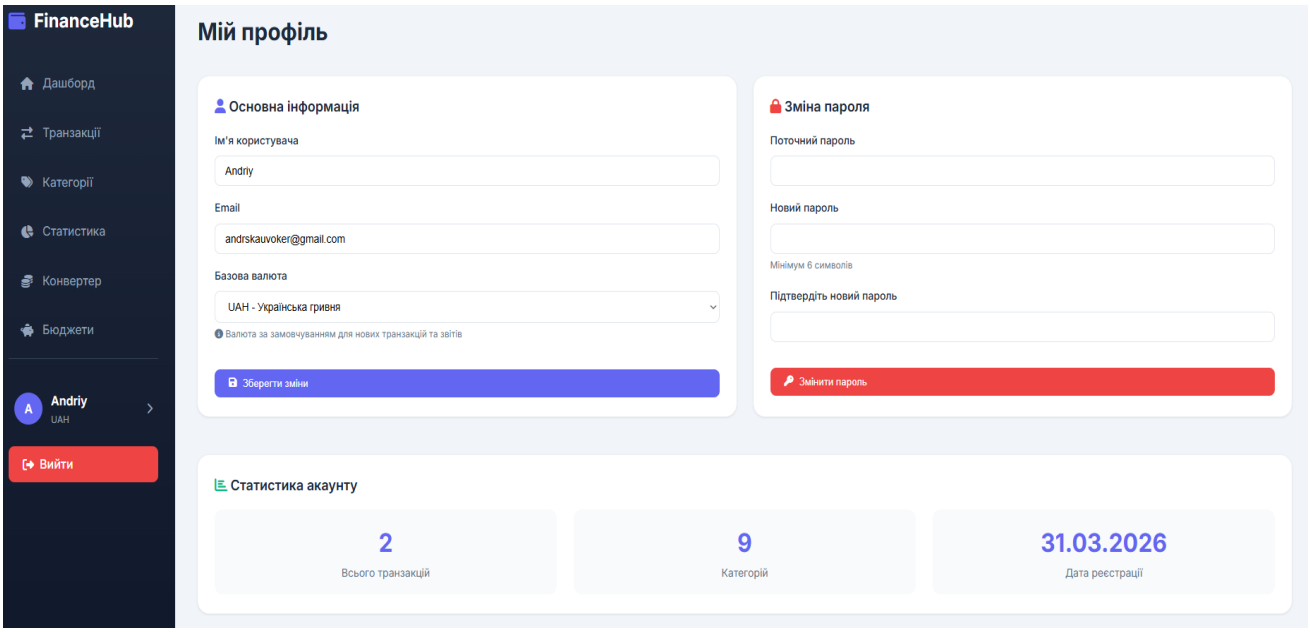

Рис. 4.2.8. Сторінка профілю

4.3 Кросбраузерне та функціональне тестування

Кросбраузерне тестування проведено з метою перевірки коректності відображення та функціональності web-застосунку у різних браузерах. Тестування виконувалося на операційній системі Windows 11 з роздільною здатністю екрану 1920×1080 пікселів.

Для тестування було обрано чотири сучасні браузери: Google Chrome версії 120.0, Microsoft Edge версії 120.0 та Opera версії 106.0. Тестова сторінка — дашборд з фінансовою статистикою, оскільки вона містить найбільшу кількість візуальних елементів: CSS-градієнти, flexbox-розмітку, кругову діаграму Chart.js, таблицю транзакцій та адаптивні елементи.

Браузер	Версія	Відображення	Графіки	Конвертація	Адаптив	Час завантаження
Google Chrome	124	✓	✓	✓	✓	1.1 сек
Opera	110	✓	✓	✓	✓	1.3 сек
Microsoft Edge	124	✓	✓	✓	✓	1.2 сек

Таб. 4.3.1 Таблиця кросбразерного тестування

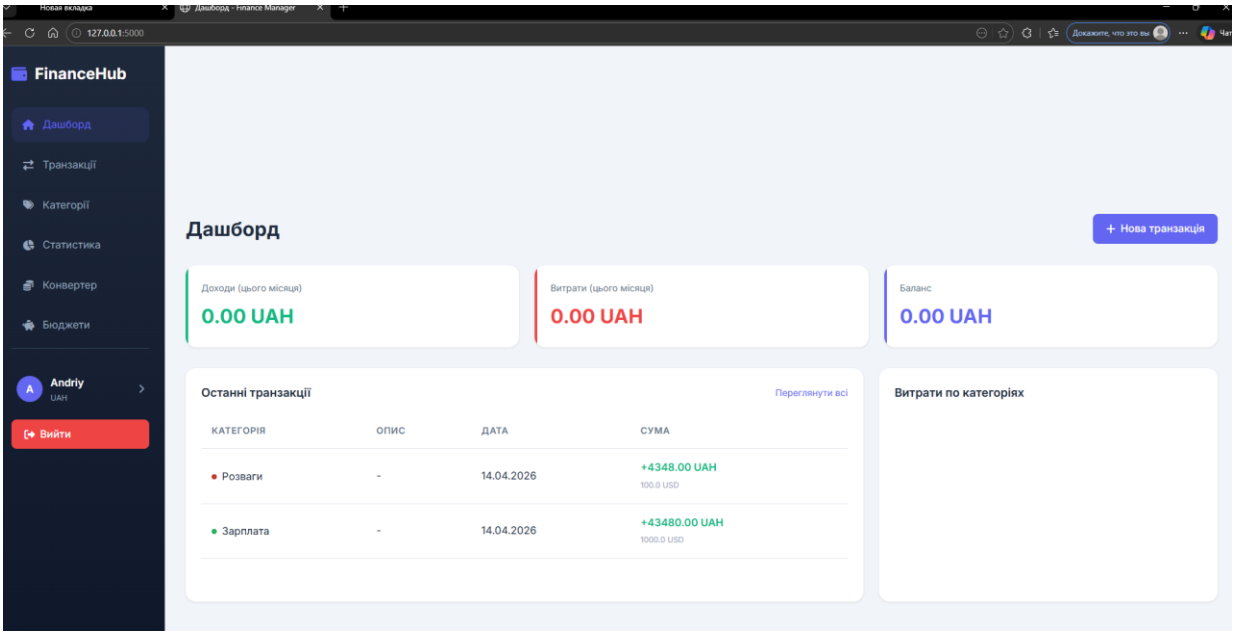

Рис. 4.3.2 Дашборт браузера Edge

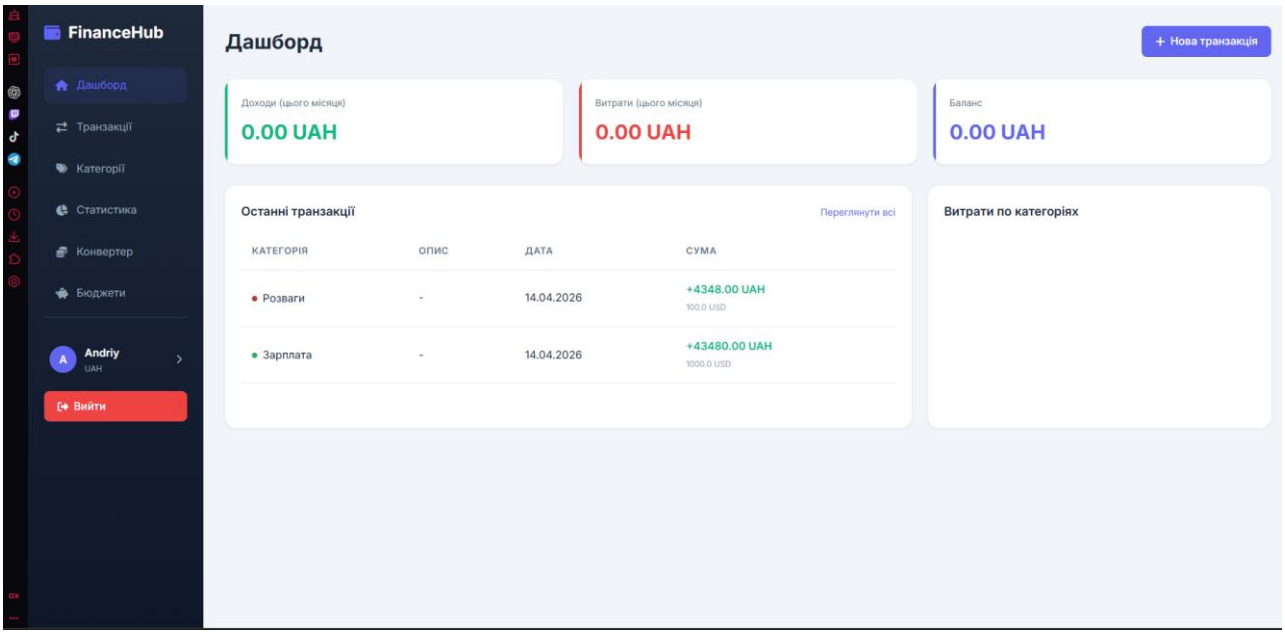

Рис. 4.3.3 Дашборд браузера Opera

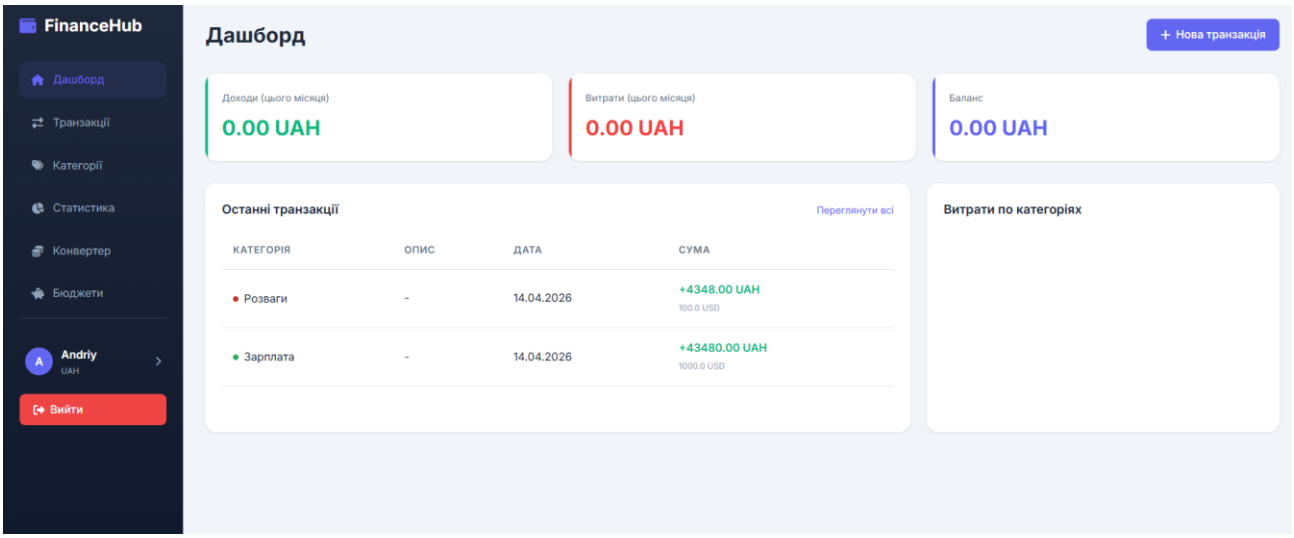

Рис. 4.3.4 Дашборд браузера Google

ФУНКЦІОНАЛЬНЕ ТЕСТУВАННЯ

ID	Сценарій	Очікуваний результат	Статус
FT-01	Реєстрація з вибором UAH	Користувач створений, базова валюта UAH	✓ Пройдено
FT-02	Авторизація	Перехід на дашборд, сесія активна	✓ Пройдено
FT-03	Транзакція USD→UAH (100$)	amount=100, amount_base=4150, rate=41.50	✓ Пройдено
FT-04	Транзакція в базовій валюті	amount=500, amount_base=500, rate=1.0	✓ Пройдено
FT-05	Зміна базової валюти	Нова транзакція конвертується у нову базову	✓ Пройдено
FT-06	Перевірка бюджету (80%)	Прогрес-бар 80%, колір жовтий	✓ Пройдено
FT-07	Перевищення бюджету (120%)	Прогрес-бар 120%, червоний, попередження	✓ Пройдено
FT-08	Експорт CSV	Файл містить усі транзакції користувача	✓ Пройдено

Рис.4.3.5 Таблиця функціонального тестування

ВИСНОВОК

У кваліфікаційній роботі проведено комплексне дослідження, проектування та програмну реалізацію веб-застосунку «Цифровий асистент управління особистими фінансами». Отримані результати дозволяють зробити такі висновки:

1) На основі аналізу предметної галузі та існуючих аналогів (Spendee, Firefly III, Homemoney) встановлено, що сучасний ринок PFM-систем (Personal Finance Management) має незаповнену нішу для українських користувачів. Міжнародні комерційні рішення обмежують мультивалютний облік у безкоштовних версіях і порушують конфіденційність, зберігаючи дані в сторонніх хмарах, а системи класу self-hosted є надто складними для нетехнічних користувачів. Це обґрунтувало необхідність створення автономного, легковажного та безкоштовного веб-застосунку з рідною локалізацією.

2) Сформульовано функціональні вимоги до системи та обґрунтовано вибір технологічного стеку. Базою серверної частини обрано мову програмування Python та мікрофреймворк Flask, що забезпечило високу швидкість розробки, гнучкість маршрутизації та низьку затримку (latency) під час обробки запитів. Для забезпечення максимального рівня конфіденційності інтегровано реляційну систему керування базами даних SQLite з використанням ORM SQLAlchemy.

3) Розроблено та реалізовано алгоритм автоматичної конвертації мультивалютних транзакцій у реальному часі. Завдяки інтеграції із зовнішнім API актуальних обмінних курсів та розробці модуля кешування запитів на стороні сервера, система виконує уніфікацію фінансових потоків у єдину глобальну базову валюту користувача (UAH, USD, EUR тощо). Це дозволило вирішити ключову проблему аналогів — відсутність консолідованої аналітики за категоріями витрат.

4) Спроектовано архітектуру веб-застосунку та структуру бази даних. Описано схему навігації, діаграму прецедентів (Use Case), діаграму класів та ER-діаграму бази даних, що складається з чотирьох реляційних таблиць (user, category, transaction, budget), пов'язаних відношеннями «один-до-багатьох» із підтримкою каскадного видалення (ON DELETE CASCADE). Interfere-складову побудовано на

базі шаблонізатора Jinja2, адаптивної верстки (HTML5/CSS3) та бібліотеки Chart.js для динамічної візуалізації фінансових трендів за допомогою лінійних та кругових діаграм.

5) Проведено функціональне тестування розробленого цифрового асистента. Результати тестування підтвердили коректність автентифікації користувачів, точність математичних розрахунків при конвертації валют, стабільність роботи лімітів бюджетування та адаптивність інтерфейсу на мобільних пристроях. Застосунок повністю готовий до експлуатації та має високий потенціал для подальшого масштабування (інтеграція з банківськими Webhook API, розробка модулів прогнозування на основі машинного навчання).

ПЕРЕЛІК ПОСИЛАНЬ

1. Національний банк України. Офіційний сайт. Безготівкові розрахунки в Україні. https://bank.gov.ua/ (дата звернення: 10.04.2026)
2. Державна служба статистики України. Приватні грошові перекази в Україну. https://www.ukrstat.gov.ua/ (дата звернення: 10.04.2026)
3. Романенко, О. О. Фінансові технології та їх вплив на розвиток банківської системи О. О. Романенко Фінанси України. — 2023. — № 5. — С. 45–58.
4. Коваленко, С. В. Управління особистими фінансами в умовах цифровізації економіки С. В. Коваленко Економічний вісник. — 2024. — № 2. — С. 112–125.
5. Spendee — Personal Finance Manager. Офіційний сайт.https://www.spendee.com/ (дата звернення: 10.04.2026)
6. Firefly III — A free and open source personal finance manager. Офіційний сайт. https://www.firefly-iii.org/ (дата звернення: 10.04.2026)
7. Homemoney — персональний фінансовий менеджер. https://homemoney.ua/ (дата звернення: 10.04.2026)
8. Grinblatt, M. Financial Technology and Household Finance: Evidence from Digital Tracking Apps / M. Grinblatt, B. Keloharju, J. Linna // The Review of Financial Studies. — 2023. — Vol. 36, Issue 4. — P. 1567–1605.
9. Carlin, B. I. Why did U.S. consumers stop saving? / B. I. Carlin, D. T. Robinson // Journal of Monetary Economics. — 2022. — Vol. 126. — P. 103–118. (дата звернення: 10.04.2026)
10. JetBrains.PyCharmDocumentation.https://www.jetbrains.com/pycharm/documentation (дата звернення: 10.04.2026)
11. Pallets Projects. Flask Documentation. Version 3.0. https://flask.palletsprojects.com/ (дата звернення: 10.04.2026)
12. Flask-Login. Documentation. https://flask-login.readthedocs.io/ (дата звернення: 10.04.2026)
13. Werkzeug. Documentation. https://werkzeug.palletsprojects.com/ (дата звернення: 10.04.2026)

14. SQLite. Official Documentation. https://www.sqlite.org/docs.html (дата звернення: 10.04.2026))
15. Chart.js. Documentation. https://www.chartjs.org/docs (дата звернення: 10.04.2026))
16. SQLAlchemy. Documentation. Version 2.0. https://docs.sqlalchemy.org/ (дата звернення: 10.04.2026)
17. Exchangerate-API. Free Currency Conversion API. https://www.exchangerate-api.com/ (дата звернення: 10.04.2026)
18. Font Awesome. Icons Library. Version 6.4.0. https://fontawesome.com/ (дата звернення: 10.04.2026)
19. Google Fonts. Inter Font Family. https://fonts.google.com/specimen/Inter (дата звернення: 10.04.2026)

ДОДАТОК А. ДЕМОНСТРАЦІЙНІ МАТЕРІАЛИ

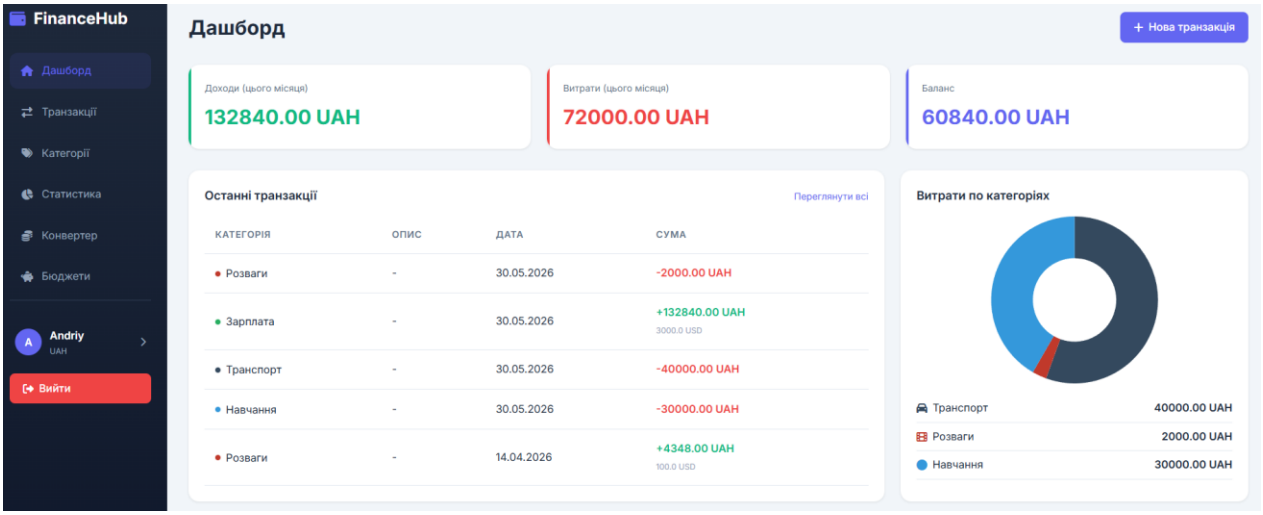

Рисунок А.1- Дашборд з доходами та витратами

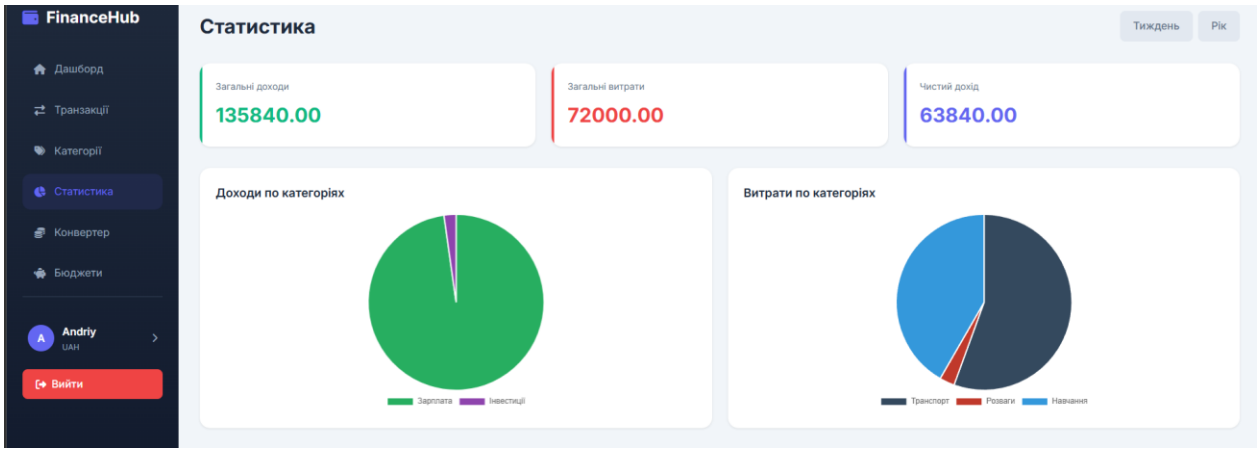

Рисунок А.2- Загальна статистика

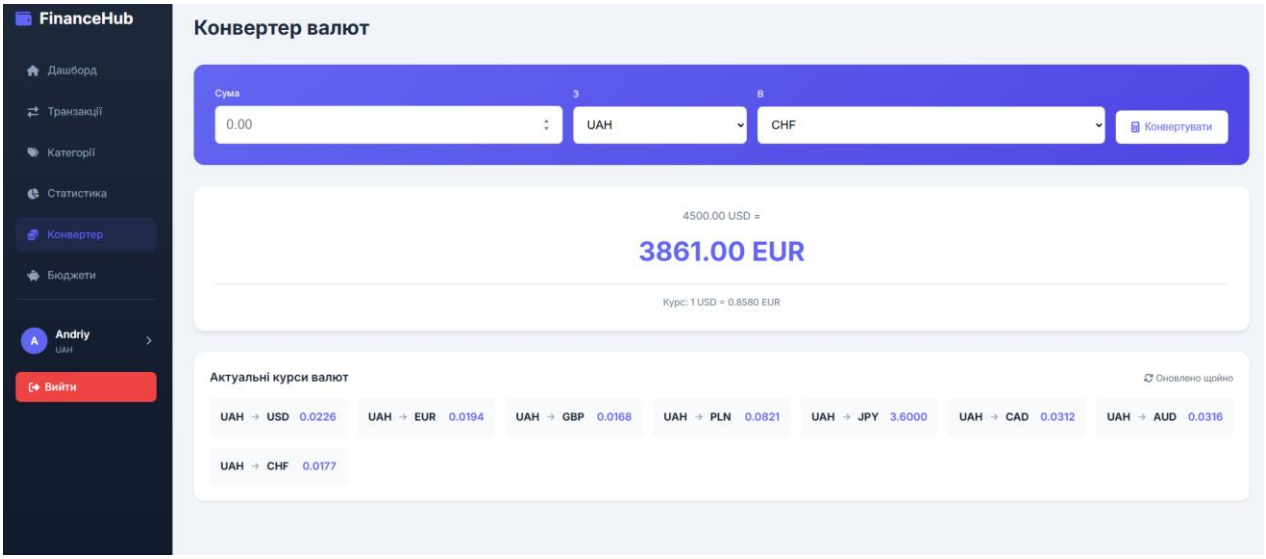

Рисунок А.3- Конвертація валют з USD в EUR

ДОДАТОК Б. ЛІСТИНГИ ОСНОВНИХ МОДУЛІВ

Головний файл app.py:

```
from flask import Flask, render_template, request, redirect, url_for, flash, jsonify, Response
from flask_login import LoginManager, login_user, logout_user, login_required, current_user
from datetime import datetime, timedelta, timezone
import requests

from config import Config
from models import db, User, Category, Transaction, Budget
from database import init_db

app = Flask(__name__)
app.config.from_object(Config)

init_db(app)
login_manager = LoginManager()
login_manager.init_app(app)
login_manager.login_view = 'login'


@login_manager.user_loader
def load_user(user_id):
    return db.session.get(User, int(user_id))


@app.context_processor
def inject_now():
    return {'now': datetime.now}


def get_exchange_rates(base_currency='USD'):
    """Отримує курси валют з API або повертає дефолтні"""
    try:
        url = f"{Config.EXCHANGE_BASE_URL}{base_currency}"
        response = requests.get(url, timeout=10)
        if response.status_code == 200:
            return response.json()
    except:
        pass
    # Дефолтні курси якщо API недоступний
    return {
        'rates': {
            'USD': 1, 'EUR': 0.85, 'GBP': 0.73, 'UAH': 36.5,
            'PLN': 4.0, 'JPY': 110.0, 'CAD': 1.25, 'AUD': 1.35, 'CHF': 0.92
        }
    }


def convert_currency(amount, from_currency, to_currency):
    """Конвертує суму з однієї валюти в іншу"""
    if from_currency == to_currency:
        return amount
    rates = get_exchange_rates(from_currency)
    if rates and 'rates' in rates:
        rate = rates['rates'].get(to_currency)
        if rate:
            return round(amount * rate, 2)
    return amount
```

```
# ==================== АУТЕНТИФІКАЦІЯ ====================

@app.route('/register', methods=['GET', 'POST'])
def register():
    if current_user.is_authenticated:
        return redirect(url_for('dashboard'))

    if request.method == 'POST':
        username = request.form.get('username')
        email = request.form.get('email')
        password = request.form.get('password')
        base_currency = request.form.get('base_currency', 'USD')

        if User.query.filter_by(username=username).first():
            flash('Користувач з таким іменем вже існує', 'error')
            return redirect(url_for('register'))

        if User.query.filter_by(email=email).first():
            flash('Користувач з таким email вже існує', 'error')
            return redirect(url_for('register'))

        user = User(username=username, email=email, base_currency=base_currency)
        user.password = password
        db.session.add(user)
        db.session.commit()

        # Стандартні категорії
        default_cats = [
            ('Зарплата', 'income', '#27ae60', 'fa-money-bill-wave'),
            ('Фріланс', 'income', '#2980b9', 'fa-laptop-code'),
            ('Інвестиції', 'income', '#8e44ad', 'fa-chart-line'),
            ('Продукти', 'expense', '#e67e22', 'fa-shopping-cart'),
            ('Транспорт', 'expense', '#34495e', 'fa-car'),
            ('Житло', 'expense', '#16a085', 'fa-home'),
            ('Розваги', 'expense', '#c0392b', 'fa-film'),
            ('Здоров\'я', 'expense', '#d35400', 'fa-heartbeat'),
        ]

        for name, type_, color, icon in default_cats:
            cat = Category(name=name, type=type_, color=color, icon=icon, user_id=user.id)
            db.session.add(cat)

        db.session.commit()
        flash('Реєстрація успішна!', 'success')
        return redirect(url_for('login'))

    return render_template('register.html')


@app.route('/login', methods=['GET', 'POST'])
def login():
    if current_user.is_authenticated:
        return redirect(url_for('dashboard'))

    if request.method == 'POST':
        username = request.form.get('username')
        password = request.form.get('password')
        remember = bool(request.form.get('remember'))

        user = User.query.filter_by(username=username).first()
        if user and user.check_password(password):
```

```
            login_user(user, remember=remember)
            flash('Ви успішно увійшли!', 'success')
            return redirect(url_for('dashboard'))

        flash('Невірне ім\'я користувача або пароль', 'error')

    return render_template('login.html')


@app.route('/logout')
@login_required
def logout():
    logout_user()
    flash('Ви вийшли з системи', 'info')
    return redirect(url_for('login'))


# ==================== ПРОФІЛЬ ====================

@app.route('/profile', methods=['GET', 'POST'])
@login_required
def profile():
    if request.method == 'POST':
        action = request.form.get('action')

        # Оновлення профілю
        if action == 'update_profile':
            username = request.form.get('username')
            email = request.form.get('email')
            base_currency = request.form.get('base_currency')

            # Перевірка чи ім'я не зайняте іншим користувачем
            if username != current_user.username:
                existing = User.query.filter_by(username=username).first()
                if existing:
                    flash('Це ім\'я користувача вже зайняте', 'error')
                    return redirect(url_for('profile'))

            # Перевірка чи email не зайнятий
            if email != current_user.email:
                existing = User.query.filter_by(email=email).first()
                if existing:
                    flash('Цей email вже використовується', 'error')
                    return redirect(url_for('profile'))

            current_user.username = username
            current_user.email = email
            current_user.base_currency = base_currency
            db.session.commit()
            flash('Профіль оновлено!', 'success')

        # Зміна пароля
        elif action == 'change_password':
            current_password = request.form.get('current_password')
            new_password = request.form.get('new_password')
            confirm_password = request.form.get('confirm_password')

            if not current_user.check_password(current_password):
                flash('Поточний пароль невірний', 'error')
                return redirect(url_for('profile'))

            if new_password != confirm_password:
```

```
            flash('Нові паролі не співпадають', 'error')
            return redirect(url_for('profile'))

        if len(new_password) < 6:
            flash('Пароль має бути не менше 6 символів', 'error')
            return redirect(url_for('profile'))

        current_user.password = new_password
        db.session.commit()
        flash('Пароль змінено!', 'success')

    return redirect(url_for('profile'))

return render_template('profile.html')


# ==================== ГОЛОВНІ МАРШРУТИ ====================

@app.route('/')
@login_required
def dashboard():
    today = datetime.now(timezone.utc)
    start_of_month = today.replace(day=1, hour=0, minute=0, second=0, microsecond=0)

    month_transactions = Transaction.query.filter(
        Transaction.user_id == current_user.id,
        Transaction.date >= start_of_month
    ).all()

    # Використовуємо конвертовані суми (amount_base) для розрахунків
    total_income = sum(t.amount_base for t in month_transactions if t.type == 'income')
    total_expense = sum(t.amount_base for t in month_transactions if t.type == 'expense')
    balance = total_income - total_expense

    recent_transactions = Transaction.query.filter_by(user_id=current_user.id) \
        .order_by(Transaction.date.desc()).limit(5).all()

    category_stats = db.session.query(
        Category.name, Category.color, Category.icon,
        db.func.sum(Transaction.amount_base).label('total')
    ).join(Transaction).filter(
        Transaction.user_id == current_user.id,
        Transaction.type == 'expense',
        Transaction.date >= start_of_month
    ).group_by(Category.id).all()

    return render_template('dashboard.html',
                           total_income=total_income,
                           total_expense=total_expense,
                           balance=balance,
                           base_currency=current_user.base_currency,
                           recent_transactions=recent_transactions,
                           category_stats=category_stats)


@app.route('/transactions', methods=['GET', 'POST'])
@login_required
def transactions():
    if request.method == 'POST':
        amount = float(request.form.get('amount'))
        currency = request.form.get('currency', current_user.base_currency)
        description = request.form.get('description')
```

```
        type_ = request.form.get('type')
        category_id = int(request.form.get('category_id'))
        date_str = request.form.get('date')

        date = datetime.strptime(date_str, '%Y-%m-%d') if date_str else datetime.now(timezone.utc)

        # Автоматична конвертація в базову валюту
        amount_base = convert_currency(amount, currency, current_user.base_currency)

        transaction = Transaction(
            amount=amount,  # Оригінальна сума
            currency=currency,  # Оригінальна валюта
            amount_base=amount_base,  # Конвертована в базову валюту
            description=description,
            type=type_,
            category_id=category_id,
            user_id=current_user.id,
            date=date
        )
        db.session.add(transaction)
        db.session.commit()

        # Показуємо інформацію про конвертацію
        if currency != current_user.base_currency:
            flash(f'Транзакцію додано! Конвертовано: {amount} {currency} = {amount_base}
{current_user.base_currency}',
                  'success')
        else:
            flash('Транзакцію додано!', 'success')

        return redirect(url_for('transactions'))

    filter_type = request.args.get('type', 'all')
    query = Transaction.query.filter_by(user_id=current_user.id)

    if filter_type != 'all':
        query = query.filter_by(type=filter_type)

    transactions = query.order_by(Transaction.date.desc()).all()
    categories = Category.query.filter_by(user_id=current_user.id).all()

    return render_template('transactions.html',
                           transactions=transactions,
                           categories=categories,
                           filter_type=filter_type,
                           today=datetime.now().strftime('%Y-%m-%d'))


@app.route('/transaction/<int:id>/delete', methods=['POST'])
@login_required
def delete_transaction(id):
    transaction = db.session.get(Transaction, id)
    if not transaction or transaction.user_id != current_user.id:
        flash('Доступ заборонено', 'error')
        return redirect(url_for('transactions'))

    db.session.delete(transaction)
    db.session.commit()
    flash('Транзакцію видалено', 'success')
    return redirect(url_for('transactions'))
```

```
@app.route('/categories', methods=['GET', 'POST'])
@login_required
def categories():
    if request.method == 'POST':
        name = request.form.get('name')
        type_ = request.form.get('type')
        color = request.form.get('color', '#3498db')
        icon = request.form.get('icon', 'fa-circle')

        category = Category(name=name, type=type_, color=color, icon=icon, user_id=current_user.id)
        db.session.add(category)
        db.session.commit()
        flash('Категорію створено!', 'success')
        return redirect(url_for('categories'))

    income_categories = Category.query.filter_by(user_id=current_user.id, type='income').all()
    expense_categories = Category.query.filter_by(user_id=current_user.id, type='expense').all()

    return render_template('categories.html',
                           income_categories=income_categories,
                           expense_categories=expense_categories)


@app.route('/category/<int:id>/delete', methods=['POST'])
@login_required
def delete_category(id):
    category = db.session.get(Category, id)
    if not category or category.user_id != current_user.id:
        flash('Доступ заборонено', 'error')
        return redirect(url_for('categories'))

    if category.transactions:
        flash('Неможливо видалити категорію з транзакціями', 'error')
        return redirect(url_for('categories'))

    db.session.delete(category)
    db.session.commit()
    flash('Категорію видалено', 'success')
    return redirect(url_for('categories'))


@app.route('/statistics')
@login_required
def statistics():
    period = request.args.get('period', 'month')
    today = datetime.now(timezone.utc)

    if period == 'week':
        start_date = today - timedelta(days=7)
    elif period == 'year':
        start_date = today.replace(month=1, day=1)
    else:
        start_date = today.replace(day=1)

    # Використовуємо amount_base для статистики
    income_by_category = db.session.query(
        Category.name, Category.color,
        db.func.sum(Transaction.amount_base).label('total')
    ).join(Transaction).filter(
        Transaction.user_id == current_user.id,
        Transaction.type == 'income',
        Transaction.date >= start_date
```

```
    ).group_by(Category.id).all()

    expense_by_category = db.session.query(
        Category.name, Category.color,
        db.func.sum(Transaction.amount_base).label('total')
    ).join(Transaction).filter(
        Transaction.user_id == current_user.id,
        Transaction.type == 'expense',
        Transaction.date >= start_date
    ).group_by(Category.id).all()

    total_income = sum(item.total for item in income_by_category)
    total_expense = sum(item.total for item in expense_by_category)

    # Дані для графіка часу (використовуємо amount_base)
    transactions = Transaction.query.filter(
        Transaction.user_id == current_user.id,
        Transaction.date >= start_date
    ).order_by(Transaction.date).all()

    timeline_data = {}
    for t in transactions:
        date_key = t.date.strftime('%Y-%m-%d')
        if date_key not in timeline_data:
            timeline_data[date_key] = {'income': 0, 'expense': 0}
        timeline_data[date_key][t.type] += t.amount_base

    return render_template('statistics.html',
                           period=period,
                           timeline_data=timeline_data,
                           income_by_category=income_by_category,
                           expense_by_category=expense_by_category,
                           total_income=total_income,
                           total_expense=total_expense)


@app.route('/converter', methods=['GET', 'POST'])
@login_required
def converter():
    result = None
    currencies = ['USD', 'EUR', 'GBP', 'UAH', 'PLN', 'JPY', 'CAD', 'AUD', 'CHF']

    if request.method == 'POST':
        amount = float(request.form.get('amount'))
        from_currency = request.form.get('from_currency')
        to_currency = request.form.get('to_currency')

        converted = convert_currency(amount, from_currency, to_currency)
        rates = get_exchange_rates(from_currency)

        result = {
            'original_amount': amount,
            'from_currency': from_currency,
            'converted_amount': converted,
            'to_currency': to_currency,
            'rate': rates['rates'].get(to_currency, 1) if rates else 1
        }

    return render_template('converter.html',
                           currencies=currencies,
                           result=result,
                           base_currency=current_user.base_currency)
```

```
@app.route('/budgets', methods=['GET', 'POST'])
@login_required
def budgets():
    if request.method == 'POST':
        amount = float(request.form.get('amount'))
        period = request.form.get('period', 'monthly')
        category_id = request.form.get('category_id') or None
        if category_id:
            category_id = int(category_id)

        budget = Budget(amount=amount, period=period, category_id=category_id, user_id=current_user.id)
        db.session.add(budget)
        db.session.commit()
        flash('Бюджет встановлено!', 'success')
        return redirect(url_for('budgets'))

    budgets = Budget.query.filter_by(user_id=current_user.id).all()
    categories = Category.query.filter_by(user_id=current_user.id, type='expense').all()

    budget_progress = []
    for budget in budgets:
        if budget.period == 'monthly':
            start_date = datetime.now(timezone.utc).replace(day=1)
        else:
            start_date = datetime.now(timezone.utc) - timedelta(days=7)

        # Використовуємо amount_base для підрахунку витрат
        query = db.session.query(db.func.sum(Transaction.amount_base)).filter(
            Transaction.user_id == current_user.id,
            Transaction.type == 'expense',
            Transaction.date >= start_date
        )
        if budget.category_id:
            query = query.filter(Transaction.category_id == budget.category_id)

        spent = query.scalar() or 0

        budget_progress.append({
            'budget': budget,
            'spent': spent,
            'remaining': budget.amount - spent,
            'percentage': min((spent / budget.amount) * 100, 100) if budget.amount > 0 else 0
        })

    return render_template('budgets.html', budgets=budget_progress, categories=categories)


@app.route('/export/transactions')
@login_required
def export_transactions():
    import csv
    import io

    transactions = Transaction.query.filter_by(user_id=current_user.id).all()

    output = io.StringIO()
    writer = csv.writer(output)
    writer.writerow(['Дата', 'Тип', 'Категорія', 'Сума', 'Валюта', 'Сума (базова)', 'Базова валюта', 'Опис'])

    for t in transactions:
```

```
        writer.writerow([
            t.date.strftime('%Y-%m-%d'),
            t.type,
            t.category.name,
            t.amount,
            t.currency,
            t.amount_base,
            current_user.base_currency,
            t.description or ''
        ])

    output.seek(0)

    return Response(
        output,
        mimetype='text/csv',
        headers={
            'Content-Disposition': f'attachment; filename=transactions_{datetime.now().strftime("%Y%m%d")}.csv'
        }
    )


@app.route('/api/exchange-rates')
@login_required
def api_exchange_rates():
    base = request.args.get('base', current_user.base_currency)
    rates = get_exchange_rates(base)
    return jsonify(rates)


if __name__ == '__main__':
    app.run(debug=True, port=5000)
```

Файл base.html:

```
<!DOCTYPE html>
<html lang="uk">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>{% block title %}Finance Manager{% endblock %}</title>
    <link rel="stylesheet" href="https://cdnjs.cloudflare.com/ajax/libs/font-awesome/6.4.0/css/all.min.css">
    <link href="https://fonts.googleapis.com/css2?family=Inter:wght@300;400;500;600;700&display=swap"
rel="stylesheet">
    <script src="https://cdn.jsdelivr.net/npm/chart.js"></script>
    <style>
        :root {
            --primary: #6366f1;
            --primary-dark: #4f46e5;
            --success: #10b981;
            --danger: #ef4444;
            --dark: #1e293b;
            --light: #f8fafc;
            --gray: #64748b;
            --sidebar-width: 260px;
        }

        * { margin: 0; padding: 0; box-sizing: border-box; }

        body {
            font-family: 'Inter', sans-serif;
            background: #f1f5f9;
            color: var(--dark);
```

```
    line-height: 1.6;
}

.sidebar {
    position: fixed;
    left: 0;
    top: 0;
    width: var(--sidebar-width);
    height: 100vh;
    background: linear-gradient(180deg, #1e293b 0%, #0f172a 100%);
    color: white;
    padding: 1.5rem;
    overflow-y: auto;
    z-index: 1000;
}

.logo {
    font-size: 1.5rem;
    font-weight: 700;
    margin-bottom: 2rem;
    display: flex;
    align-items: center;
    gap: 0.75rem;
    color: white;
    text-decoration: none;
}

.logo i { color: var(--primary); }

.nav-menu { list-style: none; }
.nav-item { margin-bottom: 0.5rem; }

.nav-link {
    display: flex;
    align-items: center;
    gap: 0.75rem;
    padding: 0.875rem 1rem;
    color: #94a3b8;
    text-decoration: none;
    border-radius: 0.75rem;
    transition: all 0.3s ease;
}

.nav-link:hover, .nav-link.active {
    background: rgba(99, 102, 241, 0.1);
    color: var(--primary);
}

.nav-link i { width: 20px; text-align: center; }

.user-section {
    margin-top: auto;
    padding-top: 2rem;
    border-top: 1px solid #334155;
}

.user-info {
    display: flex;
    align-items: center;
    gap: 0.75rem;
    margin-bottom: 1rem;
    text-decoration: none;
```

```
    color: inherit;
    cursor: pointer;
    padding: 0.5rem;
    border-radius: 0.75rem;
    transition: all 0.3s ease;
}

.user-info:hover {
    background: rgba(255,255,255,0.1);
}

.user-info .fa-chevron-right {
    transition: transform 0.3s ease;
}

.user-info:hover .fa-chevron-right {
    transform: translateX(4px);
}

.user-avatar {
    width: 40px;
    height: 40px;
    border-radius: 50%;
    background: var(--primary);
    display: flex;
    align-items: center;
    justify-content: center;
    font-weight: 600;
}

.main-content {
    margin-left: var(--sidebar-width);
    padding: 2rem;
    min-height: 100vh;
}

.header {
    display: flex;
    justify-content: space-between;
    align-items: center;
    margin-bottom: 2rem;
}

.page-title {
    font-size: 1.875rem;
    font-weight: 700;
    color: var(--dark);
}

.card {
    background: white;
    border-radius: 1rem;
    padding: 1.5rem;
    box-shadow: 0 1px 3px 0 rgba(0, 0, 0, 0.1);
    margin-bottom: 1.5rem;
}

.card-header {
    display: flex;
    justify-content: space-between;
    align-items: center;
    margin-bottom: 1rem;
```

```
}

.card-title {
    font-size: 1.125rem;
    font-weight: 600;
}

.stats-grid {
    display: grid;
    grid-template-columns: repeat(auto-fit, minmax(250px, 1fr));
    gap: 1.5rem;
    margin-bottom: 2rem;
}

.stat-card {
    background: white;
    border-radius: 1rem;
    padding: 1.5rem;
    box-shadow: 0 1px 3px 0 rgba(0, 0, 0, 0.1);
    position: relative;
    overflow: hidden;
}

.stat-card::before {
    content: '';
    position: absolute;
    top: 0;
    left: 0;
    width: 4px;
    height: 100%;
}

.stat-card.income::before { background: var(--success); }
.stat-card.expense::before { background: var(--danger); }
.stat-card.balance::before { background: var(--primary); }

.stat-label {
    font-size: 0.875rem;
    color: var(--gray);
    margin-bottom: 0.5rem;
}

.stat-value {
    font-size: 1.875rem;
    font-weight: 700;
}

.stat-card.income .stat-value { color: var(--success); }
.stat-card.expense .stat-value { color: var(--danger); }
.stat-card.balance .stat-value { color: var(--primary); }

.btn {
    display: inline-flex;
    align-items: center;
    gap: 0.5rem;
    padding: 0.625rem 1.25rem;
    border-radius: 0.5rem;
    font-weight: 500;
    text-decoration: none;
    border: none;
    cursor: pointer;
    transition: all 0.3s ease;
```

```
}

.btn-primary {
    background: var(--primary);
    color: white;
}

.btn-primary:hover { background: var(--primary-dark); }
.btn-success { background: var(--success); color: white; }
.btn-danger { background: var(--danger); color: white; }

.form-group { margin-bottom: 1rem; }

.form-label {
    display: block;
    margin-bottom: 0.5rem;
    font-weight: 500;
    font-size: 0.875rem;
}

.form-control {
    width: 100%;
    padding: 0.625rem 0.875rem;
    border: 1px solid #e2e8f0;
    border-radius: 0.5rem;
    font-size: 0.875rem;
    transition: border-color 0.3s ease;
}

.form-control:focus {
    outline: none;
    border-color: var(--primary);
}

.table {
    width: 100%;
    border-collapse: collapse;
}

.table th, .table td {
    padding: 1rem;
    text-align: left;
    border-bottom: 1px solid #e2e8f0;
}

.table th {
    font-weight: 600;
    color: var(--gray);
    font-size: 0.875rem;
    text-transform: uppercase;
    letter-spacing: 0.05em;
}

.table tr:hover { background: #f8fafc; }

.badge {
    display: inline-flex;
    align-items: center;
    padding: 0.25rem 0.75rem;
    border-radius: 9999px;
    font-size: 0.75rem;
    font-weight: 600;
```

```
}

.badge-income {
    background: #d1fae5;
    color: #065f46;
}

.badge-expense {
    background: #fee2e2;
    color: #991b1b;
}

.alert {
    padding: 1rem;
    border-radius: 0.5rem;
    margin-bottom: 1rem;
}

.alert-success {
    background: #d1fae5;
    color: #065f46;
    border: 1px solid #a7f3d0;
}

.alert-error {
    background: #fee2e2;
    color: #991b1b;
    border: 1px solid #fecaca;
}

.auth-container {
    min-height: 100vh;
    display: flex;
    align-items: center;
    justify-content: center;
    background: linear-gradient(135deg, #667eea 0%, #764ba2 100%);
    padding: 2rem;
}

.auth-card {
    background: white;
    padding: 2.5rem;
    border-radius: 1rem;
    width: 100%;
    max-width: 420px;
    box-shadow: 0 25px 50px -12px rgba(0, 0, 0, 0.25);
}

.auth-title {
    font-size: 1.5rem;
    font-weight: 700;
    text-align: center;
    margin-bottom: 1.5rem;
}

.chart-container {
    position: relative;
    height: 300px;
    margin-top: 1rem;
}

.category-grid {
```

```
    display: grid;
    grid-template-columns: repeat(auto-fill, minmax(200px, 1fr));
    gap: 1rem;
}

.category-card {
    background: white;
    border-radius: 0.75rem;
    padding: 1rem;
    display: flex;
    align-items: center;
    gap: 0.75rem;
    box-shadow: 0 1px 3px rgba(0,0,0,0.1);
}

.category-icon {
    width: 40px;
    height: 40px;
    border-radius: 0.5rem;
    display: flex;
    align-items: center;
    justify-content: center;
    color: white;
}

.converter-box {
    background: linear-gradient(135deg, var(--primary) 0%, var(--primary-dark) 100%);
    color: white;
    padding: 2rem;
    border-radius: 1rem;
    margin-bottom: 2rem;
}

.result-box {
    background: white;
    padding: 2rem;
    border-radius: 1rem;
    text-align: center;
    box-shadow: 0 4px 6px -1px rgba(0, 0, 0, 0.1);
}

.result-amount {
    font-size: 2.5rem;
    font-weight: 700;
    color: var(--primary);
}

.progress-bar {
    width: 100%;
    height: 8px;
    background-color: #e2e8f0;
    border-radius: 4px;
    overflow: hidden;
}

.progress-fill {
    height: 100%;
    background: linear-gradient(90deg, var(--primary) 0%, var(--primary-dark) 100%);
    transition: width 0.5s ease;
}

@keyframes fadeIn {
```

```
            from { opacity: 0; transform: translateY(10px); }
            to { opacity: 1; transform: translateY(0); }
        }

        .card { animation: fadeIn 0.5s ease; }

        @media (max-width: 1024px) {
            .sidebar { transform: translateX(-100%); transition: transform 0.3s ease; }
            .sidebar.active { transform: translateX(0); }
            .main-content { margin-left: 0; }
            .stats-grid { grid-template-columns: 1fr; }
        }

        .filters {
            display: flex;
            gap: 1rem;
            margin-bottom: 1.5rem;
            flex-wrap: wrap;
        }

        .filter-group {
            display: flex;
            align-items: center;
            gap: 0.5rem;
        }
    </style>
    {% block extra_css %}{% endblock %}
</head>
<body>
    {% if current_user.is_authenticated %}
    <aside class="sidebar">
        <a href="{{ url_for('dashboard') }}" class="logo">
            <i class="fas fa-wallet"></i>
            <span>FinanceHub</span>
        </a>

        <ul class="nav-menu">
            <li class="nav-item">
                <a href="{{ url_for('dashboard') }}" class="nav-link {% if request.endpoint == 'dashboard' %}active{%
endif %}">
                    <i class="fas fa-home"></i>
                    <span>Дашборд</span>
                </a>
            </li>
            <li class="nav-item">
                <a href="{{ url_for('transactions') }}" class="nav-link {% if request.endpoint == 'transactions'
%}active{% endif %}">
                    <i class="fas fa-exchange-alt"></i>
                    <span>Транзакції</span>
                </a>
            </li>
            <li class="nav-item">
                <a href="{{ url_for('categories') }}" class="nav-link {% if request.endpoint == 'categories' %}active{%
endif %}">
                    <i class="fas fa-tags"></i>
                    <span>Категорії</span>
                </a>
            </li>
            <li class="nav-item">
                <a href="{{ url_for('statistics') }}" class="nav-link {% if request.endpoint == 'statistics' %}active{% endif
%}">
                    <i class="fas fa-chart-pie"></i>
```

```
                <span>Статистика</span>
            </a>
        </li>
        <li class="nav-item">
            <a href="{{ url_for('converter') }}" class="nav-link {% if request.endpoint == 'converter' %}active{%
endif %}">
                <i class="fas fa-coins"></i>
                <span>Конвертер</span>
            </a>
        </li>
        <li class="nav-item">
            <a href="{{ url_for('budgets') }}" class="nav-link {% if request.endpoint == 'budgets' %}active{% endif
%}">
                <i class="fas fa-piggy-bank"></i>
                <span>Бюджети</span>
            </a>
        </li>
    </ul>

    <div class="user-section">
        <a href="{{ url_for('profile') }}" class="user-info">
            <div class="user-avatar">
                {{ current_user.username[0].upper() }}
            </div>
            <div>
                <div style="font-weight: 600;">{{ current_user.username }}</div>
                <div style="font-size: 0.75rem; color: #94a3b8;">{{ current_user.base_currency }}</div>
            </div>
            <i class="fas fa-chevron-right" style="margin-left: auto; color: #94a3b8; font-size: 0.875rem;"></i>
        </a>
        <a href="{{ url_for('logout') }}" class="btn btn-danger" style="width: 100%;">
            <i class="fas fa-sign-out-alt"></i>
            Вийти
        </a>
    </div>
</aside>

<main class="main-content">
    {% with messages = get_flashed_messages(with_categories=true) %}
        {% if messages %}
            {% for category, message in messages %}
                <div class="alert alert-{{ 'error' if category == 'error' else category }}">
                    {{ message }}
                </div>
            {% endfor %}
        {% endif %}
    {% endwith %}

    {% block content %}{% endblock %}
</main>
{% else %}
    {% block auth_content %}{% endblock %}
{% endif %}

<script>
    setTimeout(() => {
        document.querySelectorAll('.alert').forEach(alert => {
            alert.style.opacity = '0';
            setTimeout(() => alert.remove(), 300);
        });
    }, 5000);
</script>
```

```
    {% block extra_js %}{% endblock %}
</body>
</html>
```