Міністерство освіти і науки України
Криворізький національний університет
Кафедра моделювання та програмного забезпечення

КВАЛІФІКАЦІЙНА РОБОТА

на здобуття ступеня вищої освіти магістра

за спеціальністю 121 – Інженерія програмного забезпечення

На тему: Розробка веб-додатку для обліку особистих фінансів з аналізом фінансової ефективності

Засвідчую, що в цій кваліфікаційній роботі немає запозичень із праць інших авторів без відповідних посилань.

Студент гр. ІПЗ–23-2м_____/ Д. І. Бухбіндер /

Керівник кваліфікаційної роботи _____/ І. А. Котов /

Економіко-організаційна частина _____/ О. В. Шамрай /

Нормоконтроль _____/ І. А. Котов /

Завідувач кафедрою _____/ А. М. Стрюк /

Кривий Ріг

2024

Криворізький національний університет

Факультет: Інформаційних технологій

Кафедра: Моделювання та програмного забезпечення

Ступінь вищої освіти: магістр

Спеціальність: 121 – Інженерія програмного забезпечення

ЗАТВЕРДЖУЮ:
Зав. кафедри
_____ А. М. Стрюк
«____» _____ 2024 р.

ЗАВДАННЯ

на кваліфікаційну роботу

студента групи ІПЗ–23-2м Бухбіндера Давида Ігоровича

1. Тема: Розробка веб-додатку для обліку особистих фінансів з аналізом фінансової ефективності затверджена наказом по КНУ № 278с від «15» квітня 2024 р.

2. Термін подання студентом закінченої роботи: «30» листопада 2024р.

3. Вихідні дані по роботі: розробка веб-додатку для обліку особистих фінансів з аналізом фінансової ефективності.

4. Зміст пояснювальної записки (перелік питань, що їх треба розробити): виконати аналіз існуючих програм-аналогів на ринку, визначити функціонал розроблюваного додатку, спроектувати додаток, розробити програмне забезпечення, здійснити тестування розробленого додатку.

5. Перелік ілюстративного матеріалу: функціональна схема, блок–схема розробленого алгоритму, скріншоти роботи програми.

Календарний план:

№	Найменування етапів кваліфікаційної роботи	Термін виконання
1	Аналіз літературних джерел та огляд інтернет-ресурсів з заданої тематики	08.01.24 – 14.02.24
2	Пошук існуючих методів вирішення проблеми	15.02.24 – 10.03.24
3	Формулювання актуальності роботи і постановка завдань	11.03.24 – 29.03.24
4	Оформлення матеріалів першого розділу роботи	30.03.24 – 15.04.24
5	Визначення об’єкту, предмету та мети дослідження	16.04.24 – 02.05.24
6	Оформлення матеріалів другого розділу роботи	03.05.24 – 12.05.24
7	Розробка інформаційного забезпечення системи	13.05.24 – 06.06.24
8	Оформлення матеріалів третього розділу роботи	07.06.24 – 25.06.24
9	Розробка функціональної схеми та алгоритму програми	26.06.24 – 11.07.24
10	Розробка програмного забезпечення системи	12.07.24 – 10.09.24
11	Тестування програмного забезпечення	11.09.24 – 04.10.24
12	Оформлення матеріалів четвертого розділу роботи	05.10.24 – 16.10.24
13	Аналіз економічної ефективності інновації	17.10.24 – 02.11.24
14	Оформлення матеріалів п’ятого розділу роботи	03.11.24 – 14.11.24
15	Остаточне оформлення пояснювальної записки	15.11.24 – 29.11.24

Дата видачі завдання: «05» січня 2024р.

Студент: _____ / Д. І. Бухбіндер /

Керівник роботи: _____ / І. А. Котов /

РЕФЕРАТ

ВЕБ-ДОДАТОК, ОСОБИСТІ ФІНАНСИ, АНАЛІЗ ФІНАНСОВОЇ ЕФЕКТИВНОСТІ, УПРАВЛІННЯ БЮДЖЕТОМ, ДОХОДИ ТА ВИТРАТИ, ФІНАНСОВЕ ПЛАНУВАННЯ, ТРЕКЕР ВИТРАТ, ФІНАНСОВА СТАТИСТИКА

Пояснювальна записка: 97 с., 30 рис., 3 табл., 1 дод., 25 джерел.

Мета кваліфікаційної роботи: розробка веб-додатку для обліку особистих фінансів з аналізом фінансової ефективності.

Об’єкт досліджень: процес розроблення програмного забезпечення для обліку особистих фінансів на основі веб-технологій.

У теоретичній частині роботи виконано аналіз існуючих на сьогодні аналогічних програмних рішень на ринку програмного забезпечення. Зазначені сильні та слабкі сторони існуючих програмних продуктів. Обґрунтовані актуальність роботи, мета та сформульовані завдання для розробки веб-додатку. Розглянуті можливі методи вирішення проблеми, розроблено математичне та інформаційне забезпечення створюваної системи.

У практичній частині кваліфікаційної роботи реалізовано функціональну схему розроблюваного додатку та алгоритм його роботи. Розроблено інтерфейс веб-додатку та програмну логіку його роботи. Проведено тестування розробленого програмного забезпечення.

Створений веб-додаток для обліку особистих фінансів з аналізом фінансової ефективності може знайти своє застосування у широкому колі користувачів.

ABSTRACT

WEB APPLICATION, PERSONAL FINANCE, FINANCIAL PERFORMANCE ANALYSIS, BUDGET MANAGEMENT, INCOME AND EXPENSES, FINANCIAL PLANNING, EXPENSE TRACKER, FINANCIAL STATISTICS

Explanatory note: 97 p., 30 fig., 3 tabl., 1 app., 25 references.

The aim of the qualifying work: development of the web application for personal finance accounting with financial efficiency analysis.

Design object: the process of developing web-based personal finance software.

In the theoretical part of the work, the analysis of existing similar software solutions in the software market has performed. The strengths and weaknesses of existing software products have indicated. The relevance of the work, the purpose and objectives for the development of the web application have formulated. Possible methods of solving the problem have considered, mathematical and information support of the created system have developed.

In the practical part of the qualification work, the functional scheme of the developed web application and the algorithm of its work have implemented. The interface of the web application and the program logic of the web application have developed. The developed software has tested.

The created web application for personal finance accounting with financial efficiency analysis can be used by a wide range of users.

ЗМІСТ

ВСТУП

В умовах військового стану та економічної нестабільності питання управління особистими фінансами набуває особливої важливості. Сучасні користувачі стикаються з необхідністю не тільки контролювати свої доходи і витрати, а й аналізувати фінансову ефективність для досягнення своїх довгострокових цілей. Зростання рівня життя та ускладнення фінансових інструментів вимагають використання більш сучасних та інтуїтивно зрозумілих рішень, які дають можливість ефективно планувати бюджет, відстежувати фінансові потоки та ухвалювати обґрунтовані рішення.

Розробка веб-додатку для обліку особистих фінансів з аналізом фінансової ефективності дозволить запропонувати користувачам комплексний інструмент, що має поєднати зручність використання з потужними аналітичними можливостями. Такий застосунок забезпечить не тільки просте введення та класифікацію доходів і витрат, а й надасть візуалізацію даних, розрахунок ключових фінансових показників, а також дозволить подальші витрати та доходи.

Актуальність дипломного проєкту зумовлена зростаючим попитом на програмні рішення, що сприяють ефективному управлінню особистими фінансами. Розроблений веб-додаток буде відповідає зазначеним потребам, та запропонує нові підходи до аналізу фінансової ефективності, що дасть змогу користувачам підвищити якість керування своїми заощадженнями і дозволить забезпечити свою фінансову стабільність.

Таким чином, ця робота спрямована безпосередньо на створення функціонального веб-додатка, що стане корисним інструментом для широкого кола користувачів, які прагнуть покращити управління своїми фінансами та досягти поставлених фінансових цілей. Реалізація цього проєкту матиме, в першу чергу, соціальний ефект, допомагаючи людям більш усвідомлено підходити до питань особистих фінансів. Завдяку реалізації у вигляді веб-додатку, застосування розробленого програмного забезпечення буде можливе

як на базі персональних комп’ютерів, так і з використанням мобільних пристроїв та планшетів, що забезпечить безперебійний доступ до даних користувача через мережу Інтернет з різних пристроїв та локацій.

1 ТЕОРЕТИЧНІ ЗАСАДИ УПРАВЛІННЯ ОСОБИСТИМИ ФІНАНСАМИ ТА АНАЛІЗ НАЯВНИХ ПРОГРАМНИХ РІШЕНЬ

1.1 Поняття та значення управління особистими фінансами

Управління особистими фінансами являє собою сукупність процесів планування, контролю та оптимізації доходів і витрат людини або домогосподарства з метою досягнення фінансової стабільності та забезпечення добробуту. Цей процес може охоплювати такі моменти, як складання бюджету, накопичення заощаджень, інвестування, управління боргами та планування фінансових цілей на короткострокову і довгострокову перспективу (рисунок 1.1).

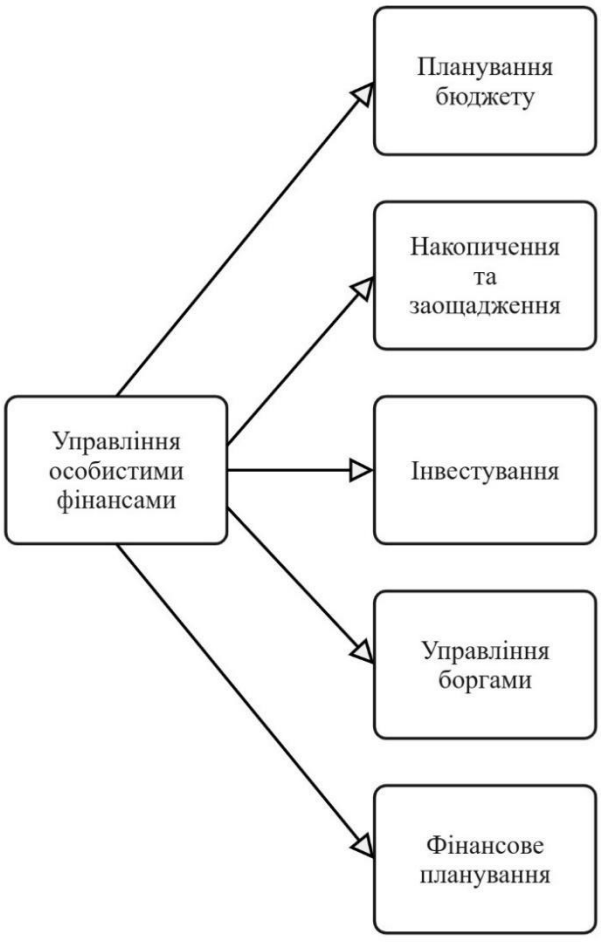


Рисунок 1.1 – Напрямки управління особистими фінансами

Розглянемо детальніше зазначені напрямки.

Планування бюджету виступає основою цього процесу, дозволяючи розподіляти доходи між необхідними витратами та заощадженнями. Складання детального бюджету допомагає не лише уникнути непотрібних витрат, але й забезпечує можливість накопичення коштів для майбутніх потреб або інвестицій. Відомо, що люди, що ведуть детальний облік своїх фінансів, мають більшу ймовірність досягти своїх фінансових цілей [1].

Накопичення заощаджень є наступним важливим елементом управління особистими фінансами. Воно забезпечує фінансову подушку безпеки на випадок непередбачуваних витрат або раптової втрати доходу. Важливість заощаджень підкреслюється у [2], де зазначається, що регулярні заощадження сприяють підвищенню фінансової стійкості та зменшенню стресу, пов’язаного з фінансовими труднощами. Систематичне накопичення дозволяє також планувати великі покупки, такі як купівля житла або автомобіля, а також інвестувати у власний розвиток чи освіту.

Інвестування, в свою чергу, дозволяє примножувати накопичені кошти та забезпечувати додатковий дохід. Правильне інвестування в різні фінансові інструменти, такі як акції, облігації або нерухомість, сприяє зростанню капіталу та захисту від інфляції. Інвестування також відкриває можливості для досягнення довгострокових фінансових цілей, таких як забезпечення комфортної пенсії або фінансування освіти дітей. Теоретичні підходи до інвестування, зокрема моделі ризику та доходності [3], допомагають людям приймати обґрунтовані рішення щодо розподілу своїх ресурсів.

Управління боргами є ще одним критично важливим аспектом особистих фінансів. Ефективний контроль за заборгованістю дозволяє мінімізувати витрати на обслуговування боргів та уникнути фінансових криз. Це включає в себе не лише своєчасне погашення кредитів, але й стратегічне планування для зменшення загальної заборгованості. Управління боргами також пов’язане з потенційним підвищенням кредитного рейтингу, що

відкриває доступ до вигідніших фінансових умов для людини у майбутньому [4].

Фінансове планування на короткострокову та довгострокову перспективу дозволяє визначити конкретні фінансові цілі та розробити стратегії для їх досягнення [5]. Це може включати планування купівлі житла, фінансування освіти, підготовку до пенсії або створення резервного фонду. Довгострокове планування забезпечує систематичний підхід до управління фінансами, дозволяючи ефективно розподіляти ресурси та адаптуватися до змін у фінансовій ситуації.

Загалом, управління особистими фінансами є комплексним процесом, що вимагає розуміння фінансових принципів та здатності адаптуватися до змін у економічному середовищі. Ефективне управління фінансами не тільки забезпечує фінансову стабільність для тих, хто займається ним, але й сприяє підвищенню загального рівня життя та економічного благополуччя суспільства. З урахуванням того, що в більшості випадків фінансові рішення мають значний вплив на якість життя, вміння керувати особистими фінансами стає ключовим чинником успіху та добробуту кожної людини.

1.2 Актуальність розроблення та використання програмного забезпечення для управління особистими фінансами

Як відзначалося вище, сьогодні, у зв’язку із воєним станом та стрімкими економічними змінами, управління особистими фінансами набуває дедалі більшої значущості. Зміна рівня життя, ускладнення фінансових продуктів і послуг, а також зростаюча невизначеність економічного середовища вимагають не тільки усвідомленого підходу до витрачання коштів, а й ефективних інструментів для планування та контролю фінансових потоків. У цьому контексті розробка та використання спеціалізованого програмного забезпечення для управління особистими фінансами стає не просто актуальним, а вкрай необхідним для вирішення завданням.

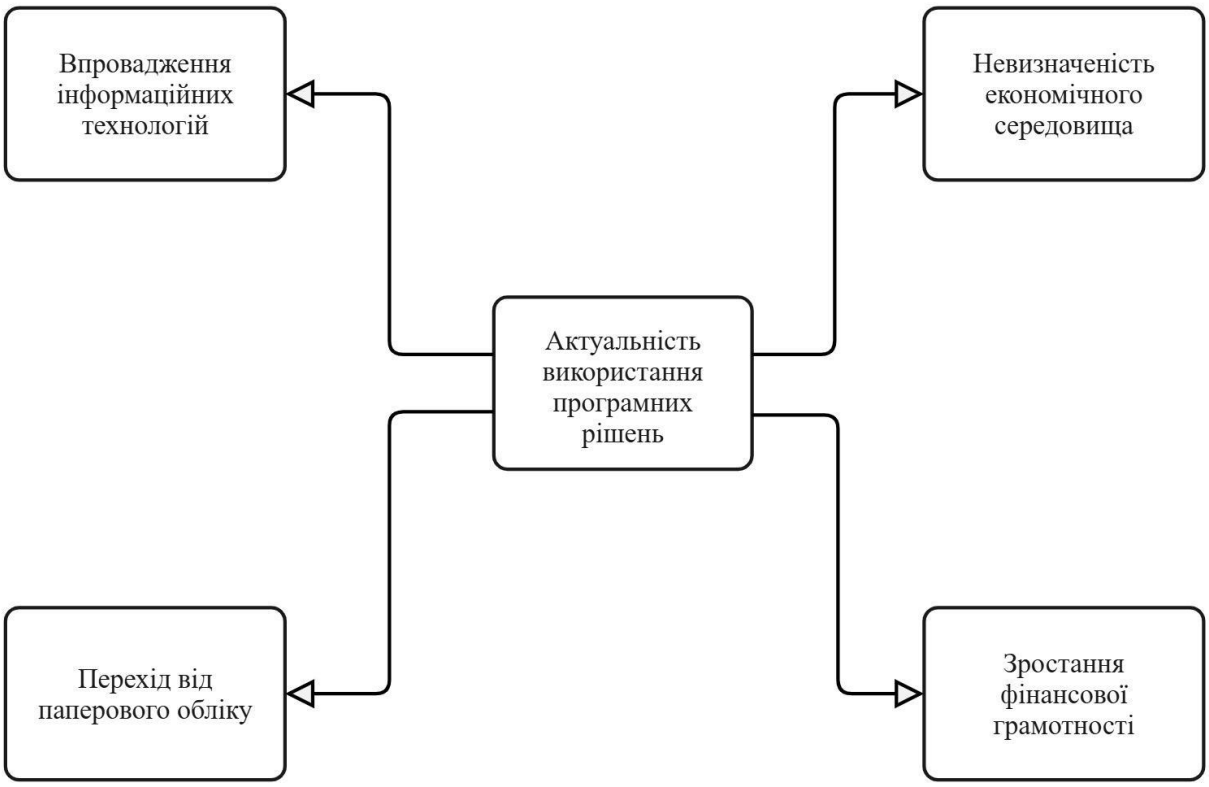


Рисунок 1.2 – Актуальність розроблення та використання програмного забезпечення для управління особистими фінансами

Одним із ключових чинників, що сприяють зростанню попиту на програмні рішення для управління фінансами, є впровадження інформаційних технологій у суспільство. Інформаційні технології кардинально змінили спосіб взаємодії людей з фінансовою інформацією, надавши доступ до онлайн-сервісів і мобільних застосунків для виконання різноманітних завдань, включно з управлінням особистими фінансами. Це не тільки спростило процес відстеження доходів і витрат, а й відкрило нові можливості для аналізу та прогнозування фінансової ефективності. Відомо [6], що понад 70% населення розвинених країн активно використовують цифрові інструменти для управління своїми фінансами. Така тенденція свідчить про необхідність подальшого розвитку та вдосконалення програмного забезпечення в цій галузі.

Невизначеність економічного середовища також посилює необхідність в ефективних інструментах управління особистими фінансами. Коливання

валютних курсів, інфляційні процеси та фінансові кризи, не кажучі про військові дії, істотно впливають на добробут людей, роблячи фінансове планування складнішим і критично важливим. В умовах, коли економічна стабільність не гарантована, наявність програм для планування бюджету та аналізу фінансової ефективності стає важливим фактором для підтримки особистої фінансової стабільності. Програмні рішення допомагають адаптуватися до швидко мінливих умов ринку, надаючи актуальну інформацію та інструменти для прийняття обґрунтованих рішень [7].

Зростання фінансової грамотності серед населення є ще одним аспектом, що підкреслює актуальність розробки програмного забезпечення для управління особистими фінансами. Сучасні користувачі дедалі більше усвідомлюють важливість уміння управляти своїми фінансами і прагнуть поліпшити свої навички в цій галузі. Доступність освітніх ресурсів та інформаційних технологій сприяє цьому прагненню. Програмні рішення, оснащені аналітичними та прогнозними функціями, дають змогу користувачам не тільки фіксувати поточні фінансові показники, а й розробляти довгострокові стратегії на основі достовірних даних. икористання Використання таких інструментів зазвичай сприяє підвищенню рівня фінансової грамотності та поліпшенню фінансової поведінки.

Крім того, перехід від паперового обліку фінансів до цифрових платформ пропонує значні переваги в прискоренні розрахунків і підвищенні наочності. Традиційні методи ведення фінансових записів на папері часто забирають багато часу, схильні до людських помилок і не надають можливості для швидкого аналізу даних. Програмне забезпечення автоматизує рутинні процеси, даючи змогу миттєво розраховувати бюджети, відстежувати витрати і доходи, а також формувати докладні звіти. Візуалізація даних за допомогою графіків і діаграм робить фінансову інформацію зрозумілішою і доступнішою, полегшуючи ухвалення рішень і виявлення тенденцій. Це підвищує ефективність управління фінансами та сприяє досягненню фінансових цілей [8].

Таким чином, актуальність розроблення та використання програмного забезпечення для управління особистими фінансами зумовлена безліччю чинників: упровадженням інформаційних технологій, невизначеністю економічного середовища, зростанням фінансової грамотності та необхідністю більш ефективних і наочних методів фінансового управління. Сучасні користувачі потребують зручних, функціональних і надійних інструментів для управління своїми фінансами, що відкриває перспективи для розроблення застосунків у цій галузі. Такі рішення сприяють поліпшенню фінансового благополуччя їх користувачів, тому їх розробка є актуальним завданням.

На ринку вже існує деяка кількість програмних рішень для реалізації поставленої задачі, тож доцільно дослідити їх та виявити їхні переваги та недоліки.

1.3 Огляд існуючих програмних засобів для обліку та керування фінансами

1.3.1. Quicken Classic

Quicken Classic [9] вважається найстарішим додатком для обліку особистих фінансів, що доступний сьогодні.

Для початку роботи з програмою необхідно створити обліковий запис у Quicken, і після цього можна додати свої фінансові рахунки в Інтернеті в і надати програмі дозвіл на отримання даних з цих рахунків, щоб вона змогла імпортувати транзакції. Після підключення власних рахунків Quicken бачить, де і коли ви використовуєте кредитну картку і скільки ви витрачаєте грошей. Вся ця інформація відображається у вигляді діаграм і списків на інформаційній панелі Quicken.

Quicken має достатньо багато функцій, також дозволено й кастомізувати зовнішній вигляд програми для більш зручного користування.

Управління транзакціями - записами про доходи і витрати - є достаньо гнучким у Quicken. Користувач може налаштовувати стовпці кожного реєстру,

щоб отримати потрібний вигляд, додавати або редагувати категорії, а також використовувати інструменти в меню транзакцій, щоб додавати примітки, прапорці, вкладення і розділяти транзакції. Також додаток дозволяє, наприклад, призначати податкові рядки, що означає, що користувач може створювати звіти для використання при підготовці податкової декларації.

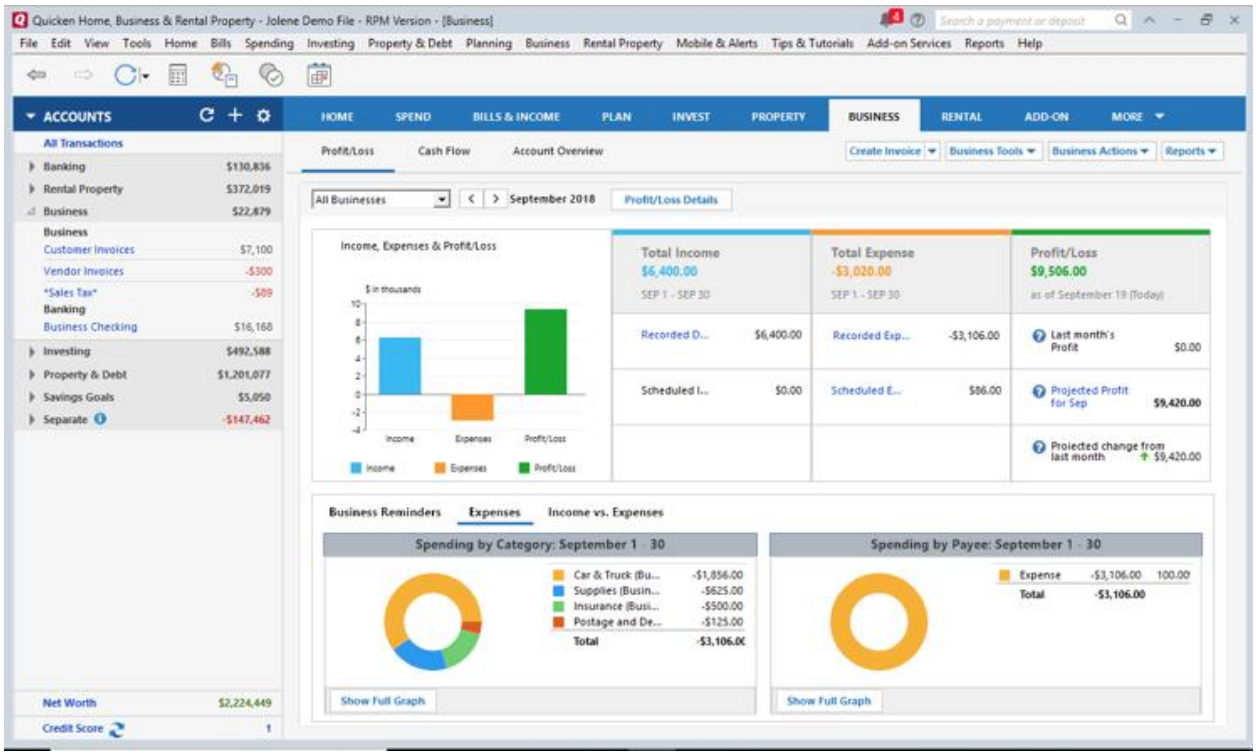


Рисунок 1.3 – Інтерфейс додатку Quicken

Нові користувачі Quicken можуть скористатися безкоштовною послугою Quicken 1-2-3, яка допоможе розпочати роботу, обмінюючись зображеннями з екрана з експертом служби підтримки.

Бюджети у Quicken, як і всі бюджети, працюють як калькулятори. Вони додають доходи і віднімають витрати, відстежуючи, скільки грошей користувач витрачає в різних категоріях, і показуючи, де він перебуває по відношенню до поставлених цілей.

Можна вибрати власні категорії або дозволити Quicken зробити це на основі минулих транзакцій. Чим більше часу проходить і чим більше транзакцій потрапляє до бюджету, тим краще користувач зможе аналізувати

свої витрати і вносити корективи. Quicken використовує кольорові смуги для ілюстрації бюджетів, щоб одразу можна було побачити, де і коли потенційно можуть виникнути проблеми. Для більш глобального контролю свого бюджету можна скористатися річною таблицею Quicken, схожою на електронну таблицю.

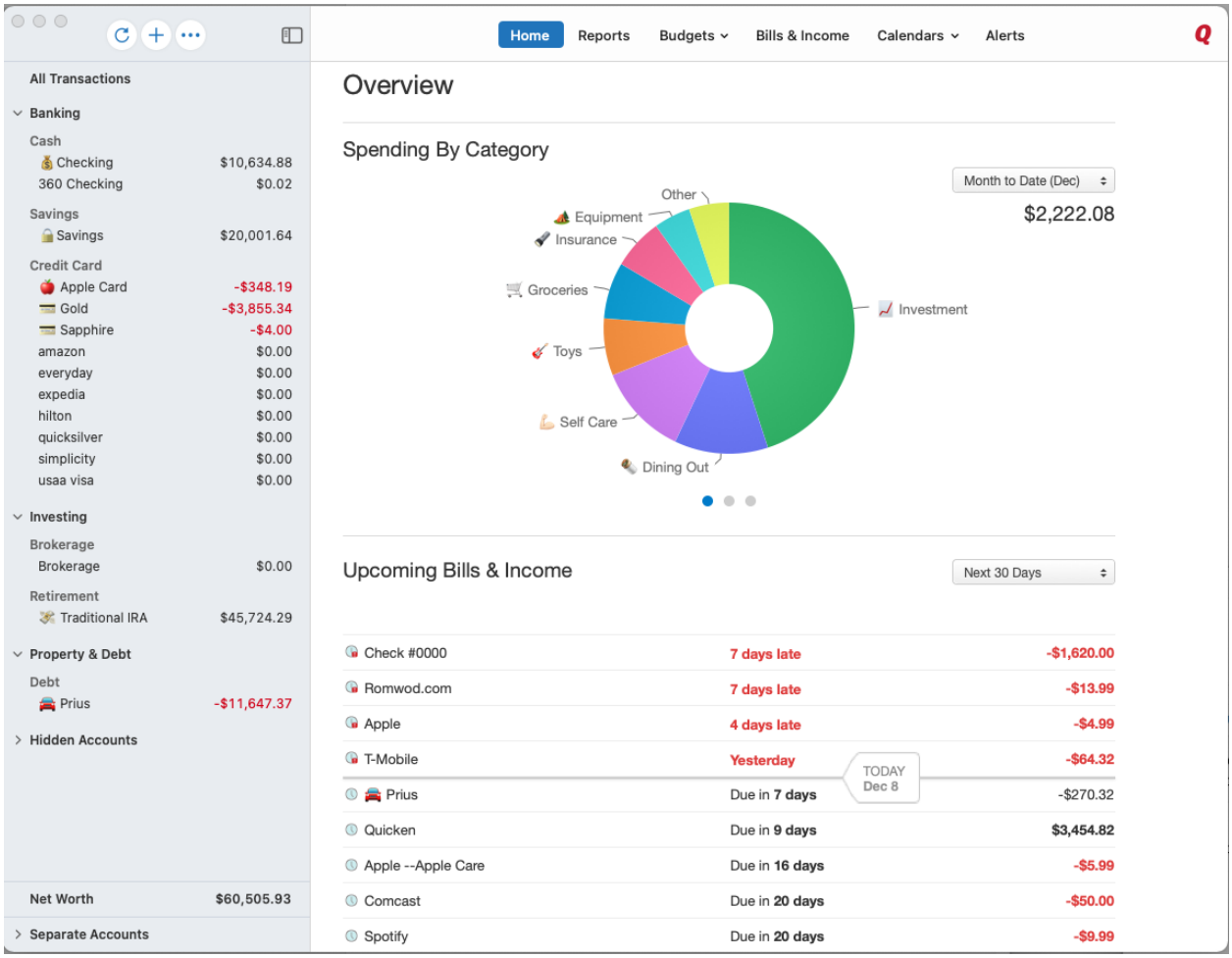


Рисунок 1.4 – Перегляд транзакцій у Quicken

Окрім бюджетів, Quicken містить цілий набір інструментів, які допоможуть планувати фінанси. Інші функції цього розділу допоможуть зменшити борги, скласти фінансовий план на все життя, відстежувати і прогнозувати податки на прибуток, а також встановити цілі для заощаджень. Ці інтерактивні інструменти поєднують у собі заповнення полів зі складними

калькуляторами та звітами. Вони відображають фінансові дані у вигляді графіків і таблиць.

Також входить до складу Quicken Classic входить Quicken Bill Manager [10], що є безкоштовною опцією. Користувач можете підключитися до своїх онлайн-платежів, щоб побачити дату сплати кожного рахунку, суму до сплати та останній здійснений платіж. Якщо хтось, кому користувач заборгував, не має доступу до інтернету, то він може додати його вручну і позначити платіж як сплачений. Більшість інших програмних засобів не мають можливості виконувати транзакції, а просто демонструють нагадування без можливості надсилати гроші. Quicken, однак, надсилає електронні платежі онлайн-платникам і паперові чеки, коли це необхідно.

Інструменти відстеження інвестицій Quicken перевершують ті, що пропонують інші програми для особистих фінансів. Окрім надзвичайно гнучкого настроюваного перегляду портфеля, Quicken включає «рентгенівський аналіз» активів [11] та численні інші способи оцінити міцність і можливу майбутню продуктивність портфеля користувача. Можна отримувати котирування в режимі реального часу, але користувачеві доведеться постійно оновлювати їх вручну. Портфелі також підтримують криптовалюти.

Варто зазначити, що Quicken - це іноді незручне поєднання старого і нового. Деякі елементи програми виглядають так, ніби їх не чіпали роками, тоді як інші виглядають оновленими та сучасними. Quicken Classic все ще використовує меню "Файл" Windows на додаток до новішої панелі інструментів, і користувачеві дійсно доведеться використовувати обидва, щоб побачити усі опції. Навіть якщо увімкнути опцію "Використовувати великі шрифти" в меню "Вигляд", деякі списки може бути важко читати. Програма настільки багатофункціональна, що потрібен певний час, щоб ознайомитися з усіма її можливостями. Але це не заважає керувати фінансами, а навпаки, дає змогу більш потужно розпоряджатися ними.

Окрім свого віку, Quicken багато в чому вирізняється з-поміж інших програм для планування бюджету та управління грошима. Він має доволі зручні інструменти для відстеження інвестицій. Як уже згадувалося, лише Quicken Classic дозволяє оплачувати рахунки безпосередньо з програми. Його безкоштовні ресурси демонструють себе з кращого боку. Крім того, Quicken – один з небагатьох додатків, що пропонує різноманітні інструменти податкового планування.

Хоча Quicken Classic - це програма, що більшою мірою призначена для настільних комп'ютерів, у неї є й версія для смартфонів - мобільний додаток Quicken [12].

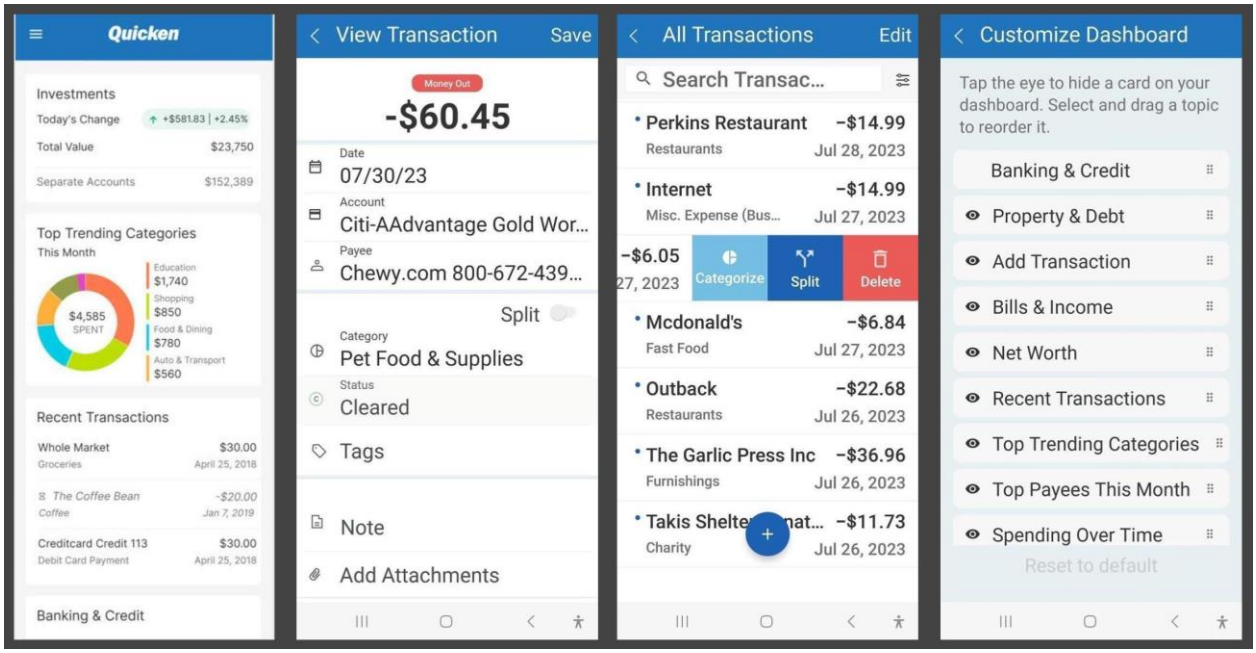
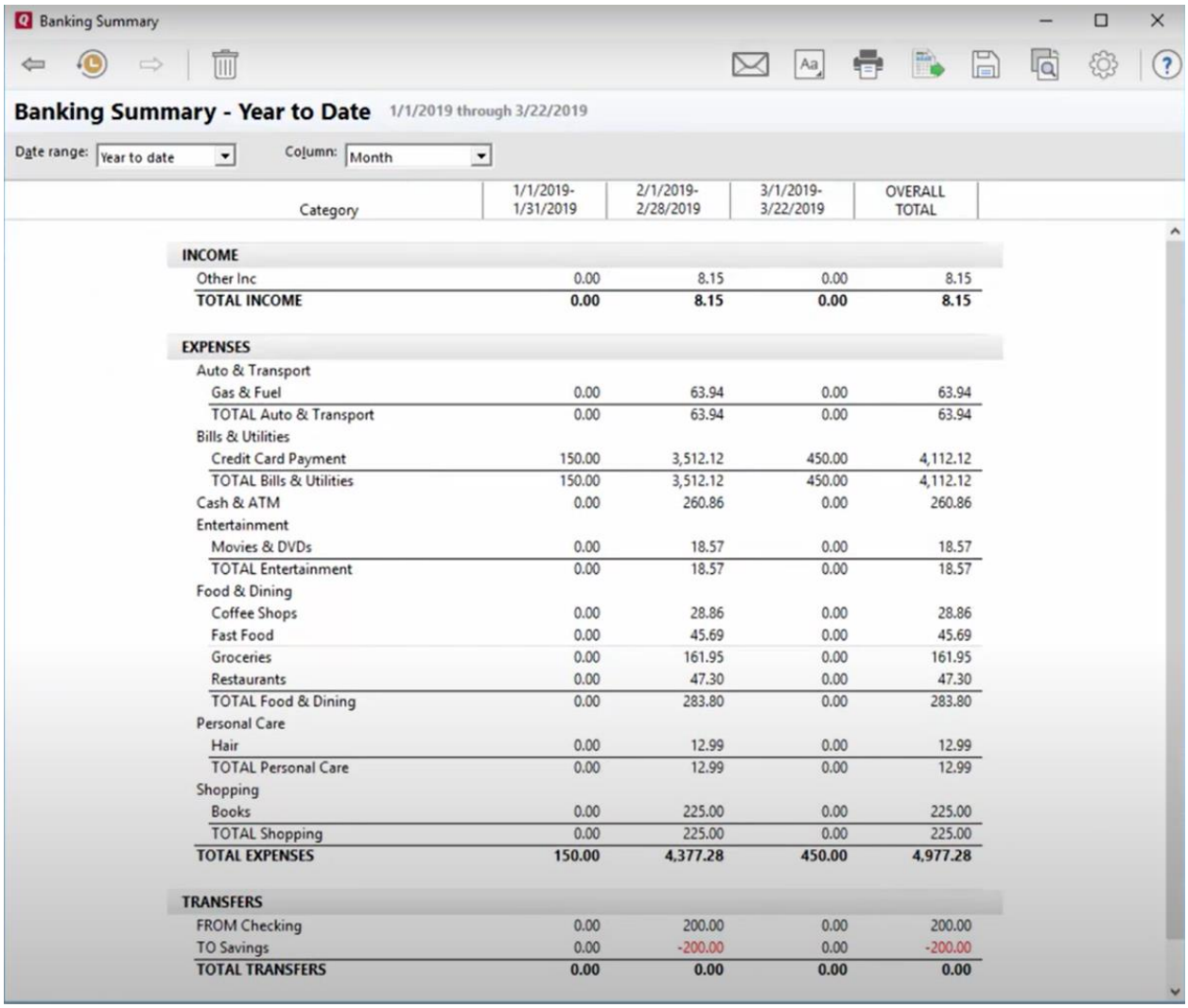


Рисунок 1.5 – Мобільний додаток Quicken

Мобільний додаток Quicken для Android та iPhone достатньо непоганий, але винятковим його назвати складно. Він також має урізаний набір інструментів, порівняно з десктопною версією, включаючи відстеження транзакцій та інвестицій, формування звітів, а також обмежене управління бюджетом і рахунками.

В той же час Quicken має безліч звітів, що легко налаштовуються. Звітування охоплює кожен елемент користувацьких фінансів, включаючи витрати, податки та інвестиції. Також наявний звіт "Дохід за цінними паперами", який надає детальний огляд дивідендів, відсотків та інших доходів, отриманих від цінних паперів.


Banking Summary

Banking Summary - Year to Date 1/1/2019 through 3/22/2019

Date range: Year to date Column: Month

Category	1/1/2019-1/31/2019	2/1/2019-2/28/2019	3/1/2019-3/22/2019	OVERALL TOTAL
INCOME				
Other Inc	0.00	8.15	0.00	8.15
TOTAL INCOME	**0.00**	**8.15**	**0.00**	**8.15**
EXPENSES				
Auto & Transport				
Gas & Fuel	0.00	63.94	0.00	63.94
TOTAL Auto & Transport	0.00	63.94	0.00	63.94
Bills & Utilities				
Credit Card Payment	150.00	3,512.12	450.00	4,112.12
TOTAL Bills & Utilities	150.00	3,512.12	450.00	4,112.12
Cash & ATM	0.00	260.86	0.00	260.86
Entertainment				
Movies & DVDs	0.00	18.57	0.00	18.57
TOTAL Entertainment	0.00	18.57	0.00	18.57
Food & Dining				
Coffee Shops	0.00	28.86	0.00	28.86
Fast Food	0.00	45.69	0.00	45.69
Groceries	0.00	161.95	0.00	161.95
Restaurants	0.00	47.30	0.00	47.30
TOTAL Food & Dining	0.00	283.80	0.00	283.80
Personal Care				
Hair	0.00	12.99	0.00	12.99
TOTAL Personal Care	0.00	12.99	0.00	12.99
Shopping				
Books	0.00	225.00	0.00	225.00
TOTAL Shopping	0.00	225.00	0.00	225.00
TOTAL EXPENSES	**150.00**	**4,377.28**	**450.00**	**4,977.28**
TRANSFERS				
FROM Checking	0.00	200.00	0.00	200.00
TO Savings	0.00	-200.00	0.00	-200.00
TOTAL TRANSFERS	**0.00**	**0.00**	**0.00**	**0.00**

Рисунок 1.6 – Формування звіту у Quicken

Quicken Classic поставляється в трьох рівнях: Deluxe ($71 на рік), Premier ($95 на рік) або Business and Personal ($131 на рік) [13]. Deluxe дозволяє створювати бюджети, управляти боргами, планувати вихід на пенсію та організовувати фінанси для подачі податкових декларацій. Premier додає інструменти для інвестування, податкові звіти та можливість відстежувати і

оплачувати рахунки. Business and Personal додає інструменти для управління малим бізнесом та орендою нерухомості.

Загалом, серед плюсів зазначеного програмного рішення можна зазначити широкий набір інструментів для керування особистими фінансами, гнучке та глибоке відстеження транзакцій. Недоліки проявляються здебільшо в залежності від десктопного додатку, а також невиваженому користувацькому інтерфейсі.

1.3.2 YNAB

YNAB [14], що розшифровується як «You Need a Budget» (Вам потрібен бюджет), - це додаток для ведення особистих фінансів з метою навчити користувача розумно витрачати гроші, щоб він міг якомога більше заощаджувати. Він має круту криву навчання і вимагає регулярної уваги, але кінцевою винагородою є гнучкий щомісячний план витрат, пристосований для досягнення поставлених фінансових цілей. Розробники впевнено стверджують, що цей додаток може змінити ставлення користувача до грошей.

YNAB є рішенням для укладання бюджету - зрештою, це закладено в назві програми. Користувач встановлює цільові показники витрат: скільки грошей потрібно на оплату рахунків (наприклад, електроенергії та інтернету), потреби (категорії з більшою варіативністю, такі як продукти та медичні витрати) та бажання (необов'язкові витрати, такі як ресторани, розваги та пожертви). Але користувач не може призначити суми для цих категорій, поки не отримає дохід, наприклад, зарплату. Транзакції або імпортуються з онлайн-банку та рахунків кредитних карток, або вводяться вручну.

YNAB базується на філософії під назвою "нульове бюджетування", що означає, що користувач розподіляє гроші за категоріями доти, доки всі вони не закінчаться. Насправді, це перше з чотирьох правил YNAB, які лежать в основі його дизайну:

а) дайте кожній гривні роботу;

б) прийміть свої справжні витрати (перетворіть великі, нечасті витрати на керовані частини);

в) тримайте удар (якщо перевитрачаєте в якійсь категорії, скоригуйте витрати)

г) розподіляйте гроші за часом (витрачайте менше, ніж заробляєте, і врешті-решт ви витрачатимете гроші, які заробили місяць або більше тому).

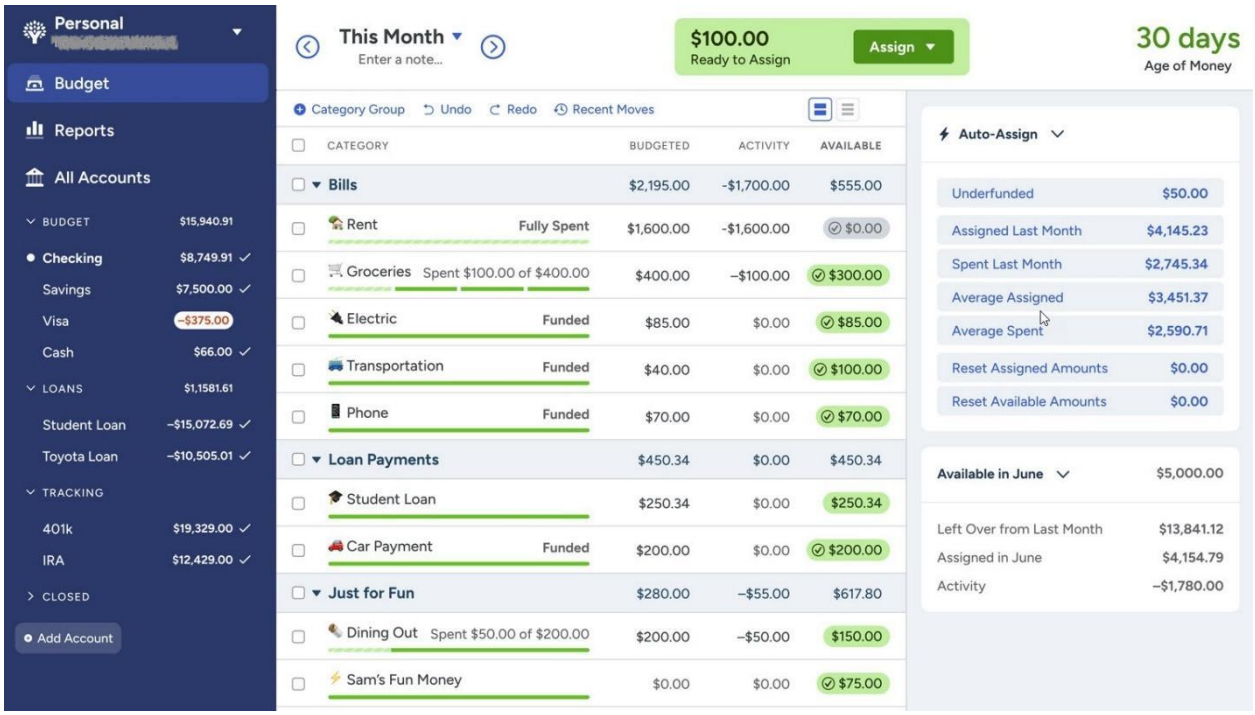


Рисунок 1.7 – Огляд інтерфейсу YNAB

Типові бюджети, зазвичай, ґрунтуються на прогнозах. Користувач оцінює свої майбутні доходи і витрати, а додатки показують йому, де його фактичний прихід і відтік грошей відносно поставленої мети. Цього може бути достатньо для людей, які просто хочуть бачити свій прогрес і не хочуть витрачати багато часу на складання бюджету.

YNAB також відрізняється від конкурентів тим, що не включає в себе оплату рахунків, відстеження інвестицій та управління цілями. З іншого боку, уся робота з YNAB полягає у встановленні та досягненні всеосяжних фінансових цілей. Користувач не просто каже, що хочете заощадити гроші на

відпустку і дозволяє додатку відстежувати прогрес; він навчаєте YNAB, як всі його витрати відповідають поставленим цілям і як його доходи і витрати змінюються протягом місяця. Таким чином, YNAB є хорошим вибором для людей з непередбачуваними доходами та витратами, наприклад, для тих, хто працює на фрілансі. YNAB надає інструменти, які допоможуть точно налаштувати свій бюджет. Але якщо користувачеві потрібна оплата рахунків, відстеження інвестицій і більш традиційні способи встановлення фінансових цілей, то більш зручною опцією може бути вищеописаний Quicken Classic.

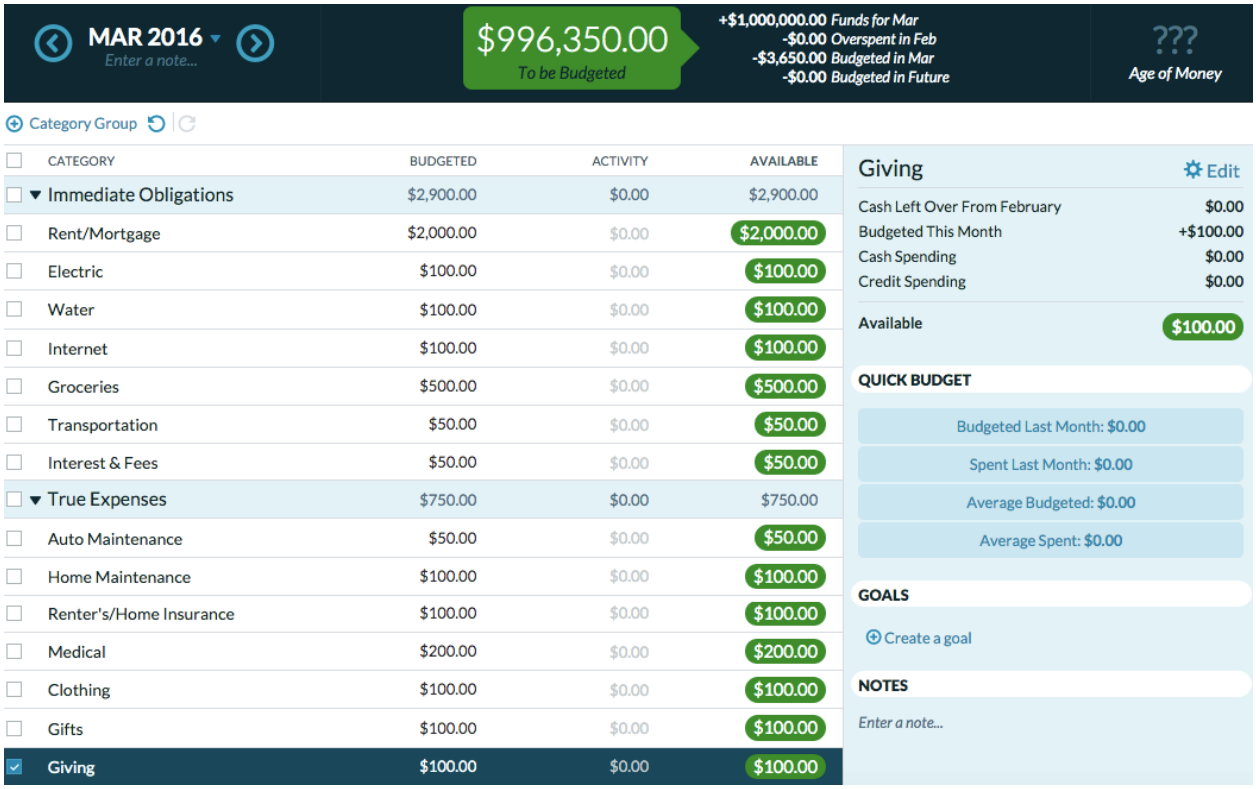


Рисунок 1.8 – Перегляд транзакцій у YNAB

На відміну від Quicken Classic, який надає численні звіти, що налаштовуються, YNAB має лише три звіти в основній версії: "Витрати", "Чиста вартість" та "Доходи та витрати". У мобільному додатку їх лише два: "Чиста вартість" і "Вік грошей", який вимірює, скільки часу в середньому гроші знаходяться на рахунках у діапазоні між зароблянням і витрачанням.

Рисунок 1.9 – Формування звітів у YNAB

Механіка використання YNAB не така вже й складна, якщо скористатися підтримкою та навчальними послугами додатку. З самого початку додаток є дружнім і простим у використанні. Він вивчає спосіб життя користувача і фінанси, як це робить більшість аналогічних додатків, коли ви починаєте користуватися ними, і створює початковий список категорій, які будуть застосовуватися до кожної статті ваших доходів і витрат. Також можна додавати та редагувати ці категорії в будь-який час.

Якщо користувач хоче імпортувати транзакції зі свого банківського рахунку, кредитних карток та інших фінансових рахунків, йому потрібно підключитися до них, увійшовши в кожен обліковий запис через YNAB. YNAB використовує кілька сторонніх сервісів фінансових підключень, включаючи Plaid [15], щоб забезпечити надійність і безпеку з'єднань.

Під час налаштування набагато краще працювати із використанням міні-таблиці зі стовпчиками "Категорія", "Призначено" (реальні гроші, які було виділено), "Активність" (імпортовані або введені вручну транзакції) і "Доступно" (те, що залишилося). Ці цифри змінюються в міру того, як користувач витрачає передбачені бюджетом кошти.

При цьому вертикальна панель праворуч показує варіанти для кожної категорії, включаючи частоту (чи повторюється дохід або витрата щотижня, щомісяця, щороку і т.д.), суму, день місяця для транзакції і користувацькі уподобання щодо роботи з цією категорією в наступному місяці. YNAB також повідомить користувачеві, чи є в нього залишок готівки з попереднього місяця. Можна налаштувати YNAB на автоматичне призначення категорій на основі отриманих даних.

Після користування додатком протягом декількох місяців, користувач отримує поточне оновлення поведінки для кожної категорії, а також історичну інформацію про кожну з них.

Для того, щоб YNAB запрацював належним чином, може знадобитися деякий час. Можливо, вам користувачеві не вдасться досягти запланованих сум з першого разу, коли він налаштує свою фінансову структуру в YNAB. Можливо, це не вийде і вдруге, і втретє. І незалежно від того, як довго людина користується системою, їй, ймовірно, доведеться вносити корективи щомісяця.

Але чим довше вона користуєтеся YNAB і чим більше він дізнається про фінансові звички користувача, тим точнішими будуть цільові суми. Усвідомлення щомісячних витрат може призвести до більш дисциплінованих витрат.

Ще одна відмінність полягає в тому, що додатки YNAB для Android та iOS, привабливі та інтуїтивно зрозумілі, не настільки повнофункціональні, як основна версія.

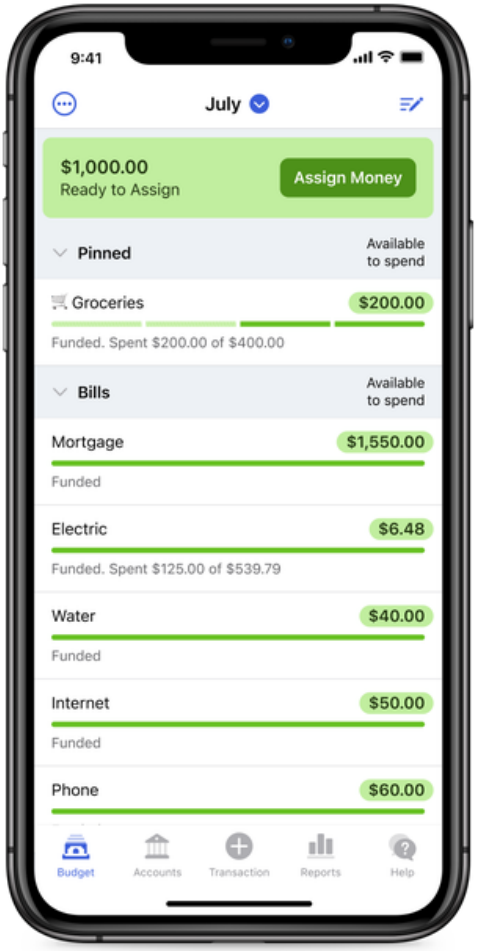


Рисунок 1.10 – Мобільний додаток YNAB

Розробники YNAB знають, що на перших порах користувачам будває важко, і тому додаток чудово справляється з наданням допомоги. Його навчальні інструменти представлені в живій, розважальній формі.

Довідкова система з можливістю пошуку містить добре написані та зрозумілі статті з інструкціями. Є можливість зв'язатися зі службою підтримки електронною поштою або в чаті й очікувати на відповідь протягом одного-двох робочих днів. Щотижня YNAB проводить безкоштовні семінари та вебінари, а також він має потужний канал на YouTube.

YNAB дорожчий за більшість фінансових додатків. YNAB коштує $14,99 на місяць або $109 на рік [17]. Існує 34-денна безкоштовна пробна версія. Розробники аргументують цей термін тим, що через місяць користувач тільки почне розуміти свій бюджет і витрати.

Вважається, що YNAB не є складною у використанні. Основна складність полягає в тому, щоб зрозуміти філософію, яка лежить в його основі, і перетворити її на розумний фінансовий менеджмент. Серед переваг інструменту можна виділити те, що він створений на основі ефективних ідей бюджетування, а його гнучкість підвищує шанси на успішне керування фінансами. Також пропонуються потужна підтримка та навчальні матеріали. Серед недоліків треба зазначити на необхідність приділення часу для опанування додатку та розуміння його концепції, а також відсутність опцій відстеження інвестицій та оплати рахунків.

1.3.3 PocketGuard

Серед вищеописаних додатків для особистих фінансів, PocketGuard [18] - це щось на кшталт майстра на всі руки. Він допомагає керувати різними частинами особистих фінансів, такими як бюджет, борги, чистий капітал, рахунки тощо. Він має непоганий інтерфейс і потужні ресурси підтримки та довідки.

Для створення облікового запису PocketGuard потрібен номер телефону. Він не вимагає імені і навіть не підтверджує вашу адресу електронної пошти, але надсилає шестизначний код автентифікації на введений номер телефону.

На початку роботи PocketGuard робить те, чого не роблять його конкуренти. Він перераховує все, що повинен робити користувач, перш ніж почати користуватися додатком, наприклад, підключитися до своїх фінансових рахунків в Інтернеті, щоб у нього були дані для роботи, а також додати свої регулярні зарплати і будь-які інші доходи. Додаток також пропонує встановити категорії та суми бюджету; є список за замовчуванням, і можна додати свій власний. PocketGuard просить переглянути всі періодичні витрати, які він витягує з імпортованих транзакцій, наприклад, підписки та інші рахунки, і вручну додати ті, яких не вистачає.

Інтерфейс додатку доволі швидкий, він використовує привабливі іконки скрізь і фотографічні зображення на сторінках з деталями рахунків. Альтернативне меню навігації веде до контенту, наприклад, сповіщень, яких багато і вони дуже корисні. Тим не менш, у PocketGuard немає жодної окремої характеристики, яка б робила його кращим за інші універсальні додатки для планування бюджету та управління фінансами.

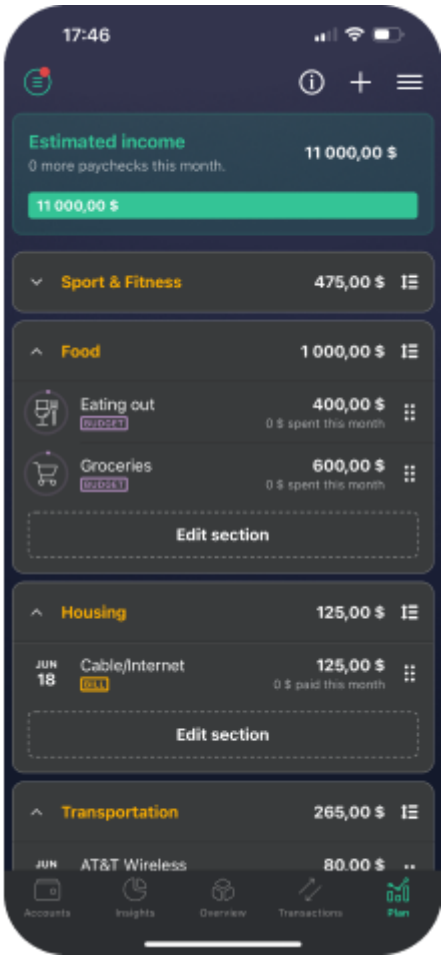


Рисунок 1.11 – Зовнішній вигляд PocketGuard

PocketGuard дуже якісно керує транзакціями, хоча й упускає деякі деталі. Наприклад, у в деяких програмах можна позначати витрати як такі, що пов'язані з податками. Натомість PocketGuard дозволяє створювати хештеги для транзакцій (наприклад, #streamingservices, #vacation) у полі Notes, щоб

згрупувати пов'язані витрати. Також у PocketGuard є міні-звіт, присвячений хештегам.

PocketGuard автоматично вводить деяку інформацію, пов'язану з транзакціями, наприклад, дату, ім'я одержувача платежу та категорію, яку програма вгадала. Користувач може відредагувати будь-яке з цих полів, прикріпити зображення, розділити транзакцію і позначити її як таку, що ігнорується в розрахунках додатку. Якщо під час імпорту транзакція була визначена як повторювана, на сторінці з'явиться відповідне посилання. Також можна перетворити разові транзакції на рахунки.

У PocketGuard є кілька так званих "Інсайтів", які дуже схожі на звіти. Це кругові діаграми та списки, які показують, наприклад, на що ви витрачаєте кошти і яким транзакціям ви присвоїли хештеги. Він добре справляється з порівнянням витрат за місяць, як у вигляді кругової діаграми, так і у вигляді списку.

Додаток надає користувачеві традиційні інструменти бюджетування. Можна вказати програмі, скільки, на вашу думку, потрібно витратити в тій чи іншій категорії, а вона відстежує, скільки ви вже витратили і скільки залишилося.

На сторінці кожної категорії є 12-місячний лінійний графік, який дозволяє порівняти поточні витрати з попередніми місяцями. Він також показує список транзакцій, пов'язаних з цією категорією. Існує категорія "Все інше" для покупок, які не були класифіковані. Невитрачені бюджетні гроші не переносяться на наступний місяць.

Число, позначене як "У кишені", показує, скільки грошей залишилося витратити після вирахування витрат та інших рахунків і заощаджень з вашого доходу.

PocketGuard має ще кілька інструментів для особистих фінансів, жоден з яких не є чимось особливим, але приємно мати їх всі в одному додатку. Тут є функція узгодження рахунків. Іншими словами, компанія може, спробувати знизити рахунки, хоча вона забирає 40% ваших заощаджень, якщо це вдається,

і нічого, якщо не вдається. У Rocket Money також є так звана послуга скасування підписки, що пропонує фактично скасувати підписку за користувача.

План погашення боргу в PocketGuard допоможе розробити стратегію скорочення боргу. У PocketGuard є два варіанти: спочатку погасити борг з найбільшими відсотками або з найменшим залишком. Можна встановлювати цілі та прив'язувати їх до рахунків для переказу та відстеження.

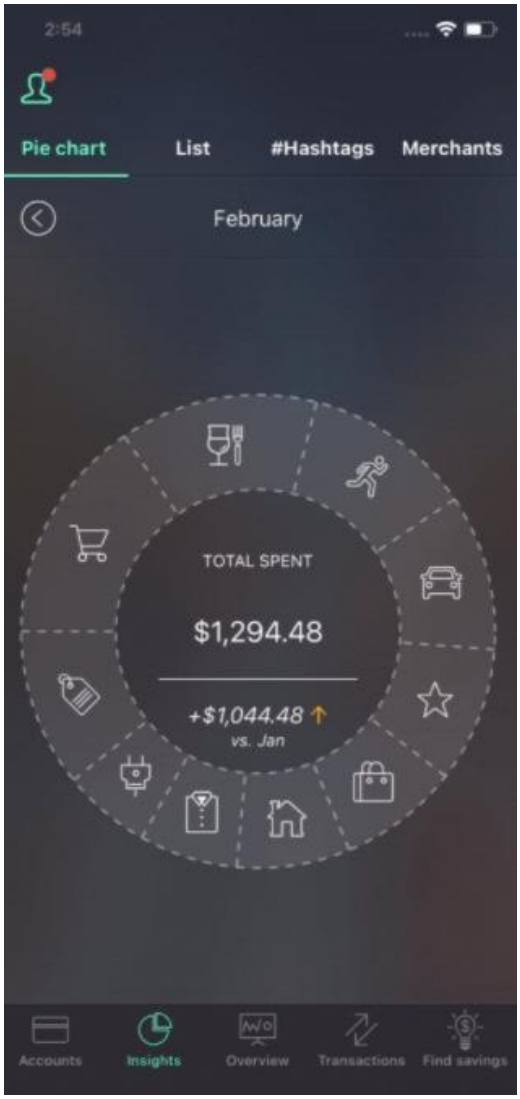


Рисунок 1.12 – Діаграма витрат PocketGuard

PocketGuard відстежує введені користувачем (або знайдені) рахунки, зіставляє їх з вхідними платежами і показує історію платежів на сторінці з

деталями рахунків. Також можна вводити рахунки вручну і позначати їх як сплачені. Автоматичні нагадування допоможуть залишатися в курсі справ.

Додаток розраховує чисту вартість на основі активів і зобов'язань, з якими пов'язаний користувач, наприклад, банківських рахунків і будь-яких кредитів, таких як іпотека. Користувач має ввести всю цю інформацію самостійно.

PocketGuard має багато хорошої довідкової інформації. В Центрі допомоги можна знайти велику кількість покрокових інструкцій з використанням анімації.

Платна версія PocketGuard Plus коштує $12,99 на місяць або $74,99 на рік. На сторінці з цінами вказано, що щомісячний тариф зі знижкою становить $19,99 на місяць [19], що є дуже високим показником. Раніше існувала безкоштовна версія PocketGuard, але наразі вона недоступна. Коли реєструється обліковий запис, можна побачити можливість "пропустити" оплату і створити безкоштовний обліковий запис, але практично всі функції заблоковані для використання, поки користувач не заплатить або не підпишеться на безкоштовну семиденну пробну версію.

Серед плюсів зазначеного програмного продукту можна відзначити непогане юзабіліті, креативний дизайн, зручне управління транзакціями та бюджетування, а також непогану довідку. Мінуси проявляються у високій ціні та відсутності кредитного рейтингу та інструментів відстеження інвестицій.

1.4. Висновки по проведеному аналізу існуючого програмного забезпечення для обліку фінансів

Дослідження показали, що Quicken, YNAB і PocketGuard є достатньо популярними сервіси для управління особистими фінансами, кожен з яких пропонує свій підхід до бюджетування та відстеження витрат.

Quicken - це один із найстаріших застосунків для фінансового обліку, який надає обширний функціонал для детального управління фінансами. З його допомогою можна відстежувати банківські рахунки, інвестиції, кредити і

навіть планувати пенсійні накопичення. Однак такий багатий набір функцій супроводжується складним інтерфейсом, який може виявитися непростим для розуміння, особливо для нових користувачів. Потрібен час і зусилля, щоб освоїти всі можливості програми і налаштувати її під свої потреби. Крім того, Quicken пропонує свої послуги на основі платної підписки, вартість якої досить висока порівняно з іншими рішеннями на ринку.

YNAB (You Need A Budget) фокусується на методології суворого бюджетування, закликаючи користувачів розподіляти кожну зароблену гривню усвідомлено. Додаток допомагає встановити фінансові цілі та стежити за їхнім досягненням, заохочуючи звичку регулярно відстежувати свої витрати й доходи. Незважаючи на інтуїтивно зрозумілий дизайн, YNAB вимагає від користувача дисципліни і постійної взаємодії з додатком для оновлення даних. Це може стати перешкодою для тих, хто не готовий приділяти багато часу управлінню бюджетом. Вартість підписки на YNAB також є суттєвою і може не підійти для людей з обмеженим бюджетом.

PocketGuard призначений для спрощення процесу відстеження щоденних витрат. Застосунок автоматично зв'язується з банківськими рахунками, аналізує фінансові потоки і надає інформацію про те, скільки грошей залишається для вільних витрат після обов'язкових платежів і заощаджень. Хоча інтерфейс PocketGuard більш доброзичливий для користувача, налаштування і синхронізація з банківськими рахунками можуть викликати складнощі, особливо якщо ваші фінансові установи не підтримуються додатком. Як і у випадку з іншими сервісами, повний доступ до функціоналу PocketGuard вимагає оформлення платної підписки.

Загальним недоліком усіх трьох додатків є відсутність української локалізації. Необхідність працювати з інтерфейсом додатку на іноземній мові може ускладнити розуміння функцій і знизити ефективність використання додатків. Поєднання мовного бар'єру з високою вартістю підписки і складністю освоєння робить ці сервіси менш привабливими для широкої аудиторії.

Також варто зауважити, що описані програмні засоби орієнтуються на західні країни та пропонують інтерфейси взаємодії з бінківськими та іншими фінансовими установами, а також опції по роботі з податками, що націлені на місцеве законодавство. Зазначені можливості не будуть діяти при використанні в Україні.

Таким чином, незважаючи на багатий функціонал і потенціал для поліпшення фінансового добробуту, зазначені програмні продукти можуть виявитися надто складними і дорогими для більшості користувачів. Відсутність підтримки української мови лише поглиблює проблему, наголошуючи на необхідності в більш простому і доступному рішенні для ефективного управління особистими фінансами.

1.5 Завдання та цілі роботи

Метою даної роботи буде створення веб-додатку для ефективного управління особистими фінансами, використовуючи сучасні технології розробки. Основні завдання будуть полягати у формулюванні детальних функціональних та нефункціональних вимог до системи, виборі оптимальних інструментів розробки, побудові чіткої та масштабованої архітектури проєкту, а також реалізації інтуїтивно зрозумілого та зручного інтерфейсу для кінцевих користувачів.

Крім того, значну увагу буде приділено впровадженню ефективних методів візуалізації даних та розробці формул для об'єктивної оцінки ефективності використання додатку. Планується інтегрувати бібліотеки для створення інтерактивних графіків та діаграм, що дозволять користувачам отримувати наочну та зрозумілу інформацію про їхні фінансові потоки.

Результатом роботи стане програмне рішення, яке надаватиме можливість додавання, редагування та перегляду фінансових транзакцій, формування детальних звітів та побудови інтерактивних графіків для аналізу фінансових даних. Таким чином, цілі роботи спрямовані на створення надійного та масштабованого застосунку, який зможе полегшити процес

управління фінансами для користувачів, сприяти підвищенню їх фінансової дисципліни та допомогти у прийнятті обґрунтованих фінансових рішень.

Таким чином, завдання та цілі роботи будуть спрямовані на створення високоякісного веб-додатку, який задовольнить потреби користувачів у зручному та ефективному управлінні фінансами, а також забезпечить надійність роботи системи.

2 ФОРМУЛЮВАННЯ ВІДОМОСТЕЙ ПРО ОБ'ЄКТ ДОСЛІДЖЕННЯ ПРИ РОЗРОБЦІ ВЕБ-ДОДАТКУ

2.1 Аналіз потенційної платформи розроблюваного програмного забезпечення та способу взаємодії з користувачем

З огляду на проведені дослідження, також варто звернути увагу і на те, що розглянуті програмні засоби для обліку особистих фінансів здебільшого представлено у вигляді десктопних і мобільних додатків. Хоча ці платформи пропонують широкий спектр функцій для управління особистими фінансами, вони не завжди виявляються найзручнішими та найдоступнішими для користувачів, які прагнуть гнучкості та простоти у фінансовому обліку.

Десктопні додатки вимагають встановлення на конкретний пристрій. Це означає, що доступ до ваших фінансових даних обмежений тим комп'ютером, на якому встановлено програму. Якщо ви перебуваєте поза домом або офісом, можливість оперативно перевірити баланс рахунку або внести нову транзакцію стає проблематичною. Мобільні додатки, представлені YNAB і PocketGuard, вирішують проблему мобільності, але стикаються з іншими обмеженнями. Маленький екран смартфона може ускладнювати сприйняття деталізованої фінансової інформації, а необхідність установлення застосунку на кожен новий пристрій створює додаткові складнощі.

Розробка веб-додатку для обліку особистих фінансів видається більш доцільним рішенням у сучасних умовах. Веб-додаток надає користувачам можливість доступу до своїх фінансових даних з будь-якого пристрою, чи то комп'ютера, чи то планшета, чи то смартфона, за наявності підключення до інтернету. Це усуває необхідність у встановленні спеціального програмного забезпечення та дає змогу працювати з даними безпосередньо через веб-браузер. Така універсальність особливо актуальна для людей, які ведуть активний спосіб життя і використовують різні пристрої в повсякденній діяльності.

Зберігання даних у хмарі є ще однією суттєвою перевагою веб-додатків. Хмарні сервіси забезпечують надійний захист і резервне копіювання інформації, що мінімізує ризик її втрати через поломку пристрою, його крадіжку або випадкове видалення даних. Користувачі можуть бути впевнені в збереженні своїх фінансових записів, незалежно від зовнішніх обставин. Крім того, хмарне зберігання полегшує синхронізацію даних між різними пристроями, забезпечуючи актуальність і цілісність інформації в будь-який момент часу.

Відсутність необхідності встановлення програмного забезпечення не тільки економить час користувачів, а й звільняє від турбот щодо сумісності з різними операційними системами та регулярних оновлень. Веб-додатки оновлюються автоматично на стороні сервера, надаючи всім користувачам доступ до останніх версій і нових функцій без додаткових дій з їхнього боку. Це спрощує процес використання і знижує технічні бар'єри, які можуть виникнути під час роботи з десктопними або мобільними додатками.

Важливо відзначити і той факт, що веб-додатки зазвичай більш гнучкі в питаннях локалізації та адаптації під різні мови і культурні особливості. Це робить їх більш доступними для широкої аудиторії, включно з користувачами, для яких відсутність української локалізації в розглянутих застосунках є суттєвою перешкодою. Можливість швидко впроваджувати нові мовні версії дає змогу веб-додаткам охоплювати більшу кількість користувачів і враховувати їхні специфічні потреби.

У світлі цих переваг розробка веб-додатка для обліку особистих фінансів видається не тільки сучаснішим, а й практичнішим підходом. Він поєднує в собі зручність доступу, безпеку, простоту використання та адаптивність до потреб користувачів. На відміну від складних і дорогих десктопних і мобільних додатків, веб-додаток здатний надати функціональність, необхідну більшості користувачів, без зайвих ускладнень і фінансових витрат.

Таким чином, незважаючи на функціональні можливості розглянутих аналогів, їхня обмеженість платформами та недоліки у зручності

використання підкреслюють необхідність пошуку альтернативних рішень. Веб-додаток виступає в ролі такого рішення, пропонуючи доступніший та ефективніший інструмент для управління особистими фінансами, який відповідає вимогам сучасного користувача та адаптується до його способу життя.

2.2 Визначення об’єкта і предмета досліджень

Проведений аналіз показав, що питання керування власними фінансами є достатньо актуальним, і він породжує попит на відповідні програмні рішення. Це не дивно, адже фінансова грамотність і здатність ефективно розпоряджатися своїми коштами дають змогу досягати поставлених цілей, уникати боргових пасток і забезпечувати фінансову стабільність. Однак багато людей стикаються з труднощами у відстеженні своїх доходів і витрат, що призводить до неоптимального використання ресурсів і фінансових проблем.

Ідея роботи полягає у створенні веб-додатку, який допоможе користувачам не тільки вести облік особистих фінансів, а й аналізувати фінансову ефективність своїх дій. Цей застосунок слугуватиме інструментом для розуміння фінансового стану користувача, надаючи аналітичні дані, виявляючи тенденції та інформацію для роздумів щодо оптимізації витрат і збільшення заощаджень. На відміну від наявних рішень, новий застосунок поєднуватиме в собі простоту використання та аналітичні можливості, адаптовані під індивідуальні потреби кожного користувача.

Мета роботи полягає у формулюванні напрямів досліджень, необхідних для розроблення цього веб-додатка. Це передбачає комплексний підхід, що включає вивчення сучасних технологій веб-розробки, методів опрацювання та візуалізації фінансових даних, а також аналізу користувацького досвіду. Дослідження спрямовані на створення додатка, який буде не тільки функціональним і надійним, а й інтуїтивно зрозумілим для користувача,

сприяючи підвищенню його фінансової грамотності та ефективності управління особистими фінансами.

Об'єктом досліджень є процес розроблення веб-додатка для обліку особистих фінансів із функціями аналізу фінансової ефективності. У межах цього процесу розглядаються різні аспекти розробки: від вибору технологічного стека та архітектури додатка до безпосередньої реалізації програмного продукту. Особлива увага приділяється способам опрацювання фінансової інформації та представлення її в зручній для користувача формі.

Предметом досліджень виступають методи і технології, що використовуються при розробці веб-додатків такого роду. Це охоплює дослідження алгоритмів аналізу фінансових даних, підходів до візуалізації інформації, а також вивчення призначених для користувача інтерфейсів і досвіду взаємодії з додатком.

Таким чином, робота спрямована на створення теоретичної та практичної бази для розроблення веб-додатку, що дасть змогу користувачам ефективно управляти своїми особистими фінансами та приймати обґрунтовані фінансові рішення. Результати досліджень сприятимуть розвитку технологій у сфері фінансових застосунків і підвищенню рівня фінансової грамотності користувачів.

2.3 Формулювання вимог до веб-додатку

Точне і повне формулювання вимог до системи є фундаментом успішного проєкту. Правильно сформульовані вимоги дають змогу розробникам і замовникам мати спільне розуміння цілей і функцій системи, що знижує ризик непорозумінь і збільшує ймовірність створення продукту, який відповідає очікуванням користувачів.

У процесі розроблення програми для фінансового обліку було визначено та функціональні та нефункціональні вимоги, спрямовані на створення ефективного та зручного інструменту для управління особистими фінансами.

Однією з основних функціональних вимог є надання користувачам можливості аналізувати співвідношення доходів і витрат за певні періоди часу. Система має давати змогу користувачам візуально оцінювати свою фінансову ефективність через графічне представлення даних. Для цього буде реалізовано функцію відображення графіка, що порівнює доходи та витрати за обрані періоди.

Користувачі зможуть вибирати період аналізу: день, тиждень або місяць. Крім того, буде передбачено можливість задавати довільний діапазон дат для гнучкішого і детальнішого аналізу фінансових даних. У разі зміни періоду або діапазону дат система автоматично оновлює графік, надаючи актуальну інформацію без затримок.

Ще однією важливою функціональною вимогою є забезпечення інтерактивності графіка. Користувачі будуть мати можливість вмикати або вимикати відображення числових значень на графіку, що дасть змогу налаштувати візуалізацію відповідно до їхніх уподобань. Цей функціонал підвищить зручність використання і надасть змогу користувачам краще розуміти свої фінансові показники.

У контексті нефункціональних вимог особливу увагу буде приділено зручності використання та інтуїтивності інтерфейсу. Система має бути розроблена таким чином, щоб користувачі без спеціальної підготовки могли легко взаємодіяти з нею. Елементи управління будуть розташовані логічно, а дизайн інтерфейсу буде зрозумілим і несуперечливим.

Продуктивність системи буде оптимізовано для забезпечення швидкого відгуку на дії користувача. При виборі нових параметрів фільтрації або періодів аналізу графік оновлюватиметься миттєво, що підвищить ефективність роботи і задоволеність користувачів.

Масштабованість системи дозволить їй ефективно працювати з великим обсягом даних. Незалежно від кількості введених транзакцій, система збереже високу продуктивність і стабільність.

Крім того, система буде розроблена з урахуванням кросплатформності та адаптивності інтерфейсу. Користувачі можуть працювати з нею на різних пристроях, включно з настільними комп'ютерами, планшетами та смартфонами. Інтерфейс автоматично адаптуватиметься до розміру екрана, забезпечуючи комфортну взаємодію незалежно від використовуваного пристрою.

Таким чином, ретельне формулювання функціональних і нефункціональних вимог до системи надасть змогу створити ефективний, надійний і зручний застосунок для фінансового обліку.

2.4 Вибір засобів розробки

Під час планування створення додатку одним із ключових етапів є вибір засобів розробки. Спочатку варто звернути увагу на вибір мови програмування і платформи. Використання C# [20] у поєднанні з .NET Core [21] видається оптимальним рішенням. C# є потужною об'єктно-орієнтованою мовою, яка надає широкі можливості для розроблення додатків різної складності. Платформа .NET Core, будучи кросплатформеною і відкритою, дає змогу створювати додатки, що працюють на різних операційних системах. Це забезпечить гнучкість у розгортанні та використанні програми на різних пристроях.

Для розробки веб-інтерфейсу доцільно використовувати ASP.NET Core MVC [22]. Цей фреймворк підтримує архітектурний шаблон Model-View-Controller [23], що сприяє розділенню логіки застосунку, користувацького інтерфейсу та даних. Такий поділ полегшує підтримку та масштабування застосунку в майбутньому. ASP.NET Core MVC також забезпечує високу продуктивність і вбудовану підтримку сучасного веб-функціоналу, що важливо для створення responsive-інтерфейсу.

При розробці клієнтської частини застосунку необхідно забезпечити інтерактивність і зручність використання. Для цього планується використовувати сучасні технології веб-розробки, такі як HTML5, CSS3 і

JavaScript. Для прискорення процесу розробки та забезпечення адаптивності інтерфейсу може бути задіяний фреймворк Bootstrap. Він надає готові компоненти та стилі, що дають змогу створювати сучасний дизайн без значних витрат часу.

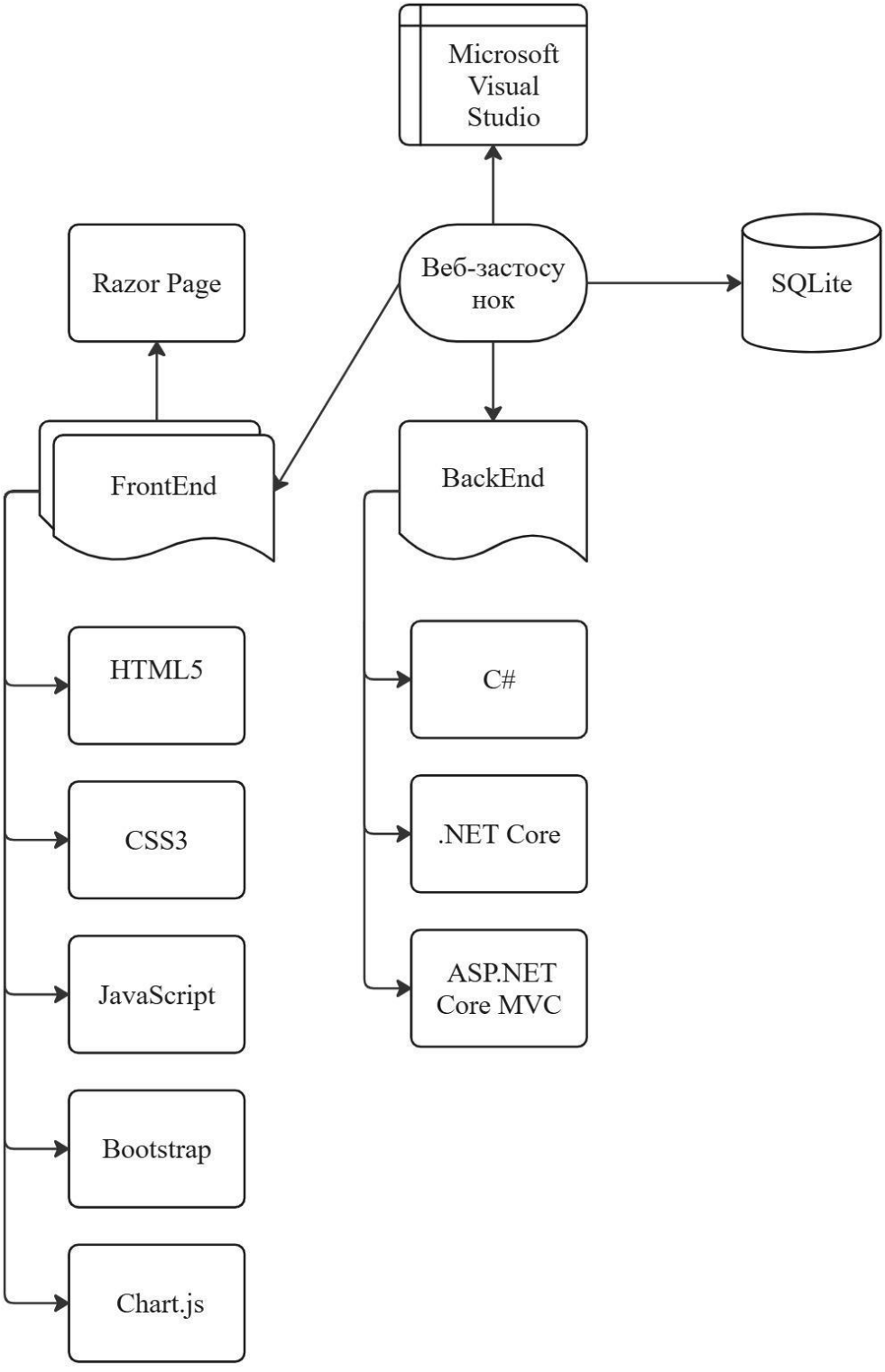


Рисунок 2.1 – Структура основних засобів розробки веб-застосунку

Для візуалізації фінансових даних і надання користувачам наочних інструментів аналізу буде використовуватися бібліотека Chart.js [24]. Вона дає змогу створювати різноманітні інтерактивні графіки та діаграми. Інтеграція Chart.js з ASP.NET Core забезпечить динамічне оновлення даних і поліпшить користувацький досвід під час роботи з фінансовою інформацією.

Зберігання та управління даними є критично важливими аспектами фінансового застосунку. Планується використовувати Entity Framework Core [25] як ORM (Object-Relational Mapping) для взаємодії з базою даних. Це дасть змогу працювати з даними на рівні об'єктів, уникаючи прямого написання SQL-запитів, що прискорить розробку і знизить ймовірність помилок. Як система управління базами даних розглядається використання SQLite.

Що стосується IDE, то Visual Studio буде використовуватися в якості основного інтегрованого середовища розробки. Вона надає потужні інструменти для створення, налагодження та тестування коду, а також має підтримку розширень, які можуть додатково спростити процес розробки.

Отже, обрані інструменти та технології мають забезпечити ефективне створення надійного та зручного застосунку для фінансового обліку. Ретельний вибір засобів розробки є запорукою успішної реалізації проєкту та його подальшого розвитку.

2.5 Структура веб-додатку

Структура розроблюваного веб-застосунку продумана такоим чином, щоб користувачі могли інтуїтивно взаємодіяти з ним, зручно отримувати інформацію та ухвалювати обґрунтовані фінансові рішення.

При запуску додатку демонструється головна сторінка, що вітає користувачів та слугує для наведення початкової інформації. Вона надає швидкий доступ до функції додавання нових транзакцій, в той час як меню хедера дозволяє перейти до інших сторінок.

Переміщуючись далі, користувачі стикаються зі сторінкою транзакцій, яка слугує основним інструментом для введення та управління фінансовими

операціями. На цій сторінці передбачено кнопку для додавання нових транзакцій. Крім того, сторінка надає можливість редагування та видалення вже введених транзакцій, що забезпечує гнучкість і точність обліку. Інтегровані фільтри дають змогу користувачам швидко знаходити потрібні записи за різними критеріями, як-от період часу, категорія або сума, що значно спрощує аналіз фінансових даних.

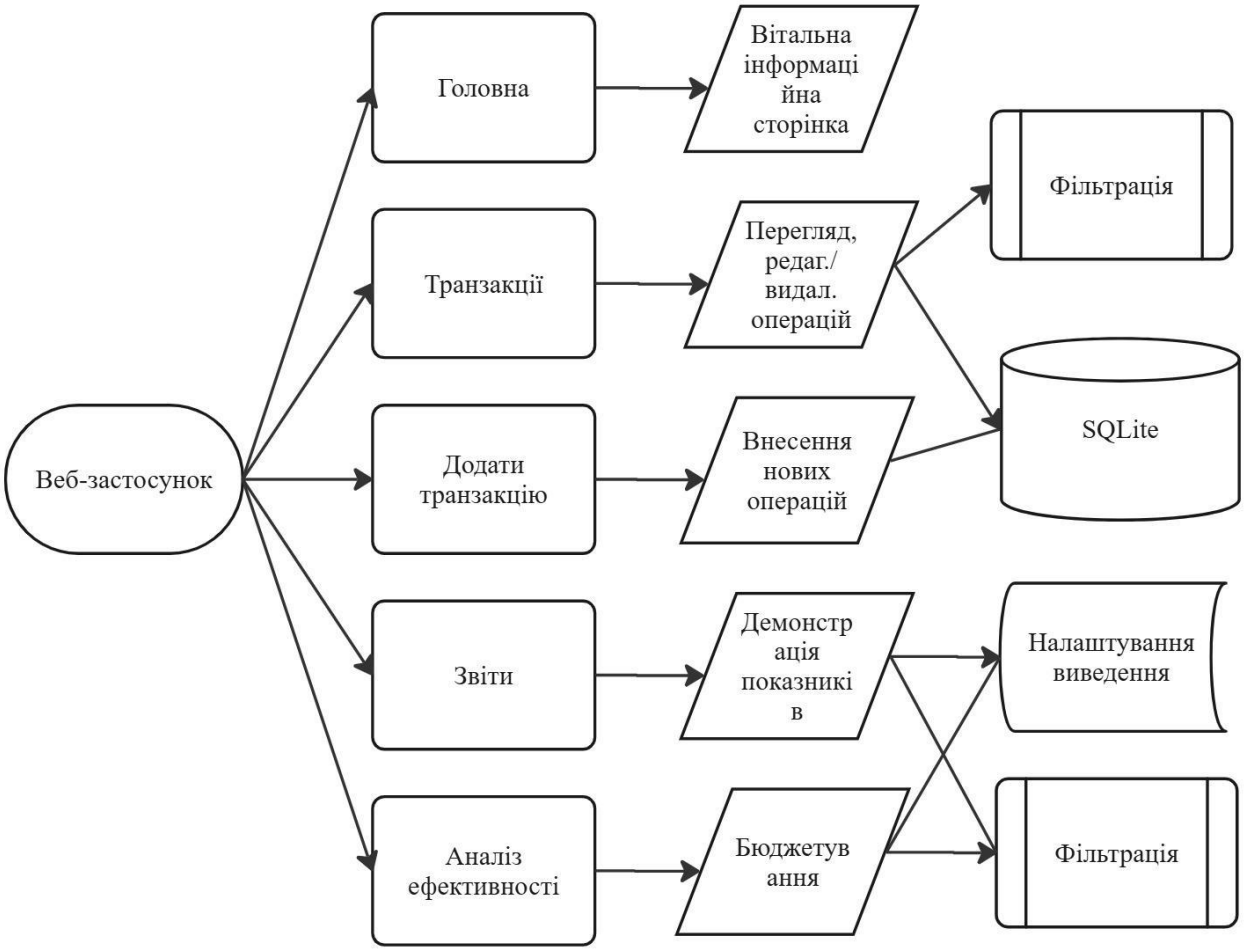


Рисунок 2.2 – Функціональна схема веб-застосунку

Далі слідує сторінка додавання транзакцій. На цій сторінці передбачено можливість додавання нових операцій, де можна вказати суму, категорію, дату та опис кожної угоди.

Наступною сторінкою є сторінка звітів, де користувачі можуть генерувати деталізовані звіти про свої фінанси. Тут реалізовано функції

вибору різних параметрів звітності, як-от часові рамки, категорії витрат і доходів. Сторінка звітів надає візуальні інструменти для аналізу, включно з графіками, діаграмами та таблицями, що дає змогу користувачам інтерпретувати свої фінансові показники та виявляти тенденції.

Особливу увагу в структурі додатка приділено сторінці аналізу ефективності, яка надає користувачеві аналіз співвідношення доходів і витрат за обрані періоди. На цій сторінці реалізовано інтерактивні графіки, що дають змогу користувачам візуально оцінювати свою фінансову ефективність. Можливість вибору різних періодів, а також можливість завдання довільного діапазону дат забезпечують гнучкість і точність аналізу. Користувачі можуть вмикати або вимикати відображення числових значень на графіках, що дає змогу налаштовувати візуалізацію відповідно до особистих уподобань.

2.6 Алгоритм роботи веб-застосунку

Веб-додаток розроблюється з метою забезпечення ефективного управління особистими фінансами користувачів. Його робота ґрунтуватиметься на архітектурі, що забезпечить взаємодію між клієнтською та серверною частинами, гарантуючи надійність та зручність використання.

Процес починається із взаємодії користувача з інтерфейсом додатка, який надає інтуїтивно зрозумілі засоби для введення та перегляду фінансових даних.

Користувач на початку роботи потрапляє на головну інформаційну сторінку, де від може перейти до додавання транзацій. Також він може перейти на сторінку транзакцій, де йому буде представлена зведена інформація про поточний фінансовий стан. Тут відображаються загальні показники доходів і витрат, а також надаються інструменти для вибору часового періоду аналізу. Взаємодія з елементами інтерфейсу, такими як кнопки вибору періоду (доба, тиждень, місяць) або вибір довільного діапазону дат, ініціює надсилання відповідних запитів на сервер. Ці запити обробляються серверною частиною додатка, яка відповідає за обробку логіки та взаємодію з базою даних.

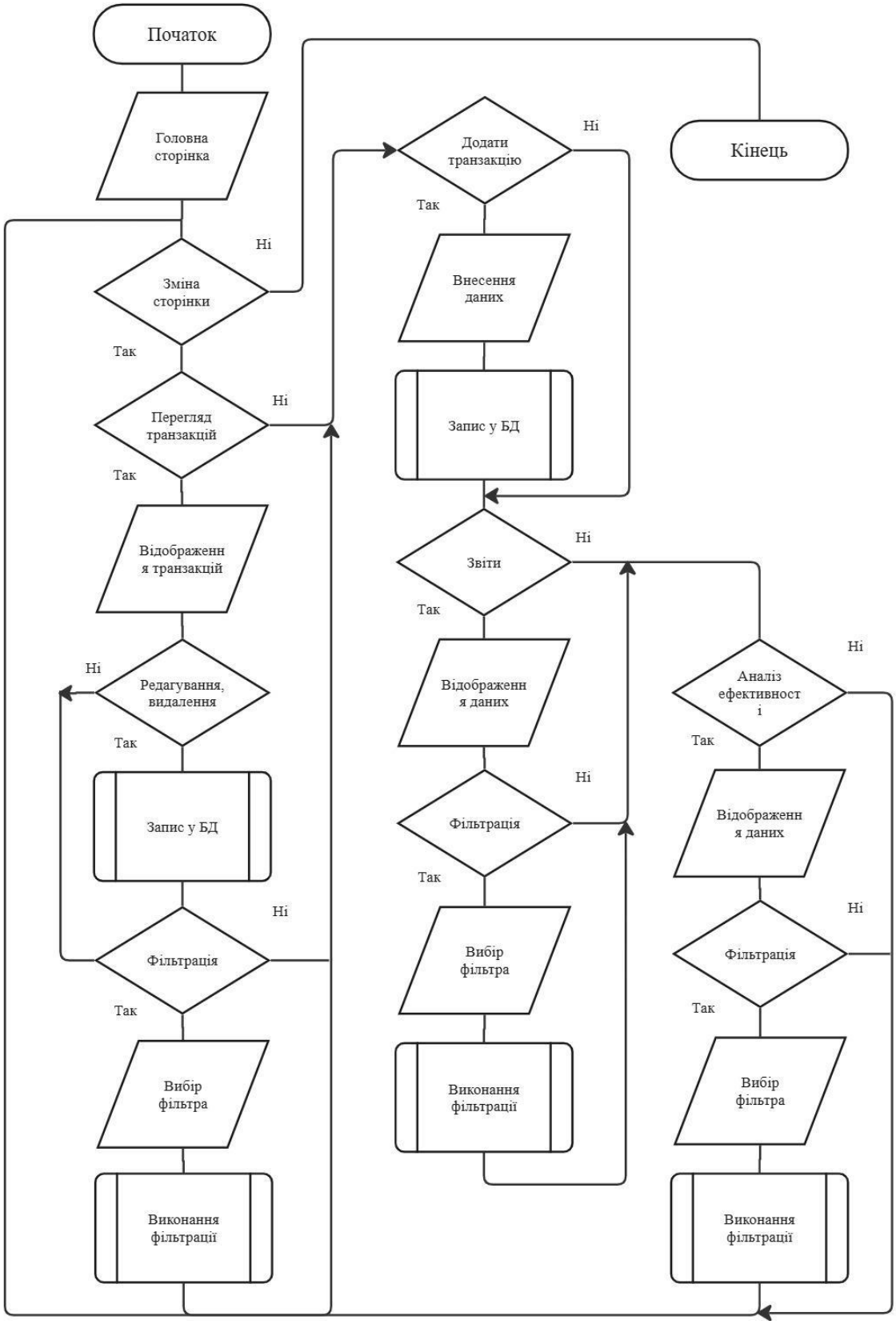


Рисунок 2.3 – Алгоритм роботи веб-застосунку

Користувач має можливість додавати нові транзакції через форму введення даних, де він зазначає суму, категорію, дату та опис операції. Ці дані

надсилаються на сервер, де бекенд обробляє їх, виконує валідацію та зберігає в базі даних. За необхідності користувач може редагувати або видаляти вже введені транзакції. Усі зміни відображаються на візуальних елементах інтерфейсу завдяки використанню асинхронних запитів і динамічного оновлення даних.

На сторінці аналізу ефективності використання фінансових засобів демонструються графіки, що надають користувачеві візуальне уявлення щодо його фінансової активності, даючи змогу швидко оцінити співвідношення доходів і витрат за обраний період. У разі вибору користувачем певного періоду часу або діапазону дат, сервер виконує запити до бази даних через Entity Framework Core, витягуючи необхідні транзакції та агрегуючи дані щодо доходів і витрат. Отримані дані потім передаються назад на клієнтську сторону, де за допомогою JavaScript і бібліотеки Chart.js формуються інтерактивні графіки.

У разі невиконання вибору певного режиму робота з веб-додатком завершується.

2.7 Математична модель оцінки ефективності створюваного веб-додатку

Для оцінки ефективності функціонування розроблюваного веб-додатку пропонується наступна математична модель:

$$ES = \left(\frac{\Delta S}{S_0} + \frac{\Delta I}{I_0} - \frac{\Delta E}{E_0}\right) * \left(\frac{\Delta T}{T_0}\right) \quad (2.1)$$

де ES – показник комплексного оцінювання ефективності використання створеного веб-додатку,

ΔS - приріст заощаджень після використання застосунку,

ΔI - збільшення доходів після використання застосунку,

ΔE - скорочення витрат після використання застосунку,

ΔT - економія часу, витраченого на управління фінансами завдяки застосунку,

S_0 – початкові заощадження,

I_0 – початкові доходи,

E_0 – початкові витрати,

T_0 – початкові витрати часу.

Після закінчення процесів розробки та тестування, використання зазначеної математичної моделі дозволить провести оцінку ефективності використання веб-додатку та отримати вимірювані характеристики, що дадуть змогу зробити об’єктивні висновки про корисність використання програми для обліку фінансів.

3 ДЕМОНСТРАЦІЯ РОЗРОБЛЕНОГО ВЕБ-ДОДАТКУ ДЛЯ ОБЛІКУ ОСОБИСТИХ ФІНАНСІВ З АНАЛІЗОМ ФІНАНСОВОЇ ЕФЕКТИВНОСТІ

3.1 Програмна реалізація створеного веб-застосунку

Програмна частина проєкту спирається на принципи розроблення веб-застосунків на базі ASP.NET Core MVC, що забезпечує високу продуктивність, коду. Основна ідея полягає в чіткому поділі логіки, інтерфейсу та даних, а також у використанні перевірених методик і патернів проєктування, які гарантують розширюваність і масштабованість проєкту.

Що стосується методології побудови проєкту, то в основі програми лежить архітектурний шаблон MVC, який реалізовано засобами ASP.NET Core MVC. Цей шаблон розділяє проєкт на три основні компоненти:

а) «Model» - визначає структуру даних і бізнес-логіку. У проєкті для опису фінансових транзакцій, категорій витрат і доходів, а також даних про користувачів використовується набір C# класів-моделей, розташованих у папці *Models/Transaction.cs, Models/Category.cs* тощо;

б) «View» - відповідає за відображення даних і формування користувацького інтерфейсу. Подання розташовані в папках *Views/Home, Views/Transactions, Views/Reports* тощо, де кожен Razor-файл (*.cshtml*) містить логіку візуалізації HTML-контенту з використанням даних, що передаються контролерами;

в) «Controller» - визначає поведінку застосунку під час надходження HTTP-запитів, обробляє їх і вибирає відповідні подання для відповіді. Контролери розташовані в папці *Controllers/HomeController.cs, Controllers/TransactionsController.cs, Controllers/ReportsController.cs* тощо.

Наприклад, у *Controllers/TransactionsController.cs* можна знайти методи *Index(), Create(), Edit()* і *Delete()*, що відповідають за відображення списку

транзакцій, створення нового запису, редагування наявного та видалення, відповідно.

Для роботи з базою даних застосовується Entity Framework Core - ORM, що спрощує взаємодію з реляційними БД. Структура даних, оголошена в моделях, проектується на таблиці в базі даних. Контекст даних (*Data/FinanceOblikContext.cs*) визначає набір DbSet-властивостей, що забезпечують доступ до відповідних таблиць.

Прикладний код для отримання, додавання або оновлення транзакцій може бути знайдений, наприклад, у методах контролера транзакцій. Наприклад, метод Create у *TransactionsController.cs* приймає модель транзакції з подання і зберігає її в базі через контекст даних:

```
[HttpPost]
[ValidateAntiForgeryToken]
public async Task<IActionResult> Create(Transaction transaction)
{
    if (ModelState.IsValid)
    {
        _context.Transactions.Add(transaction);
        await _context.SaveChangesAsync();
        return RedirectToAction(nameof(Index));
    }
    return View(transaction);
}
```

Що стосується візуалізації даних та інтеграції фронтенду, то для відображення графіків і діаграм використовується клієнтська бібліотека Chart.js. Її інтеграція реалізована через Razor-подання. У поданні *Views/Reports/Index.cshtml* підключаються скрипти Chart.js, а потім визначається <canvas> елемент, куди за допомогою JavaScript передаються дані з контролера.

Приклад використання Chart.js можна знайти в Views/Reports/Index.cshtml:

```
    <canvas id="expenseChart"></canvas>
    <script>
        var ctx =
document.getElementById('expenseChart').getContext('2d');
        var expenseChart = new Chart(ctx, {
            type: 'bar',
            data: {
                labels: @Json.Serialize(Model.Labels),
                datasets: [{
                    label: 'Расходы',
                    data: @Json.Serialize(Model.Expenses),
                    backgroundColor: 'rgba(255, 99, 132, 0.6)'
                }]
            }
        });
    </script>
```

Описана методологія побудови проєкту на ASP.NET Core MVC для веб-додатку для обліку особистих фінансів з аналізом фінансової ефективності демонструє виважену організацію коду, поділ завдань, зручність упровадження залежностей і використання ефективних інструментів для оброблення даних та візуалізації. Використання перевірених стандартів та застосування перевірених шаблонів проєктування гарантують масштабованість і зручність підтримки коду, а також спрощують інтеграцію нових функціональностей і підтримку застосунку в довгостроковій перспективі.

В ASP.NET Core впровадження залежностей використовується "з коробки". Під час старту програми в *Program.cs* налаштовують сервіси, такі як контекст бази даних, та інші компоненти.

У Program.cs можна побачити код вигляду:

```
    builder.Services.AddDbContext<FinanceOblikContext>
(options =>
    options.UseSqlServer(builder.Configuration.GetConnectionStri
ng("DefaultConnection")));
    builder.Services.AddControllersWithViews();
```

Це дає змогу використовувати додаток через конструктор контролера без необхідності явно створювати екземпляри контексту, що спрощує тестування і супровід коду.

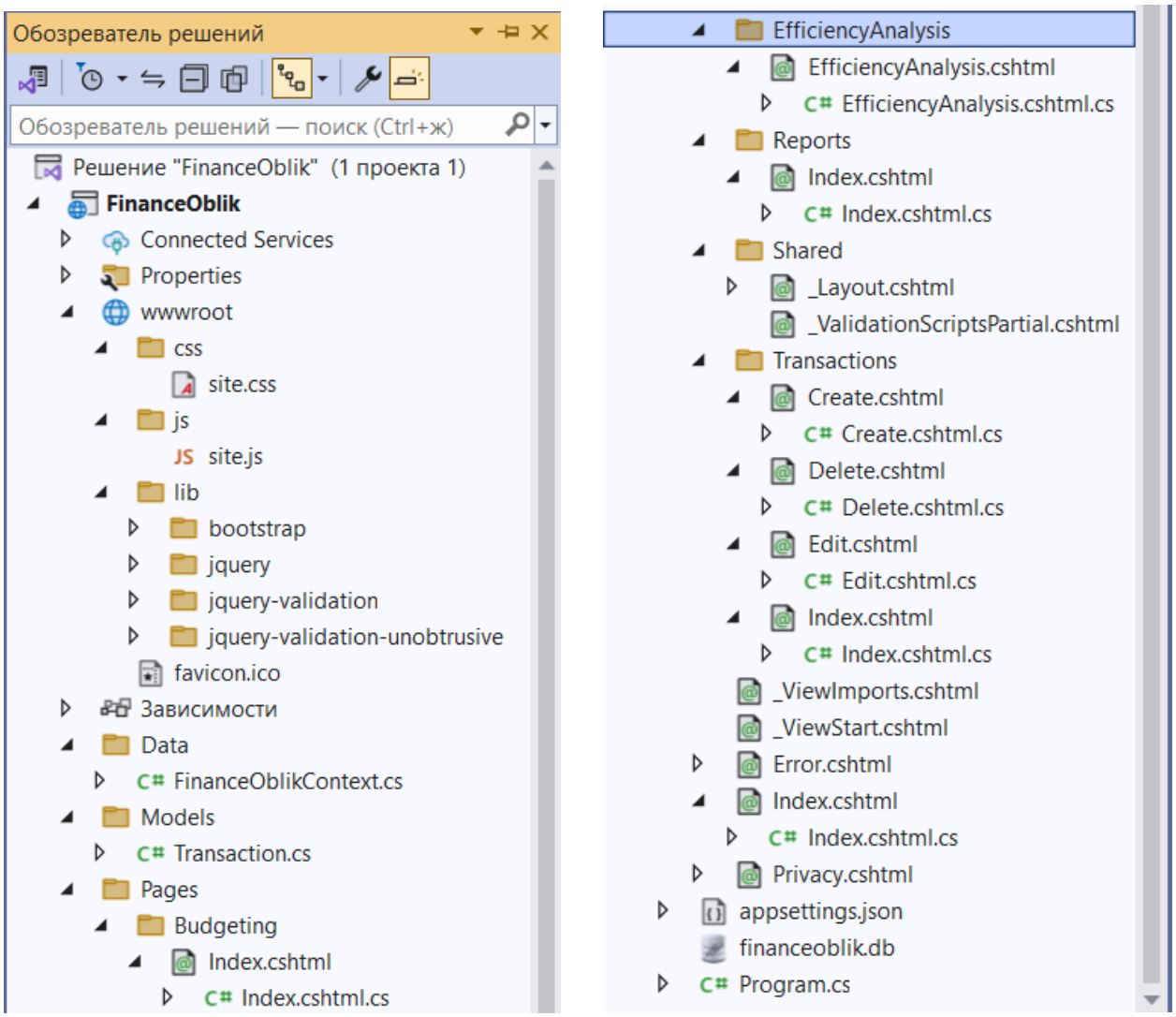


Рисунок 3.1 – Структура файлів розробленого веб-додатку

На рисунку 3.1 наведено перелік файлів розроблюваного рішення. В основі проекту знаходяться три ключові папки: *Controllers*, *Models* та *Views*. Папка *Controllers* містить класи контролерів, які відповідають за обробку HTTP-запитів, взаємодію з моделями та вибір відповідних представлень для відображення даних користувачу. Наприклад, *TransactionsController.cs* відповідає за операції з фінансовими транзакціями, такими як додавання, редагування та видалення записів.

Папка *Models* включає класи моделей, які визначають структуру даних та бізнес-логіку додатку. Тут розташовані класи, що представляють фінансові транзакції, категорії доходів і витрат, а також дані користувачів. Наприклад,

Transaction.cs описує властивості фінансової операції, такі як сума, категорія, дата та опис.

Views містить Razor-представлення, які відповідають за відображення даних у браузері. Кожен контролер має свою підпапку в *Views*, де розміщуються відповідні .cshtml файли.

Для роботи з базою даних проект використовує *Entity Framework Core*, тому у папці *Data* розташований клас *FinanceOblikContext.cs*, який визначає контекст даних та набір DbSet для кожної моделі. Ця папка також може містити міграції для оновлення структури бази даних.

wwwroot є кореневою папкою для статичних файлів, таких як CSS, JavaScript, зображення та бібліотеки. Тут зберігаються файли стилів, скрипти для інтерактивних елементів інтерфейсу та сторонні бібліотеки, наприклад, Chart.js для візуалізації фінансових даних.

Конфігураційні налаштування додатку зберігаються у файлі *appsettings.json*, який містить рядки підключення до бази даних, параметри аутентифікації та інші важливі налаштування. Цей файл можна змінювати залежно від середовища розробки, тестування чи продакшену.

Для управління залежностями та службами у проекті використовується *Dependency Injection*, налаштування яких відбувається у файлі *Program.cs*. Тут реєструються сервіси, такі як контекст бази даних, сервіси аутентифікації та інші необхідні компоненти.

Проект також включає папку *Services*, де можуть зберігатися додаткові сервіси або бізнес-логіка, які не належать безпосередньо до контролерів чи моделей. Наприклад, сервіси для обробки фінансових звітів або інтеграції з зовнішніми API.

Таким чином, структура файлового проекту веб-додатку організована з урахуванням найкращих практик розробки на ASP.NET Core MVC, що забезпечує легкість у підтримці, масштабуванні та подальшому розвитку проекту.

3.2 Інструкції з використання створеного веб-додатку

На основі вимог та рішень, сформульованих у попередньому розділі, було розроблено веб-додаток для обліку особистих фінансів. Розглянемо його інтерфейс та структуру.

Створений додаток має наступну структуру веб-сторінок:

а) Головна;

б) Транзакції;

в) Додати транзакцію;

г) Звіти;

д) Аналіз ефективності.

Розглянемо головну сторінку, що демонструється при завантаженні веб-додатку у браузері (рисунок 3.2). Вона носить здебільшого інформаційний характер та пропонує перейти по перегляду транзакцій.

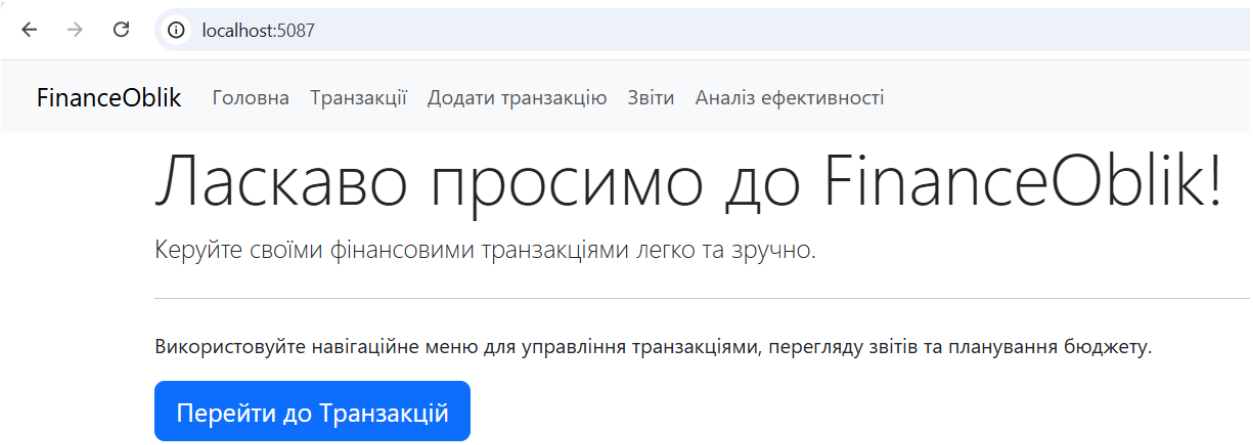


Рисунок 3.2 – Головна сторінка веб-застосунку

Перейти до переліку доходів та витрат можна шляхом натискання відповідної кнопки або вибору відповідного пункту головного меню.

На рисунку 3.3 наведено веб-сторінку «Транзакції». Як видно зі скріншоту, вона включає в себе блок, що включає таблицю з переліком операцій – доходів та витрат, та блок фільтрації, який дозволяє відображати результати згідно заданих критеріїв.

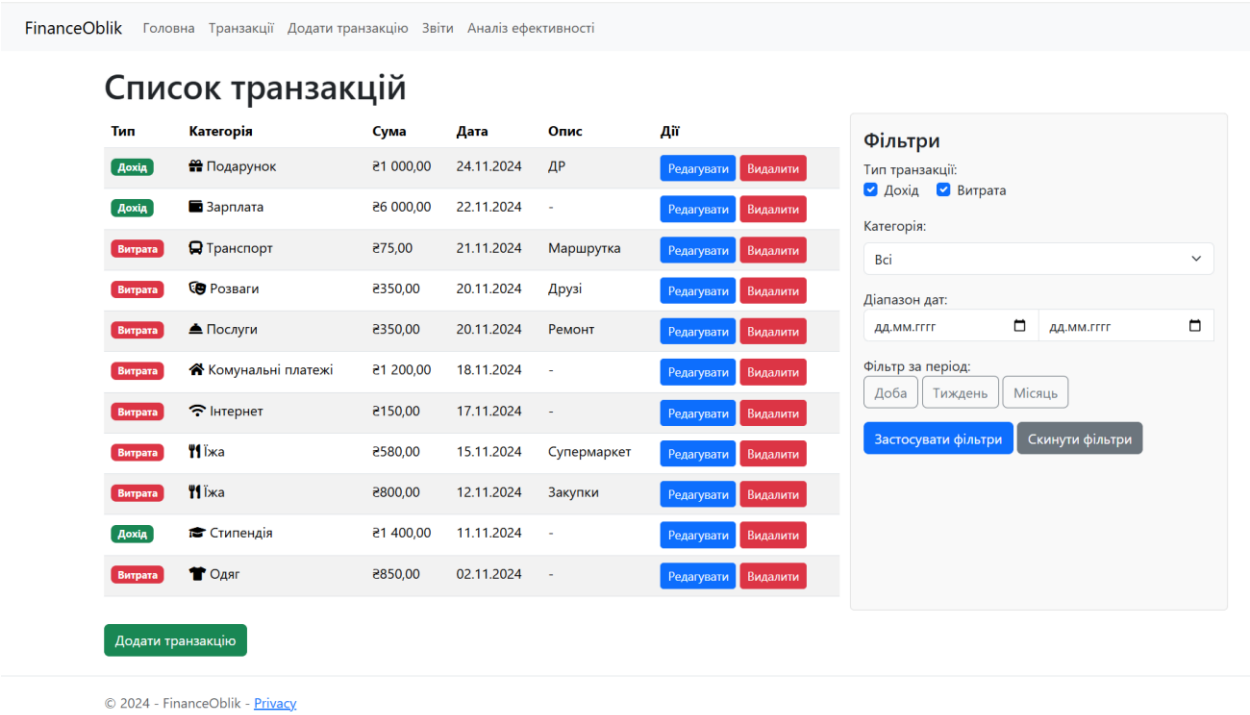


Рисунок 3.3 – Веб-сторінка «Транзакції»

В переліку операцій наводиться інформація про тип транзакції (дохід/витрата), що виділяється відповідним кольором, про категорію операції (тобто, на що саме було витрачено кошти, або з якого джерела надійшов дохід), сума, дата транзакції, перелік можливих дій по операції (редагування/видалення).

В блоці фільтрації, своєю чергою, користувач має змогу обирати тип відображуваних транзакцій шляхом використання чекбоксів, обирати необхідні категорії для відображення завдяки випадаючому меню, задавати необхідний діапазон дат по транзакціям, фільтрувати операції за останні добу, тиждень та місяць. Використання опції фільтрації дозволяє спростити роботу при великому обсязі зареєстрованих операцій.

В нижній частині сторінки розташовано кнопку «Додати транзакцію». Вона дублює пункт горизонтального меню та викликає наступне вікно (рисунок 3.4).

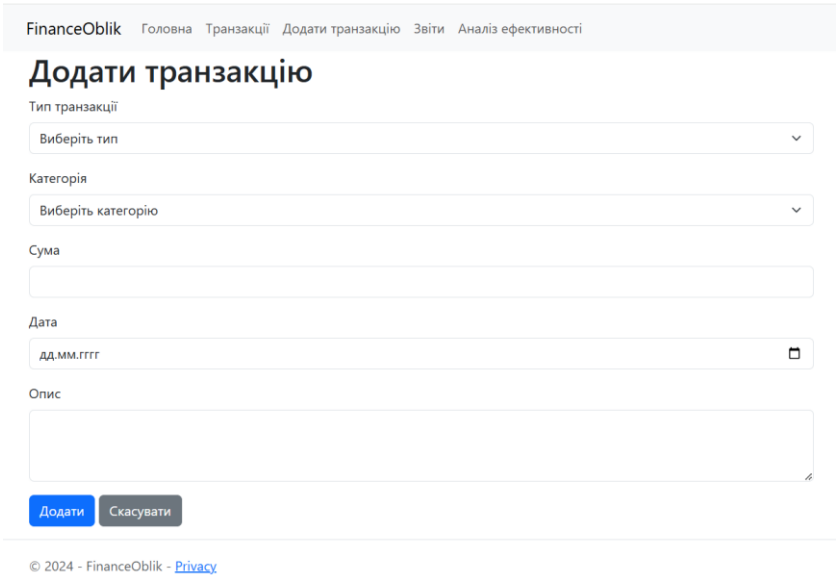


Рисунок 3.4 – Додавання нової операції

При спробі відредагувати транзакцію зі списку зазначене вікно набуде наступного вигляду (рисунок 3.5).

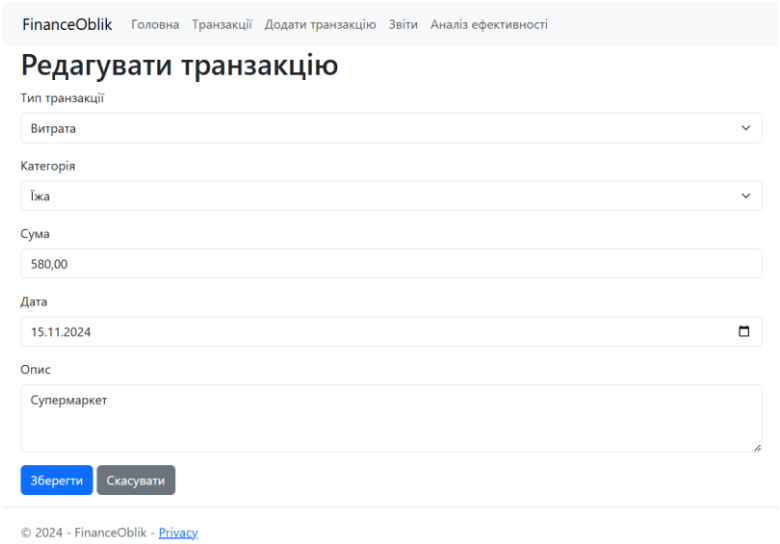


Рисунок 3.5 – Редагування наявної транзакції

Спроба видалення існуючої фінансової операції, своєю чергою, призведе до появи наступного вікна (рисунок 3.6).

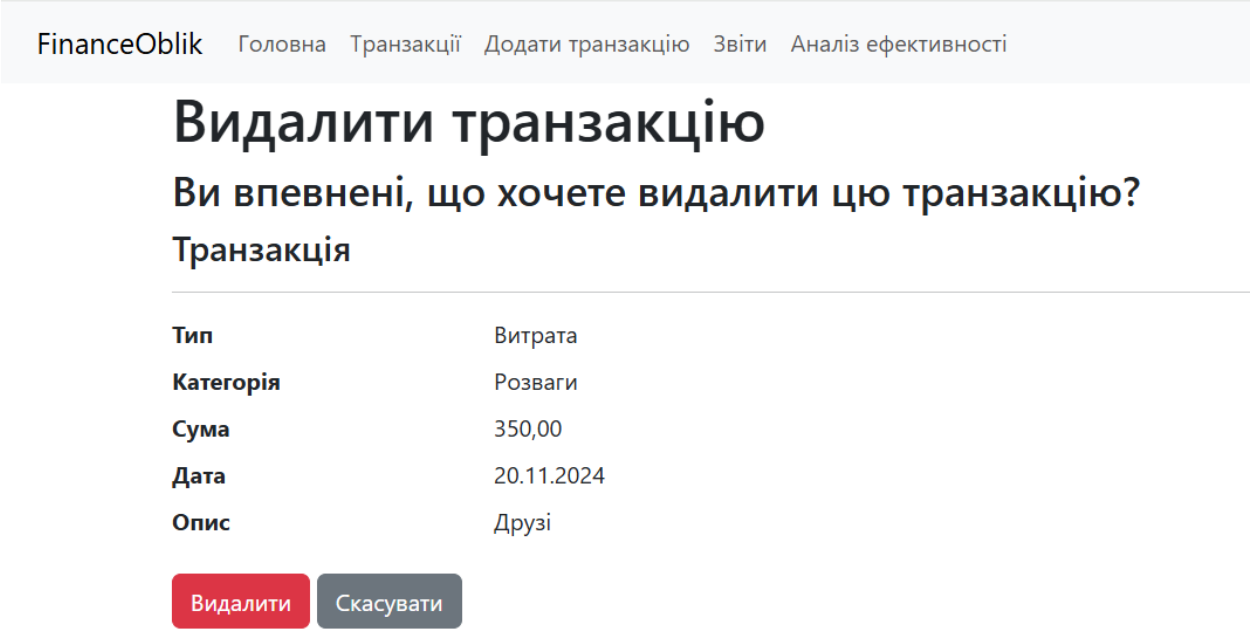


Рисунок 3.6 – Видалення транзакції

Розглянемо результат фільтрації за певним діапазоном та типом операцій (рисунок 3.7). Аналогічно реалізується фільтрація з використанням кнопок «Доба», «Тиждень», «Рік». Для вимкнення фільтру використовується кнопка «Скинути фільтри».

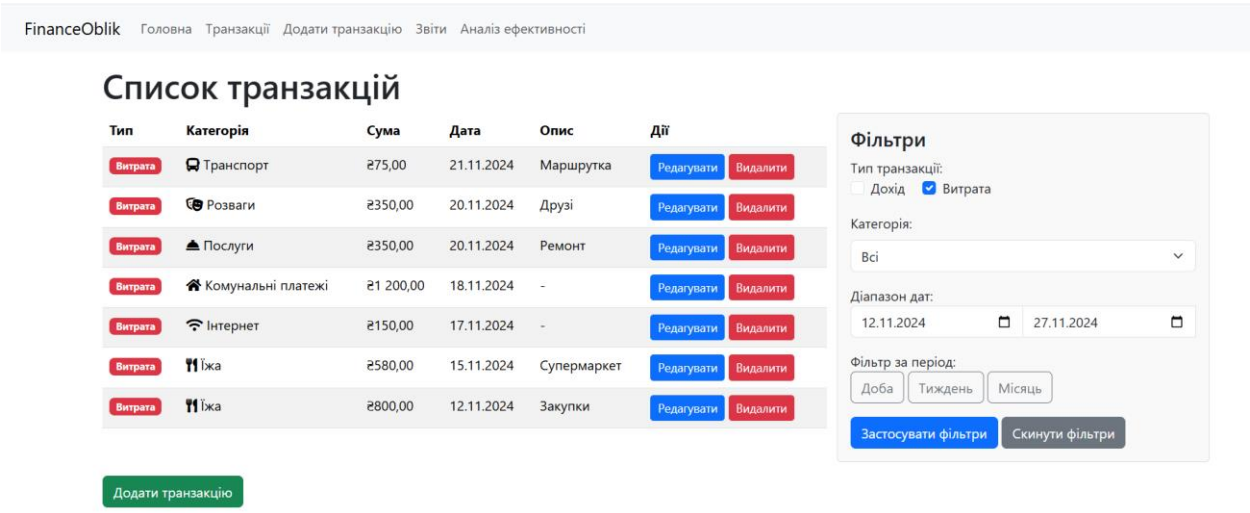


Рисунок 3.7 – Результат виконання фільрації операцій

Залежно від обраних у фільтрі типів транзакцій у випадаючому списку відображаються відповідні набори категорій (рисунок 3.7). Тип «Дохід»

включає в себе категорії, пов’язані з отриманням грошей – стипендія, зарплата, премія, виграш та інше. Тип «Витрата», відповідно, включає в себе великий перелік категорій для потенційного вкладення коштів. Для конкретизації окремої покупки в широкій категорії можна використовувати опис при створенні транзакції. Зазначений перелік наведено на рисунку 3.8.

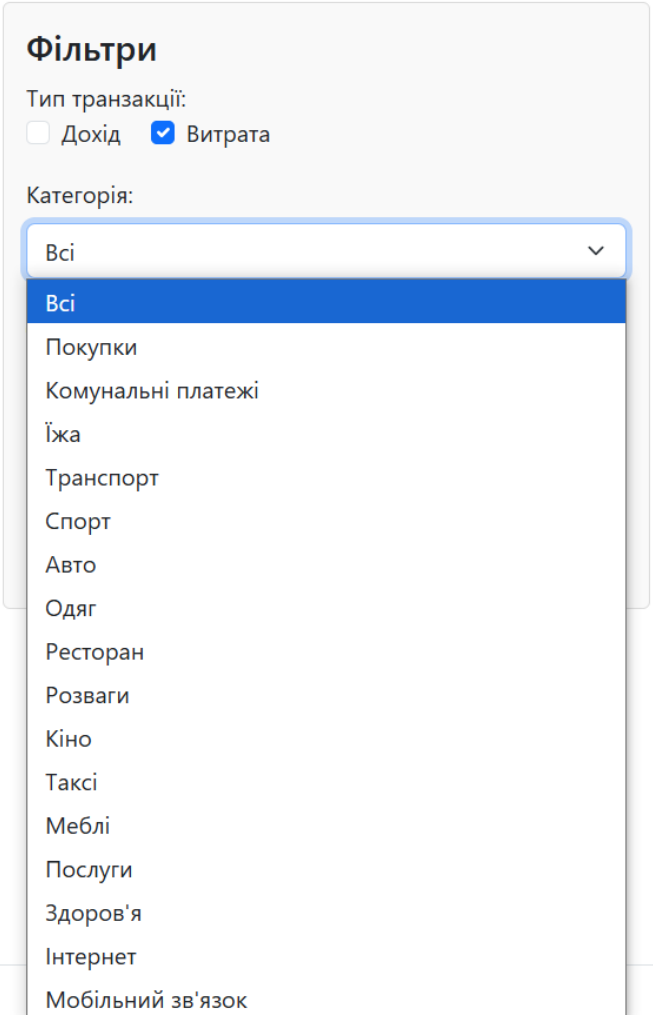


Рисунок 3.8 – Випадаючий список фільтру за категоріями

На веб-сторінці «Звіти» наводяться елементи візуалізації витрати коштів та доходу. Для цього використовуються графіки різної форми. Також можливе використання фільтрації для розділення за напрямком руху коштів та часовими інтервалами.

На рисунку 3.9 наведено початковий вигляд сторінки зі звітами, де замовчуванням обрано тип діаграми «Пиріг» та відображення доходів та витрат без фільтрації. На рисунку 3.10, своєю чергою, зображено застосування фільтра за вхідними фінансовими операціями.

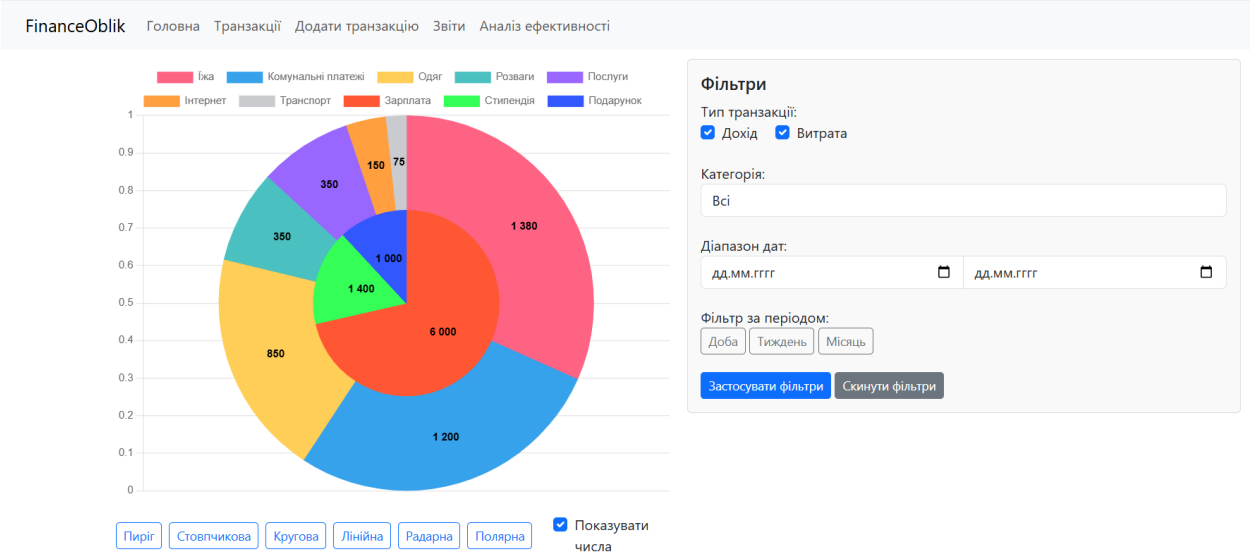


Рисунок 3.9 – Веб-сторінка «Звіти»

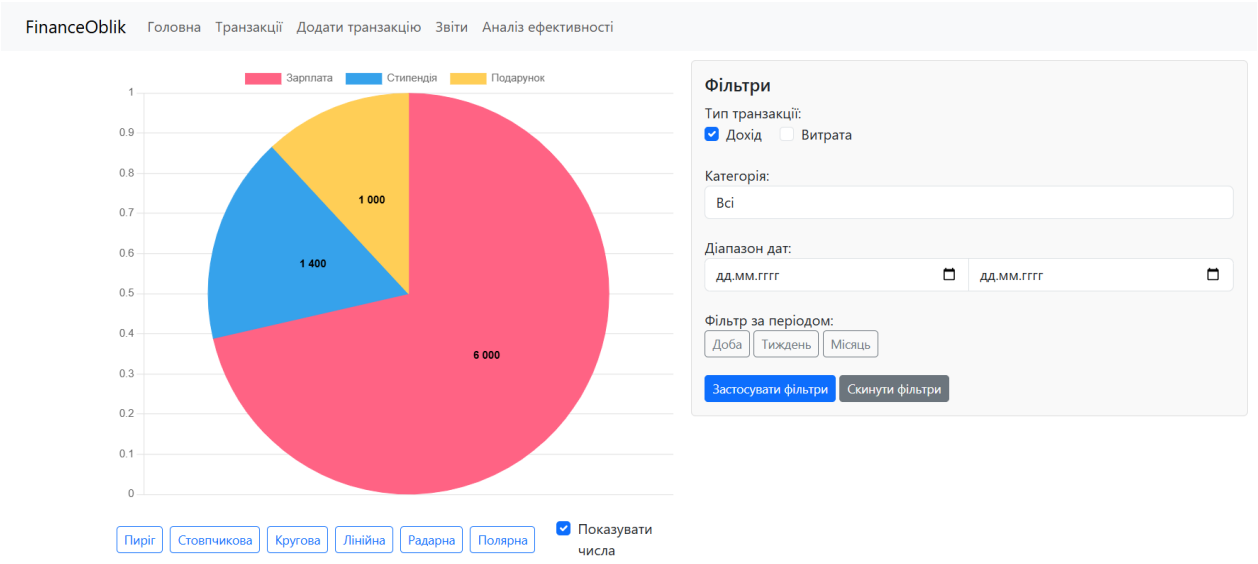


Рисунок 3.10 – Виконання фільтрації за доходами

Пирогова діаграма, за своєю суттю, є різновидом кругової діаграми з додаванням можливості подання даних у вигляді сегментів. Вона надає більш

деталізоване представлення розподілу витрат і доходів, даючи змогу користувачам бачити не тільки загальні частки, а й відносні зміни в рамках кожної категорії.

Розглянемо інші типи діаграм, що було використано в додатку. Серед них «Стовпчикова» та «Кругова» (рисунок 3.11).

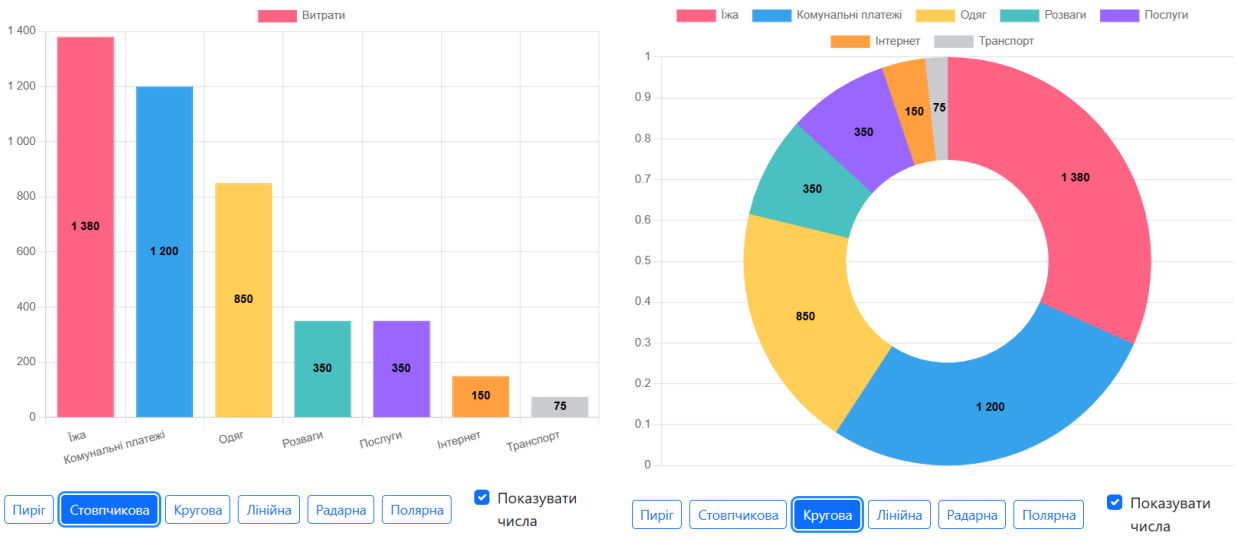


Рисунок 3.11 – Реалізація стовпчикової та кругової діаграм

Стовпчикова діаграма зручна тим, що стовпці чітко візуалізують різницю між величинами, дозволяючи користувачам миттєво оцінювати динаміку витрат. Кругова діаграма, своєю чергою, ефективно демонструє частки кожної категорії в загальному обсязі фінансових операцій. За своїм виглядом вона нагадує пирогову діаграму, але за рахунок порожньої середини дозволяє швидше оцінювати долі витрат користувача.

Лінійна діаграма за своєю сутністю в деякій мірі нагадує стовпчикову діаграму, але має більш лаконічний інтерфейс.

Для складнішого аналізу взаємозв'язків між різними фінансовими параметрами в додатку використана радарна діаграма. Вона представляє собою багатокутник, що розташований на сітці, схожій на павутиння.

Лінійна та радарна діаграми наведені на рисунку 3.12.

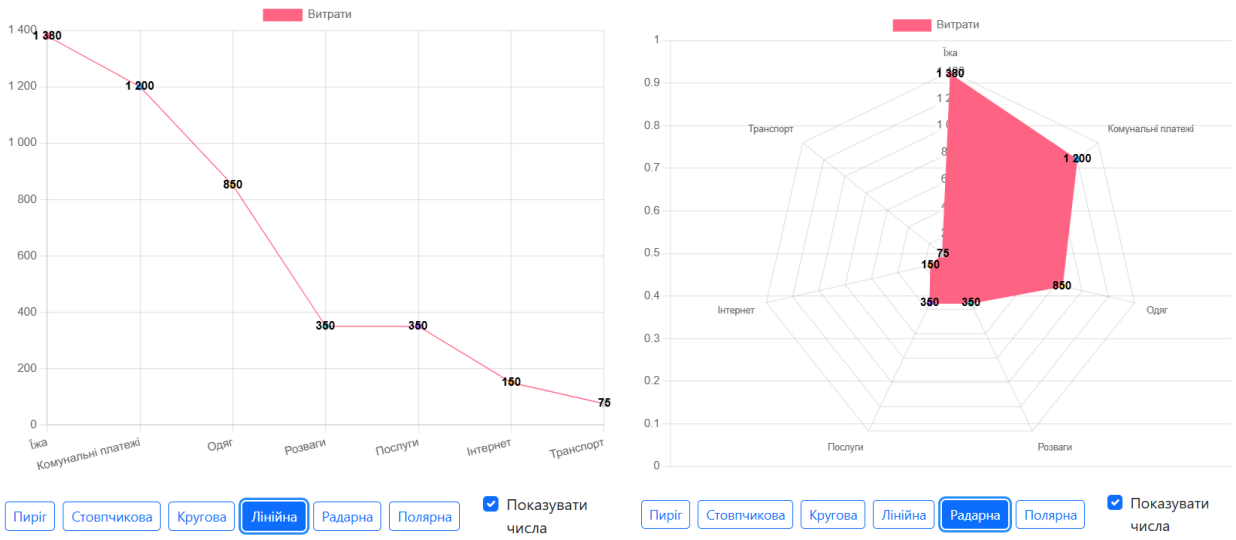


Рисунок 3.12 – Реалізація лінійної та радарної діаграм

Полярна діаграма використовується для представлення даних у просторі полярних координат. Полярна діаграма дає змогу користувачам бачити розподіл фінансових операцій за круговою шкалою, та в деякій мірі нагадує радарну діаграму (рисунок 3.13).

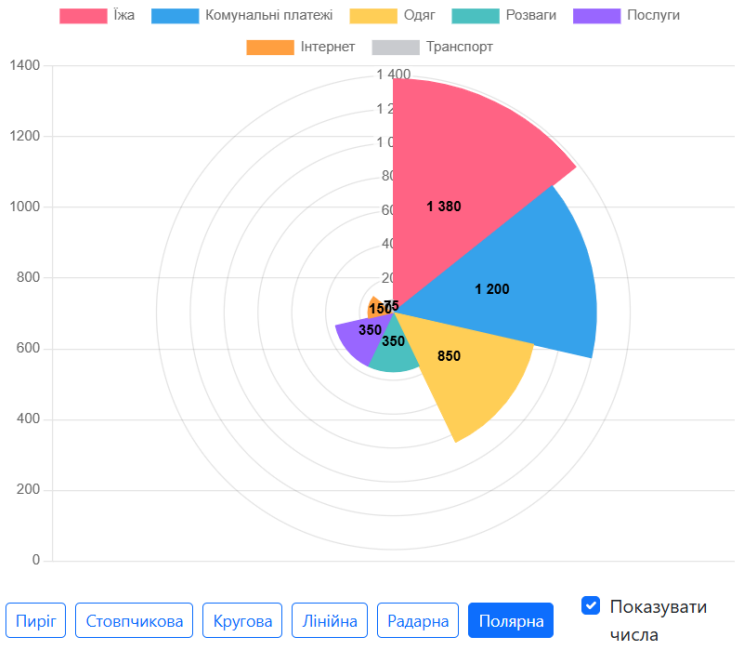


Рисунок 3.13 – Реалізація полярної діаграми

За для зручності запропонована опція відключення числових значень на діаграмах. Іноді це буває доречно через нагромадження чисел при задіянні

великої кількості категорій. Воно відбувається шляхом зміни стану чекбоксу «Показувати числа». При цьому при наведенні на сектор діаграми з'являється спливаюча підказка стосовно суми по обраній категорії. Це зображено на рисунку 3.14.

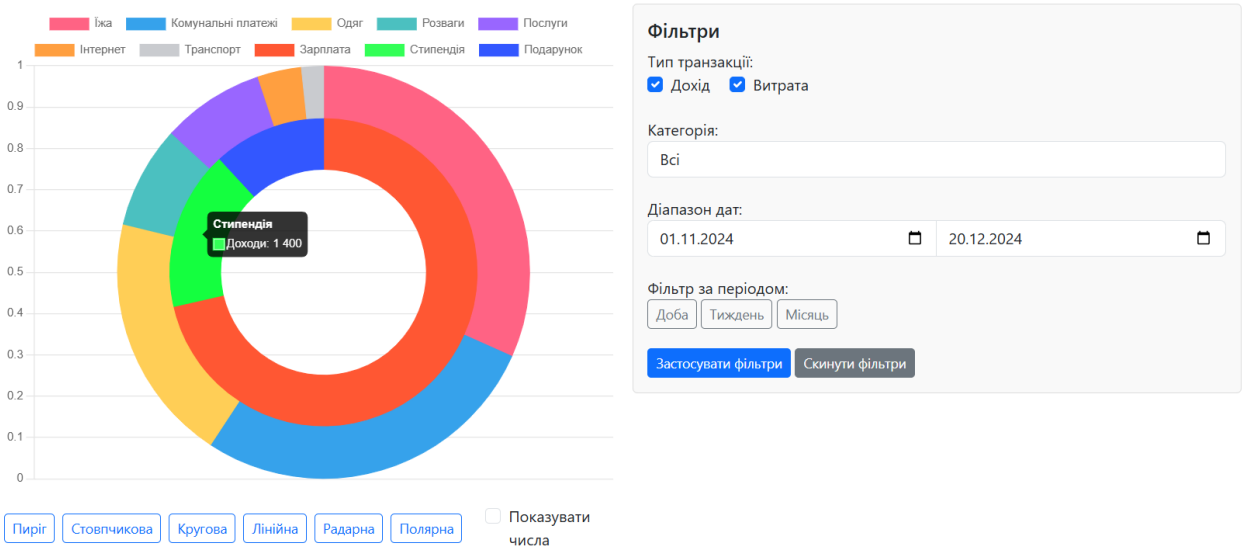


Рисунок 3.14 – Відключення демонстрації числових значень операцій

Останньою веб-сторінкою створеного додатку для обліку особистих фінансів є «Аналіз ефективності». На цій сторінці було вирішено розмістити стовпчикову діаграму аналізу доходів та витрат для надання можливості користувачеві оцінювати співвідношення зазначених операцій, ефективність його роботи та ефективність використання грошей. Для відмінності від веб-сторінки зі звітами блок фільтрації розташовано з лівої сторони, в той час як сам графік наведено з правої. Блок фільтрації майже не відрізняється від своєю реалізацією від блоків на сторінках «Звіти» та «Транзакції». Єдиною відмінністю є розташування чекбоксу демонстрації числових значень на графіках у тілі фільтра.

Веб-сторінка з аналізом фінансової ефективності наведена на рисунку 3.15.

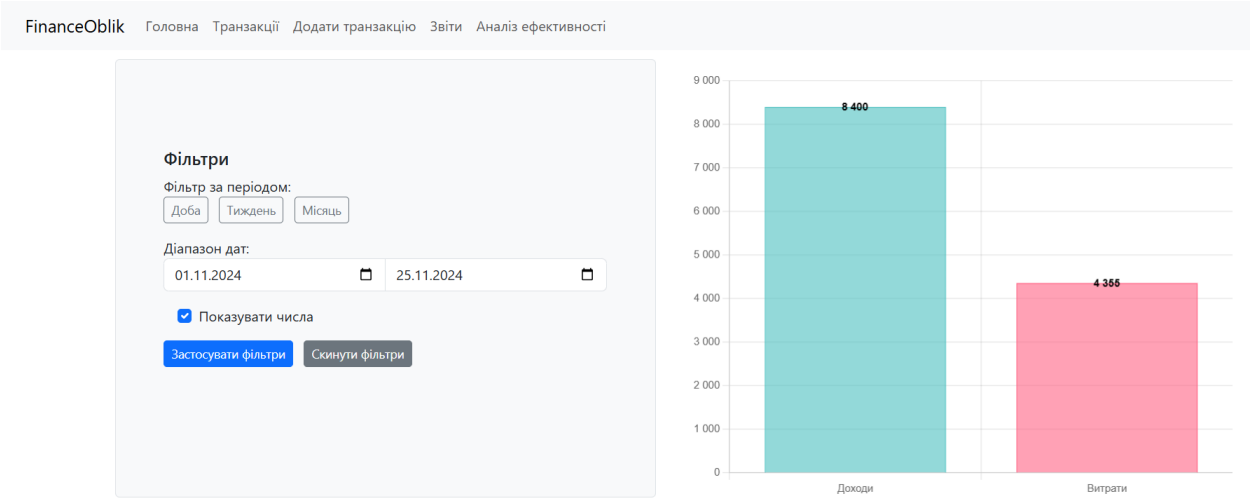


Рисунок 3.15 – Алгоритм роботи веб-застосунку

Для виходу в веб-застосунку достатньо закрити відповідну вкладку браузера.

Варто зазначити, що розгортання веб-додатку можливе з використанням різних підходів. Один із них передбачає використання IIS, вбудованого веб-сервера в Windows. Альтернативним варіантом є розгортання в хмарі, наприклад, з використанням Microsoft Azure або AWS. Також можна розглянути розгортання на платформах для веб-хостингу, які підтримують ASP.NET Core. В ході розробки та тестування використовувався вбудований у Visual Studio сервер IIS Express, який реалізує просте рішення для локального запуску додатків.

4 ДОСЛІДЖЕННЯ РОЗРОБЛЕНОГО ВЕБ-ДОДАТКУ ДЛЯ ОБЛІКУ ОСОБИСТИХ ФІНАНСІВ

Для комплексного оцінювання ефективності використання створеного веб-додатку для обліку особистих фінансів з аналізом фінансової ефективності в розділі 2.7 було розроблено математичну модель, що об'єднує такі показники, як фінансові поліпшення та економія часу при використанні додатку. На основі розрахунків згідно зазначеній математичній моделі було отримано графік на рисунку 4.1.

На представленому графіку динаміки використання веб-додатку для обліку особистих фінансів з аналізом фінансової ефективності відображено зміни цього показника протягом шести місяців — з червня по листопад. Графік ілюструє поступове зростання ефективності використання додатку, що свідчить про позитивні зміни у фінансовій діяльності користувачів.

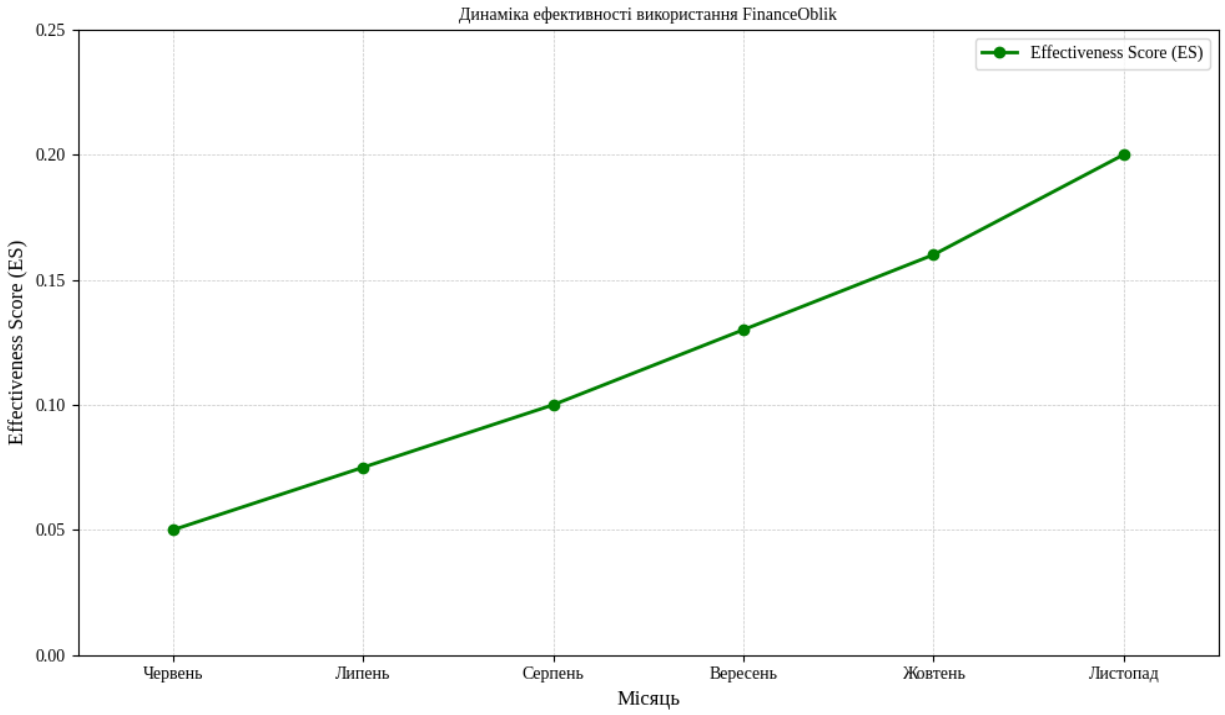


Рисунок 4.1 – Зміна показника комплексного оцінювання ефективності використання створеного веб-додатку для обліку особистих фінансів протягом шести місяців використання

Графік починається з місяця червня, де показник комплексного оцінювання ефективності використання становить 0.05. Відтоді цей показник поступово зростає, досягаючи 0.20 у листопаді. Така тенденція демонструє стабільне покращення ефективності управління фінансами користувачем додатку протягом періоду використання.

Отримана залежність є позитивною та лінійною, що означає, що з кожним місяцем ефективність використання додатку збільшується. Лінійна природа залежності свідчить про постійний темп покращення, без значних коливань чи зупинок у процесі оптимізації фінансових показників.

Продемонстровані результати свідчать про те, що розроблений веб-додаток ефективно сприяє покращенню фінансової дисципліни та управлінню фінансами користувачів. Поступове зростання показника ефективності демонструє, що додаток виконує свої функції на високому рівні, сприяючи збільшенню збережень, доходів та зменшенню витрат користувачів. Лінійна залежність підкреслює стабільність і надійність використання додатку в довгостроковій перспективі, що є важливим фактором для довіри та задоволеності користувачів.

Подібний аналіз допомагає зрозуміти, як використання веб-додатку для обліку особистих фінансів впливає на фінансову ефективність користувачів, і вказує на потенційні напрямки для подальшого вдосконалення додатку.

5 РОЗРАХУНОК ЕКОНОМІЧНОЇ ЕФЕКТИВНОСТІ РОЗРОБЛЕНОЇ СИСТЕМИ

5.1 Розрахунок собівартості веб-додатку для обліку особистих фінансів з аналізом фінансової ефективності

Визначимо собівартість створеного веб-додатку.

Для здійснення розрахунку застосовувалися наступні показники, що представлені у таблиці 5.1.

Таблиця 5.1 — Початкові показники для визначення собівартості

Найменування	Показник
Складність створення програми, днів	40
Ставка програміста (місяць), грн	17600
Кількість годин в місяці, год	160
Додаткова зарплатня (%)	10
Відрахування до соц. фондів (%)	15
Загальновиробничі витрати компанії (%)	100
ПДВ (%)	20

Виконаємо розрахунок на період у 1 місяць, що включатиме в себе 20 робочих днів.

а) Пункт 1. Комплектуючі вироби — CD-диски для інсталяції системи:

$$З_k = \sum Ц_k * n_k \tag{5.1}$$

де $З_К$ — витрати на CD-диски, грн.;

$Ц_К$ — вартість за 1 одиницю CD-диску, грн.;

n_k — кількість CD-дисків, шт.

Визначимо витрати на CD-диски:

$$З_k = 12*10 = 120грн \tag{5.2}$$

Витрати на CD-диски склали 120 грн.

б) Пункт 2. Витрати на електроенергію визначаємо згідно виразу 5.3:

$$B_E = P_E \sum W_i * t_i \tag{5.3}$$

де P_E — вартість 1 кВт-год, грн.;

Wi — середня споживана потужність в ході роботи.

Вартість 1кВт електроенергії на момент написання кваліфікаційної роботи складає 4,32 грн.

Розрахуємо витрати виробництва на електроенергію згідно виразу 5.4:

$$B_E = 4{,}32 * 0{,}65 * 160 = 449{,}28грн \tag{5.4}$$

Витрати виробництва на електроенергію складають 449,28 грн.

в) Пункт 3. Основна заробітна плата розраховується згідно виразу 5.5:

$$З_{осн} = l_{год} * T_{год} \tag{5.5}$$

де $l_{год}$ — тарифна ставка робітника (годинна), грн.;

$T_{год}$ — кількість робочих годин у місяці (160 год.).

Годинна тарифна ставка програміста, що розробляє систему, дорівнює 110 грн.

Розрахуємо основну заробітну плату програміста згідно виразу 5.6:

$$З_{осн} = 110*160 = 17600грн \quad (5.6)$$

Основна заробітна плата програміста складає 17600 грн.

г) Пункт 4. Додаткова заробітна плата розраховується згідно виразу 5.7:

$$З_{дод} = \frac{З_{осн} * Д\%}{100} \quad (5.7)$$

де $Здод$ — це додаткова заробітна плата розробника системи, грн.;

Д% — відсоток додаткової заробітної плати розробника системи (10%).

Визначимо додаткову заробітну плату згідно виразу 5.8:

$$З_{дод} = \frac{17600*10}{100} = 1760грн \quad (5.8)$$

Додаткова заробітна плата буде дорівнювати 1760 грн.

д) Пункт 5. Розмір відрахування в соц. фонди визначається згідно виразу 5.9:

$$З_{соц} = \frac{(З_{осн} + З_{дод}) * С\%}{100} \quad (5.9)$$

де *Зсоц* — розмір відрахування в соц. фонди, грн.;

С% — визначений відсоток відрахувань у соц. фонди (15%).

Обчислимо відрахування в соц. фонди згідно виразу 5.10:

$$З_{соц} = \frac{(17600+1760)*15}{100} = 2904грн \quad (5.10)$$

Розраховане відрахування в соц. фонди складає 2904 грн;

е) Пункт 6. Загальновиробничі витрати обчислюються згідно виразу 5.11:

$$З_{заг} = \frac{З_{осн} * Н_1\%}{100} \quad (5.11)$$

де $З_{заг}$ — загальновиробничі витрати підприємства, грн.;

$Н_1\%$ — запланований відсоток загальновиробничих витрат (100%).

Розрахуємо загальновиробничі витрати згідно виразу 5.12:

$$З_{заг} = \frac{17600*100}{100} = 17600грн \quad (5.12)$$

Загальновиробничі витрати підприємства складають 17600,00 грн.

Розрахуємо виробничу собівартість, виконавши послідовне додавання результатів здійснених розрахунків (вираз 5.13)

$$S_{пп} = 120 + 449{,}28 + 17600 + 1760 + 2904 + 17600 = 40433{,}28грн \quad (5.13)$$

де $S_{пп}$ — виробнича собівартість, грн.

Таким чином, виробнича собівартість розроблюваного веб-додатку складає 40433,28 грн.

В попередньо наведеній таблиці 5.2 представлено планову калькуляцію стосовно виробничої собівартості, вартості для підприємства та вартості для

замовника на створення системи багатофакторного оцінювання навчальних досягнень.

Створення системи багатофакторного оцінювання навчальних досягнень включало в себе роботи, що виконувались протягом 40 робочих днів.

Таблиця 5.2 — Планова калькуляція

Пункти калькуляції	**Сума, грн.**
CD-диски (Комплектуючи вироби)	120
Кошти на електроенергію:	449,28
Основна зар. плата	17600
Додаткова зар. плата	1760
Розмір відрахувань в соц. фонди	2904
Загальновиробничі витрати підприємства	17600
Виробнича собівартість в розрахунку на 1 місяць	40433,28
Собівартість розробки системи багатофакторного оцінювання навчальних досягнень.	80866,56

Таким чином, собівартість розробленого програмного забезпечення складає 80866,56 грн.

5.2 Визначення економічного ефекту від впровадження веб-додатку для обліку особистих фінансів з аналізом фінансової ефективності

Розрахувати економічний ефект від застосування веб-додатку для обліку особистих фінансів з аналізом фінансової ефективності є непростим завданням через складність визначення вартісної оцінки її використання. Тим не менш, можна сформулювати завдання визначення економічної ефективності

створеної системи наступним чином: вона дорівнює добутку вартісного ефекту реєстрації кожної транзакції на кількість зареєстрованих транзакцій протягом року. Математично це можна визначити шляхом запису наступного рівняння:

$$\text{Е}_{\text{еф}} = \text{Веф1} * \text{КСт} \tag{5.14}$$

Для розрахунку ефективності розробки та впровадження додатку потрібно визначити числові значення зазначених змінних. Емпірично визначенні числові показники даних параметрів наведені в таблиці 5.3. Кількість річних транзакцій визначена з розрахунку проведення 10 транзакцій на добу.

Таким чином, можна перейти до розрахунків терміну окупності розроблюваної системи за формулою:

$$T = \frac{C}{E_{еф}} \tag{5.15}$$

де Т – строк окупності розроблюваного проекту, міс.,

С – собівартість розроблюваного проекту, грн.,

$\text{Е}_{\text{еф}}$ – місячний економічний ефект від розроблюваного проекту, грн.

Таблиця 5.3 — Емпірично визначенні числові показники параметрів для розрахунку економ. ефективності впровадження веб-додатку для обліку особистих фінансів з аналізом фінансової ефективності

Параметр	**Значення**
Веф1	4
КСт	7320

Таким чином, економічний ефект складе

$$E_{еф} = 4 * 7320 = 29280 \text{ грн} \quad (5.16)$$

Визначимо термін окупності розроблюваного веб-додатку для обліку особистих фінансів з аналізом фінансової ефективності згідно виразу:

$$T = 80866{,}56/29280 = 2{,}76 \text{ року} \quad (5.17)$$

Отже, собівартість веб-додатку для обліку особистих фінансів з аналізом фінансової ефективності складає 80866,56 грн, економефект від його впровадження складає 29280 грн на рік, а витрачені засоби окупляться за 2,76 року. Таким чином, можна відмічати доцільність створення і впровадження зазначеного програмного забезпечення та економічну доцільність його застосування.

ВИСНОВКИ

В процесі розробки веб-додатку обліку особистих фінансів з аналізом фінансової ефективності були досягнуті результати, що відображають аналіз потреб користувачів і застосування сучасних веб-технологій. Створене програмне забезпечення являє собою інструмент для управління особистими фінансами, що поєднує в собі інтуїтивно зрозумілий інтерфейс та ергономічний функціонал.

Проведена робота включала детальне формулювання функціональних і нефункціональних вимог. Було підібрано засоби розробки, включно з C#, .NET Core та ASP.NET Core MVC, а також бібліотекою Chart.js, які в сукупності забезпечили ефективну реалізацію поставлених завдань.

Структура веб-додатка розроблена з акцентом на зручність та інтуїтивність. Користувачі отримують доступ до основних функцій через добре організовані сторінки. Реалізовані функції дають змогу легко додавати, редагувати та видаляти транзакції, а також генерувати деталізовані звіти та візуалізувати дані за допомогою інтерактивних графіків. Це забезпечує користувачам розуміння їхнього фінансового стану та сприяє ухваленню обґрунтованих рішень.

Алгоритм роботи програми побудований на основі інтеграції клієнтської та серверної частин, забезпечуючи плавну взаємодію та своєчасне оновлення даних.

Зазначене програмне забезпечення розроблено для широкої аудиторії, яка прагне ефективно управляти своїми фінансами та приймати обґрунтовані рішення на основі актуальних даних. Потенційними користувачами додатку є: приватні особи, які бажають провадити облік власних доходів і витрат, планувати бюджет і контролювати фінансові потоки; студенти та молоді фахівці, які починають самостійне фінансове життя; сім'ї та домашні господарства, які хочуть спільно контролювати сімейний бюджет, планувати

великі покупки або накопичення на майбутнє; малі підприємці та фрілансери, які потребують простого і зручного засобу для обліку витрат та доходів, пов'язаних із професійною діяльністю; люди, які прагнуть досягти конкретних фінансових цілей, таких як купівлю нерухомості або автівки.

Створене програмне забезпечення відповідає потребам користувачів, які цінують зручність, функціональність і надійність в інструментах фінансового обліку. Воно допомагає користувачам отримати контроль над своїми фінансами, покращити фінансову дисципліну та рухатися до поставлених цілей, що робить його актуальним і затребуваним рішенням на ринку фінансових додатків.

ПЕРЕЛІК ПОСИЛАНЬ

1. Lusardi A., Mitchell O. S. The Economic Importance of Financial Literacy: Theory and Evidence. Journal of Economic Literature. 2014. Vol. 52, no. 1. P. 5–44. URL: https://doi.org/10.1257/jel.52.1.5

2. Mishkin F. S. Financial markets & institutions. 4th ed. Boston, Mass : Addison-Wesley, 2003. 697 p.

3. Markowitz H. Portfolio Selection. The Journal of Finance. 1952. Vol. 7, no. 1. P. 77. URL: https://doi.org/10.2307/2975974

4. Sunstein C. R., Thaler R. H. Nudge: Improving Decisions About Health, Wealth, and Happiness. Yale University Press, 2008. 304 p.

5. Kiyosaki R. T. Rich Dad, Poor Dad (Rich Dad). Time Warner Paperbacks, 2002. 224 p.

6. Management for Digital Transformation / ed. by C. Machado, J. P. Davim. Cham : Springer International Publishing, 2024. URL: https://doi.org/10.1007/978-3-031-42060-3

7. Organisation for economic co-operation and development. OECD Economic Outlook 2015. Rowman & Littlefield Publishers, Incorporated, 2015.

8. Lim J. S., Heinrichs J. Digital Business Intelligence Management with Big Data Analytics. Lim, Jeen Su, 2021.

9. Quicken Classic [Електронний ресурс] – Режим доступу до ресурсу: https://www.quicken.com/

10. Quicken Bill Manager: How To Set Up Quick Pay and Check Pay [Електронний ресурс] – Режим доступу до ресурсу: https://www.quicken.com/support/quicken-bill-manager-how-set-quick-pay-and-check-pay/

11. Premium and Portfolio Manager [Електронний ресурс] – Режим доступу до ресурсу: https://www.morningstar.com/help-center/premium-and-portfolio-manager

12. Quicken Classic: Companion App [Електронний ресурс] – Режим доступу до ресурсу: https://play.google.com/store/apps/details?id=com.quicken.qm2014&hl=ru

13. The best personal finance software for power users [Електронний ресурс] – Режим доступу до ресурсу: https://www.pcmag.com/reviews/quicken-classic

14. Do Money Differently. Enjoy guilt-free spending and effortless saving with a friendly, flexible method for managing your finances [Електронний ресурс] – Режим доступу до ресурсу: https://www.ynab.com/

15. One integration, all of open banking [Електронний ресурс] – Режим доступу до ресурсу [Електронний ресурс] – Режим доступу до ресурсу: : https://plaid.com/en-eu/16

17. Powerful budgeting based on a sound philosophy [Електронний ресурс] – Режим доступу до ресурсу: https://www.pcmag.com/reviews/ynab

18. PocketGuard [Електронний ресурс] – Режим доступу до ресурсу: https://pocketguard.com/

19. Good personal finance tools with a terrific interface [Електронний ресурс] – Режим доступу до ресурсу: https://www.pcmag.com/reviews/pocketguard

20. Troelsen A., Japikse P. Pro C# 10 With . NET 6: Foundational Principles and Practices in Programming. Apress L. P., 2022.

21. .NET Core: можливості та перспективи [Електронний ресурс] – Режим доступу до ресурсу: https://dou.ua/lenta/articles/net-core/

22. Overview of ASP.NET Core MVC [Електронний ресурс] – Режим доступу до ресурсу: https://learn.microsoft.com/uk-ua/aspnet/core/mvc/overview?view=aspnetcore-9.0

23. The DCI Architecture: A New Vision of Object-Oriented Programming [Електронний ресурс] – Режим доступу до ресурсу: https://www.artima.com/articles/the-dci-architecture-a-new-vision-of-object-oriented-programming

24. Chart.js. Simple yet flexible JavaScript charting library for the modern web [Електронний ресурс] – Режим доступу до ресурсу: https://www.chartjs.org/

25. Entity Framework Core [Електронний ресурс] – Режим доступу до ресурсу: https://www.entityframeworktutorial.net/efcore/entity-framework-core.aspx

Додаток А - Програмний код

Transactions/Index.cshtml

```
@page
@model FinanceOblik.Pages.Transactions.IndexModel
@{
    ViewData["Title"] = "Список транзакцій";
}
<p></p>
<h1>Список транзакцій</h1>

<div class="row">
    <div class="col-md-8">
        <table class="table table-striped">
            <thead>
                <tr>
                    <th>Тип</th>
                    <th>Категорія</th>
                    <th>Сума</th>
                    <th>Дата</th>
                    <th>Опис</th>
                    <th>Дії</th>
                </tr>
            </thead>
            <tbody>
                @foreach (var item in Model.Transactions)
                {
                    <tr>
                        <td>
                            @if (item.Type == "Дохід")
                            {
                                <span       class="badge       bg-
success">Дохід</span>
                            }
                            else
                            {
                                <span       class="badge       bg-
danger">Витрата</span>
                            }
                        </td>
                        <td>
                            <i
class="@Model.GetCategoryIcon(item.Category)"></i> @item.Category
                        </td>
                        <td>₴@item.Amount.ToString("N2")</td>

<td>@item.Date.ToString("dd.MM.yyyy")</td>
                        <td>@item.Description</td>
                        <td>
                            <a    asp-page="./Edit"    asp-route-
id="@item.Id" class="btn btn-sm btn-primary">Редагувати</a>
```

```
                            <a   asp-page="./Delete"   asp-route-
id="@item.Id" class="btn btn-sm btn-danger">Видалити</a>
                        </td>
                    </tr>
                }
            </tbody>
        </table>
    </div>
    <div class="filter-container col-md-4">
        <h4>Фільтри</h4>
        <form method="get">
            <div class="mb-3">
                <label>Тип транзакції:</label><br />
                @foreach (var type in Model.TypeSelectList)
                {
                    <div class="form-check form-check-inline">
                        <input           class="form-check-input"
type="checkbox"      name="FilterTypes"      value="@type.Value"
@(Model.FilterTypes.Contains(type.Value) ? "checked" : "")>
                        <label                class="form-check-
label">@type.Text</label>
                    </div>
                }
            </div>
            <div class="mb-3">
                <label   asp-for="FilterCategory"   class="form-
label">Категорія:</label>
                <select      asp-for="FilterCategory"      asp-
items="Model.CategorySelectList" class="form-select">
                    <option value="">Всі</option>
                </select>
            </div>
            <div class="mb-3">
                <label>Діапазон дат:</label>
                <div class="input-group">
                    <input   asp-for="StartDate"   class="form-
control" type="date" />
                    <input asp-for="EndDate" class="form-control"
type="date" />
                </div>
            </div>
            <div class="mb-3">
                <label>Фільтр за період:</label><br />
                <button  type="submit"  name="period"  value="day"
class="btn btn-outline-secondary">Доба</button>
                <button type="submit" name="period" value="week"
class="btn btn-outline-secondary">Тиждень</button>
                <button type="submit" name="period" value="month"
class="btn btn-outline-secondary">Місяць</button>
            </div>
            <button      type="submit"      class="btn      btn-
primary">Застосувати фільтри</button>
```

```
            <a      asp-page="./Index"      class="btn      btn-
secondary">Скинути фільтри</a>
        </form>
    </div>
</div>
<a   asp-page="./Create"   class="btn   btn-success   mt-3">Додати
транзакцію</a>
```

Transactions/Index.cshtml.cs

```
using Microsoft.AspNetCore.Mvc.RazorPages;
using Microsoft.EntityFrameworkCore;
//using FinanceOblik.Data;
using FinanceOblik.Models;
using Microsoft.AspNetCore.Mvc.Rendering;
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;
using System;
using Microsoft.AspNetCore.Mvc;

namespace FinanceOblik.Pages.Transactions
{
    public class IndexModel : PageModel
    {
        private readonly FinanceOblikContext _context;

        public IndexModel(FinanceOblikContext context)
        {
            _context = context;
        }

        // Список транзакцій
        public IList<Transaction> Transactions { get; set; }

        // SelectList для типів та категорій
        public IEnumerable<SelectListItem> TypeSelectList { get;
set; }
        public SelectList CategorySelectList { get; set; }

        // Фільтри
        [BindProperty(SupportsGet = true)]
        public  List<string>  FilterTypes  {  get;  set;  }  =  new
List<string>();

        [BindProperty(SupportsGet = true)]
        public string FilterCategory { get; set; }

        [BindProperty(SupportsGet = true)]
        public DateTime? StartDate { get; set; }

        [BindProperty(SupportsGet = true)]
```

```
        public DateTime? EndDate { get; set; }

        public async Task OnGetAsync(string period)
        {
            TypeSelectList = new List<SelectListItem>
            {
                new SelectListItem { Value = "Дохід", Text =
"Дохід" },
                new SelectListItem { Value = "Витрата", Text =
"Витрата" }
            };

            var allCategories = new List<string>
            {
                "Стипендія", "Зарплата", "Премія", "Подарунок",
"Дивіденди", "Проценти", "Виграш", "Знахідка", "Інше",
                "Покупки", "Комунальні платежі", "Їжа",
"Транспорт", "Спорт", "Авто", "Одяг", "Ресторан", "Розваги",
                "Кіно", "Таксі", "Меблі", "Послуги", "Здоров'я",
"Інтернет", "Мобільний зв'язок"
            };

            CategorySelectList = new SelectList(allCategories);

            var transactions = from t in _context.Transactions
                               select t;

            if (FilterTypes != null && FilterTypes.Any())
            {
                transactions = transactions.Where(t =>
FilterTypes.Contains(t.Type));
            }

            if (!string.IsNullOrEmpty(FilterCategory))
            {
                transactions = transactions.Where(t => t.Category
== FilterCategory);
            }

            if (StartDate.HasValue && EndDate.HasValue)
            {
                transactions = transactions.Where(t => t.Date >=
StartDate.Value && t.Date <= EndDate.Value);
            }

            if (!string.IsNullOrEmpty(period))
            {
                DateTime now = DateTime.Now;
                switch (period)
                {
                    case "day":
```

```
                        transactions = transactions.Where(t =>
t.Date.Date == now.Date);
                        break;
                    case "week":
                        var startOfWeek = now.AddDays(-7);
                        transactions = transactions.Where(t =>
t.Date >= startOfWeek);
                        break;
                    case "month":
                        var startOfMonth = new DateTime(now.Year,
now.Month, 1);
                        transactions = transactions.Where(t =>
t.Date >= startOfMonth);
                        break;
                }
            }

            Transactions = await transactions.OrderByDescending(t
=> t.Date).ToListAsync();

            // Фільтрація категорій на основі типів
            if (FilterTypes.Contains("Дохід") &&
!FilterTypes.Contains("Витрата"))
            {
                var incomeCategories = new List<string>
                {
                    "Стипендія", "Зарплата", "Премія",
"Подарунок", "Дивіденди", "Проценти", "Виграш", "Знахідка", "Інше"
                };
                CategorySelectList = new
SelectList(incomeCategories);
            }
            else if (FilterTypes.Contains("Витрата") &&
!FilterTypes.Contains("Дохід"))
            {
                var expenseCategories = new List<string>
                {
                    "Покупки", "Комунальні платежі", "Їжа",
"Транспорт", "Спорт", "Авто", "Одяг", "Ресторан",
                    "Розваги", "Кіно", "Таксі", "Меблі",
"Послуги", "Здоров'я", "Інтернет", "Мобільний зв'язок"
                };
                CategorySelectList = new
SelectList(expenseCategories);
            }
            else
            {
                CategorySelectList = new
SelectList(allCategories);
            }
        }
```

```
        public string GetCategoryIcon(string category)
        {
            var icons = new Dictionary<string, string>
            {
                { "Стипендія", "fas fa-graduation-cap" },
                { "Зарплата", "fas fa-wallet" },
                { "Премія", "fas fa-award" },
                { "Подарунок", "fas fa-gift" },
                { "Дивіденди", "fas fa-chart-line" },
                { "Проценти", "fas fa-percent" },
                { "Виграш", "fas fa-trophy" },
                { "Знахідка", "fas fa-search" },
                { "Інше", "fas fa-ellipsis-h" },
                { "Покупки", "fas fa-shopping-cart" },
                { "Комунальні платежі", "fas fa-home" },
                { "Їжа", "fas fa-utensils" },
                { "Транспорт", "fas fa-bus" },
                { "Спорт", "fas fa-basketball-ball" },
                { "Авто", "fas fa-car" },
                { "Одяг", "fas fa-tshirt" },
                { "Ресторан", "fas fa-concierge-bell" },
                { "Розваги", "fas fa-theater-masks" },
                { "Кіно", "fas fa-film" },
                { "Таксі", "fas fa-taxi" },
                { "Меблі", "fas fa-couch" },
                { "Послуги", "fas fa-concierge-bell" },
                { "Здоров'я", "fas fa-heartbeat" },
                { "Інтернет", "fas fa-wifi" },
                { "Мобільний зв'язок", "fas fa-mobile-alt" }
            };

            return icons.ContainsKey(category) ? icons[category]
: "fas fa-question-circle";
        }
    }
}
```

Transactions/Create.cshtml.cs

```
using Microsoft.AspNetCore.Mvc;
using Microsoft.AspNetCore.Mvc.RazorPages;
using FinanceOblik.Models;
using Microsoft.AspNetCore.Mvc.Rendering;
using System.Collections.Generic;
using System.Threading.Tasks;


namespace FinanceOblik.Pages.Transactions
{
    public class CreateModel : PageModel
    {
        private readonly FinanceOblikContext _context;
```

```
        public CreateModel(FinanceOblikContext context)
        {
            _context = context;
        }

        [BindProperty]
        public Transaction Transaction { get; set; }

        public IEnumerable<SelectListItem> TypeSelectList { get;
set; }
        public SelectList CategorySelectList { get; set; }

        public void OnGet()
        {
            TypeSelectList = new List<SelectListItem>
            {
                new SelectListItem { Value = "Дохід", Text =
"Дохід" },
                new SelectListItem { Value = "Витрата", Text =
"Витрата" }
            };

            var allCategories = new List<string>
            {
                // Категорії для "Дохід"
                "Стипендія", "Зарплата", "Премія", "Подарунок",
"Дивіденди", "Проценти", "Виграш", "Знахідка", "Інше",
                // Категорії для "Витрата"
                "Покупки", "Комунальні платежі", "Їжа",
"Транспорт", "Спорт", "Авто", "Одяг", "Ресторан", "Розваги",
                "Кіно", "Таксі", "Меблі", "Послуги", "Здоров'я",
"Інтернет", "Мобільний зв'язок"
            };

            CategorySelectList = new SelectList(allCategories);
        }

        public async Task<IActionResult> OnPostAsync()
        {
            if (!ModelState.IsValid)
            {
                TypeSelectList = new List<SelectListItem>
                {
                    new SelectListItem { Value = "Дохід", Text =
"Дохід" },
                    new SelectListItem { Value = "Витрата", Text
= "Витрата" }
                };

                var allCategories = new List<string>
                {
```

```
                    "Стипендія", "Зарплата", "Премія",
"Подарунок", "Дивіденди", "Проценти", "Виграш", "Знахідка",
"Інше",
                    "Покупки", "Комунальні платежі", "Їжа",
"Транспорт", "Спорт", "Авто", "Одяг", "Ресторан", "Розваги",
                    "Кіно", "Таксі", "Меблі", "Послуги",
"Здоров'я", "Інтернет", "Мобільний зв'язок"
                };

                CategorySelectList = new
SelectList(allCategories);
                return Page();
            }

            _context.Transactions.Add(Transaction);
            await _context.SaveChangesAsync();

            return RedirectToPage("./Index");
        }
    }
}
```

Transactions/Edit.cshtml.cs

```
using Microsoft.AspNetCore.Mvc;
using Microsoft.AspNetCore.Mvc.RazorPages;
using FinanceOblik.Models;
using Microsoft.AspNetCore.Mvc.Rendering;
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;
using Microsoft.EntityFrameworkCore;


namespace FinanceOblik.Pages.Transactions
{
    public class EditModel : PageModel
    {
        private readonly FinanceOblikContext _context;

        public EditModel(FinanceOblikContext context)
        {
            _context = context;
        }

        [BindProperty]
        public Transaction Transaction { get; set; }

        public IEnumerable<SelectListItem> TypeSelectList { get;
set; }
        public SelectList CategorySelectList { get; set; }

        public async Task<IActionResult> OnGetAsync(int id)
```

```
        {
            Transaction = await
_context.Transactions.FindAsync(id);

            if (Transaction == null)
            {
                return NotFound();
            }

            TypeSelectList = new List<SelectListItem>
            {
                new SelectListItem { Value = "Дохід", Text =
"Дохід", Selected = Transaction.Type == "Дохід" },
                new SelectListItem { Value = "Витрата", Text =
"Витрата", Selected = Transaction.Type == "Витрата" }
            };

            var allCategories = new List<string>
            {
                "Стипендія", "Зарплата", "Премія", "Подарунок",
"Дивіденди", "Проценти", "Виграш", "Знахідка", "Інше",
                "Покупки", "Комунальні платежі", "Їжа",
"Транспорт", "Спорт", "Авто", "Одяг", "Ресторан", "Розваги",
                "Кіно", "Таксі", "Меблі", "Послуги", "Здоров'я",
"Інтернет", "Мобільний зв'язок"
            };

            CategorySelectList = new SelectList(allCategories,
Transaction.Category);

            return Page();
        }

        public async Task<IActionResult> OnPostAsync()
        {
            if (!ModelState.IsValid)
            {
                // Повторна ініціалізація списків у випадку
невдалої валідації
                TypeSelectList = new List<SelectListItem>
                {
                    new SelectListItem { Value = "Дохід", Text =
"Дохід", Selected = Transaction.Type == "Дохід" },
                    new SelectListItem { Value = "Витрата", Text
= "Витрата", Selected = Transaction.Type == "Витрата" }
                };

                var allCategories = new List<string>
                {
                    "Стипендія", "Зарплата", "Премія",
"Подарунок", "Дивіденди", "Проценти", "Виграш", "Знахідка",
"Інше",
```

```
                    "Покупки", "Комунальні платежі", "Їжа",
"Транспорт", "Спорт", "Авто", "Одяг", "Ресторан", "Розваги",
                    "Кіно", "Таксі", "Меблі", "Послуги",
"Здоров'я", "Інтернет", "Мобільний зв'язок"
                };

                CategorySelectList = new
SelectList(allCategories, Transaction.Category);
                return Page();
            }

            _context.Attach(Transaction).State =
EntityState.Modified;

            try
            {
                await _context.SaveChangesAsync();
            }
            catch (DbUpdateConcurrencyException)
            {
                if (!TransactionExists(Transaction.Id))
                {
                    return NotFound();
                }
                else
                {
                    throw;
                }
            }

            return RedirectToPage("./Index");
        }

        private bool TransactionExists(int id)
        {
            return _context.Transactions.Any(e => e.Id == id);
        }
    }
}
```

Transactions/Delete.cshtml.cs

```
using Microsoft.AspNetCore.Mvc;
using Microsoft.AspNetCore.Mvc.RazorPages;
using FinanceOblik.Models;
using System.Threading.Tasks;

namespace FinanceOblik.Pages.Transactions
{
    public class DeleteModel : PageModel
    {
        private readonly FinanceOblikContext _context;
```

```
        public DeleteModel(FinanceOblikContext context)
        {
            _context = context;
        }

        [BindProperty]
        public Transaction Transaction { get; set; }

        public async Task<IActionResult> OnGetAsync(int id)
        {
            Transaction = await
_context.Transactions.FindAsync(id);

            if (Transaction == null)
            {
                return NotFound();
            }

            return Page();
        }

        public async Task<IActionResult> OnPostAsync()
        {
            if (Transaction == null)
            {
                return NotFound();
            }

            var transactionToDelete = await
_context.Transactions.FindAsync(Transaction.Id);

            if (transactionToDelete != null)
            {

_context.Transactions.Remove(transactionToDelete);
                await _context.SaveChangesAsync();
            }

            return RedirectToPage("./Index");
        }
    }
}
```

Reports/Index.cshtml

```
@page
@model FinanceOblik.Pages.Reports.IndexModel
@using System.Text.Json
@{
    ViewData["Title"] = "Звіти";
}
```

```
<style>
    .chart-container {
        position: relative;
        padding-top: 0px;
    }
</style>

<div class="reports-container">
    <div class="reports-left">
        <div class="chart-container" style="height: 500px;">
            <canvas id="expenseChart"></canvas>
        </div>
        <div class="mt-3 d-flex align-items-center">
            <button type="button" class="btn btn-outline-primary
btn-sm change-chart-type me-2" data-type="pie">Пиріг</button>
            <button type="button" class="btn btn-outline-primary
btn-sm          change-chart-type          me-2"          data-
type="bar">Стовпчикова</button>
            <button type="button" class="btn btn-outline-primary
btn-sm          change-chart-type          me-2"          data-
type="doughnut">Кругова</button>
            <button type="button" class="btn btn-outline-primary
btn-sm change-chart-type me-2" data-type="line">Лінійна</button>
            <button type="button" class="btn btn-outline-primary
btn-sm change-chart-type me-2" data-type="radar">Радарна</button>
            <button type="button" class="btn btn-outline-primary
btn-sm          change-chart-type          me-2"          data-
type="polarArea">Полярна</button>

            <div class="form-check form-check-inline ms-3">
                <input  class="form-check-input"  type="checkbox"
id="toggleDataLabels" checked>
                <label                   class="form-check-label"
for="toggleDataLabels">Показувати числа</label>
            </div>
        </div>
    </div>
    <div class="reports-right">
        <div class="filter-container">
            <h5>Фільтри</h5>
            <form method="get" id="filterForm">
                <div class="filter-section">
                    <label>Тип транзакції:</label><br />
                    <div class="form-check form-check-inline">
                        <input          class="form-check-input"
type="checkbox"    name="FilterTypes"    checked    value="Дохід"
@(Model.FilterTypes.Contains("Дохід") ? "checked" : "")>
                        <label                class="form-check-
label">Дохід</label>
                    </div>
                    <div class="form-check form-check-inline">
```

```
                        <input           class="form-check-input"
type="checkbox"   name="FilterTypes"   checked   value="Витрата"
@(Model.FilterTypes.Contains("Витрата") ? "checked" : "")>
                        <label                 class="form-check-
label">Витрата</label>
                    </div>
                </div>

                <div class="filter-section">
                    <label>Категорія:</label>
                    <select     asp-for="FilterCategory"     asp-
items="Model.CategorySelectList" class="form-control select2">
                        <option value="">Всі</option>
                    </select>
                </div>

                <div class="filter-section">
                    <label>Діапазон дат:</label>
                    <div class="input-group">
                        <input    type="date"    name="StartDate"
value="@(Model.StartDate?.ToString("yyyy-MM-dd"))"   class="form-
control">
                        <input     type="date"     name="EndDate"
value="@(Model.EndDate?.ToString("yyyy-MM-dd"))"     class="form-
control">
                    </div>
                </div>

                <div class="filter-section">
                    <label>Фільтр за періодом:</label><br />
                    <button                            type="button"
onclick="changeChartType('day')" class="btn btn-outline-secondary
btn-sm">Доба</button>
                    <button                            type="button"
onclick="changeChartType('week')"     class="btn     btn-outline-
secondary btn-sm">Тиждень</button>
                    <button                            type="button"
onclick="changeChartType('month')"    class="btn    btn-outline-
secondary btn-sm">Місяць</button>
                </div>

                <div class="filter-section">
                    <button type="submit" class="btn btn-primary
btn-sm">Застосувати фільтри</button>
                    <a         href="@Url.Page("/Reports/Index")"
class="btn btn-secondary btn-sm">Скинути фільтри</a>
                </div>
            </form>
        </div>
    </div>
</div>
```

```
@section Scripts {

    <script
src="https://cdn.jsdelivr.net/npm/chart.js@3.9.1/dist/chart.min.
js"></script>

    <script     src="https://cdn.jsdelivr.net/npm/chartjs-plugin-
datalabels"></script>
    <script>
        document.addEventListener("DOMContentLoaded", function ()
{

            var                         ctx                         =
document.getElementById('expenseChart').getContext('2d');
            var chartType = 'pie'; // Початковий тип графіка
            var chart;

            function  generateChart(expenseData,  expenseLabels,
incomeData, incomeLabels, categoryColors, type, showLabels) {
                if (chart) {
                    chart.destroy(); // Знищити старий графік
                }

                // Створення унікального списку категорій
                var          allLabels          =          [...new
Set(expenseLabels.concat(incomeLabels))];

                // Формування мапи категорій до кольорів
                var colorMap = {};
                for (var category in categoryColors) {
                    if  (categoryColors.hasOwnProperty(category))
{
                        colorMap[category]                         =
categoryColors[category];
                    }
                }

                // Формування даних для витрат
                var expenseDataset = {
                    label: 'Витрати',
                    data: allLabels.map(label => {
                        var index = expenseLabels.indexOf(label);
                        return index >= 0 ? expenseData[index] :
0;
                    }),
                    backgroundColor:    allLabels.map(label    =>
colorMap[label] || 'rgba(0,0,0,0.1)'),
                    borderColor:      allLabels.map(label      =>
colorMap[label] || 'rgba(0,0,0,1)'),
                    borderWidth: 1
                };
```

```
                // Формування даних для доходів
                var incomeDataset = {
                    label: 'Доходи',
                    data: allLabels.map(label => {
                        var index = incomeLabels.indexOf(label);
                        return index >= 0 ? incomeData[index] : 0;
                    }),
                    backgroundColor:    allLabels.map(label    =>
colorMap[label] || 'rgba(0,0,0,0.1)'),
                    borderColor:      allLabels.map(label      =>
colorMap[label] || 'rgba(0,0,0,1)'),
                    borderWidth: 1
                };

                var datasets = [];

                // Додавання витрат до графіка, якщо є дані
                if (expenseData.length > 0 && expenseLabels.length
> 0 && !(expenseLabels.length == 1 && expenseLabels[0] == "Немає
даних")) {
                    datasets.push(expenseDataset);
                }

                // Додавання доходів до графіка, якщо є дані
                if (incomeData.length > 0 && incomeLabels.length
> 0 && !(incomeLabels.length == 1 && incomeLabels[0] == "Немає
даних")) {
                    datasets.push(incomeDataset);
                }

                chart = new Chart(ctx, {
                    type: type,
                    data: {
                        labels: allLabels,
                        datasets: datasets
                    },
                    options: {
                        responsive: true,
                        maintainAspectRatio: false,
                        plugins: {
                            datalabels: {
                                color: '#000',
                                anchor: 'center',
                                align: 'center',
                                offset: -10,
                                formatter: function (value) {
                                    return  value  ===  0  ?  ''  :
value.toLocaleString();
                                },
                                display: showLabels,
                                font: {
                                    size: 12,
```

```
                                weight: 'bold',
                            }
                        },
                        legend: {
                            position: 'top',
                            labels: {
                                padding: 15
                            }
                        },
                    },
                    scales: {
                        y: {
                            beginAtZero: true
                        }
                    }
                },
                plugins: [ChartDataLabels]
            });
        }

        var expenseData = @Html.Raw(JsonSerializer.Serialize(Model.ChartData));
        var expenseLabels = @Html.Raw(JsonSerializer.Serialize(Model.ChartLabels));
        var incomeData = @Html.Raw(JsonSerializer.Serialize(Model.IncomeData));
        var incomeLabels = @Html.Raw(JsonSerializer.Serialize(Model.IncomeLabels));
        var categoryColors = @Html.Raw(JsonSerializer.Serialize(Model.CategoryColors));

        var showDataLabels = true;
        generateChart(expenseData, expenseLabels, incomeData, incomeLabels, categoryColors, chartType, showDataLabels);
        var toggleDataLabelsCheckbox = document.getElementById('toggleDataLabels');
        toggleDataLabelsCheckbox.addEventListener('change', function () {
            showDataLabels = this.checked;
            generateChart(expenseData, expenseLabels, incomeData, incomeLabels, categoryColors, chartType, showDataLabels);
        });

        document.querySelectorAll('.change-chart-type').forEach(button => {
            button.addEventListener('click', function () {
                chartType = this.dataset.type; // Отримати тип з кнопки
                generateChart(expenseData, expenseLabels, incomeData, incomeLabels, categoryColors, chartType, showDataLabels); // Оновити графік
```

```
                });
            });
        });
    </script>
}
```

Reports/Index.cshtml.cs

```
using Microsoft.AspNetCore.Mvc.RazorPages;
using Microsoft.EntityFrameworkCore;
using FinanceOblik.Models;
using Microsoft.AspNetCore.Mvc.Rendering;
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;
using System;
using Microsoft.AspNetCore.Mvc;

namespace FinanceOblik.Pages.Reports
{
    public class IndexModel : PageModel
    {
        private readonly FinanceOblikContext _context;
        private readonly ILogger<IndexModel> _logger;

        public IndexModel(FinanceOblikContext context,
ILogger<IndexModel> logger)
        {
            _context = context;
            _logger = logger;
        }

        public IList<Transaction> Transactions { get; set; }

        public List<decimal> ChartData { get; set; }
        public List<string> ChartLabels { get; set; }

        public List<decimal> IncomeData { get; set; }
        public List<string> IncomeLabels { get; set; }

        public Dictionary<string, string> CategoryColors { get;
set; }

        public SelectList CategorySelectList { get; set; }

        [BindProperty(SupportsGet = true)]
        public List<string> FilterTypes { get; set; } = new
List<string>();

        [BindProperty(SupportsGet = true)]
        public string FilterCategory { get; set; }

        [BindProperty(SupportsGet = true)]
```

```
        public DateTime? StartDate { get; set; }

        [BindProperty(SupportsGet = true)]
        public DateTime? EndDate { get; set; }

        [BindProperty(SupportsGet = true)]
        public string ChartType { get; set; } = "pie";

        public async Task OnGetAsync(string period)
        {
            var transactions =
_context.Transactions.AsNoTracking().AsQueryable();

            if (FilterTypes != null && FilterTypes.Any())
            {
                transactions = transactions.Where(t =>
FilterTypes.Contains(t.Type));
            }

            if (!string.IsNullOrEmpty(FilterCategory))
            {
                transactions = transactions.Where(t =>
t.Category == FilterCategory);
            }

            if (StartDate.HasValue && EndDate.HasValue)
            {
                transactions = transactions.Where(t => t.Date >=
StartDate.Value && t.Date <= EndDate.Value);
            }

            if (!string.IsNullOrEmpty(period))
            {
                DateTime now = DateTime.Now;
                switch (period)
                {
                    case "day":
                        transactions = transactions.Where(t =>
t.Date.Date == now.Date);
                        break;
                    case "week":
                        var startOfWeek = now.AddDays(-
(int)now.DayOfWeek);
                        transactions = transactions.Where(t =>
t.Date >= startOfWeek);
                        break;
                    case "month":
                        var startOfMonth = new
DateTime(now.Year, now.Month, 1);
                        transactions = transactions.Where(t =>
t.Date >= startOfMonth);
                        break;
```

```
                }
            }

            Transactions = await transactions.ToListAsync();

            _logger.LogInformation($"Total Transactions after
filtering: {Transactions.Count}");
            _logger.LogInformation($"Expense Transactions:
{Transactions.Count(t => t.Type == "Витрата")}");
            _logger.LogInformation($"Income Transactions:
{Transactions.Count(t => t.Type == "Дохід")}");

            if (FilterTypes.Contains("Витрата"))
            {
                var expenseTransactions = Transactions.Where(t
=> t.Type == "Витрата").ToList();
                var groupedExpense =
expenseTransactions.GroupBy(t => t.Category)

.Select(g => new { Category = g.Key, Total = g.Sum(t =>
t.Amount) })

.OrderByDescending(g => g.Total)

.ToList();

                ChartLabels = groupedExpense.Select(g =>
g.Category).ToList();
                ChartData = groupedExpense.Select(g =>
g.Total).ToList();
            }
            else
            {
                ChartLabels = new List<string>();
                ChartData = new List<decimal>();
            }

            if (FilterTypes.Contains("Дохід"))
            {
                var incomeTransactions = Transactions.Where(t =>
t.Type == "Дохід").ToList();
                var groupedIncome = incomeTransactions.GroupBy(t
=> t.Category)
                                                      .Select(g
=> new { Category = g.Key, Total = g.Sum(t => t.Amount) })

.OrderByDescending(g => g.Total)
                                                      .ToList();

                IncomeLabels = groupedIncome.Select(g =>
g.Category).ToList();
```

```
                IncomeData = groupedIncome.Select(g =>
g.Total).ToList();
            }
            else
            {
                IncomeLabels = new List<string>();
                IncomeData = new List<decimal>();
            }

            var allCategories = new
HashSet<string>(ChartLabels.Concat(IncomeLabels));

            // Список попередньо визначених кольорів
            var predefinedColors = new List<string>
            {
                "#FF6384", "#36A2EB", "#FFCE56", "#4BC0C0",
                "#9966FF", "#FF9F40", "#C9CBCF", "#FF5733",
                "#33FF57", "#3357FF", "#F333FF", "#33FFF3",
                "#F3FF33", "#FF33F3", "#33F3FF", "#A133FF",
                "#33FFA1", "#FFA133", "#A1FF33", "#FF33A1"
            };

            CategoryColors = new Dictionary<string, string>();
            int colorIndex = 0;
            Random random = new Random();

            foreach (var category in allCategories)
            {
                if (colorIndex < predefinedColors.Count)
                {
                    CategoryColors[category] =
predefinedColors[colorIndex];
                    colorIndex++;
                }
                else
                {
                    // Генерація випадкового кольору, якщо
попередньо визначені закінчилися
                    string randomColor = $"#{random.Next(0,
16777215).ToString("X6")}";
                    CategoryColors[category] = randomColor;
                }
            }

            var allCategoriesList = new List<string>
            {
                "Стипендія", "Зарплата", "Премія", "Подарунок",
"Дивіденди", "Проценти", "Виграш", "Знахідка", "Інше",
                "Покупки", "Комунальні платежі", "Їжа",
"Транспорт", "Спорт", "Авто", "Одяг", "Ресторан", "Розваги",
                "Кіно", "Таксі", "Меблі", "Послуги", "Здоров'я",
"Інтернет", "Мобільний зв'язок"
```

```
            };

            if (FilterTypes.Contains("Витрата") &&
!FilterTypes.Contains("Дохід"))
            {
                var expenseCategories = new List<string>
                {
                    "Покупки", "Комунальні платежі", "Їжа",
"Транспорт", "Спорт", "Авто", "Одяг", "Ресторан",
                    "Розваги", "Кіно", "Таксі", "Меблі",
"Послуги", "Здоров'я", "Інтернет", "Мобільний зв'язок"
                };
                CategorySelectList = new
SelectList(expenseCategories);
            }
            else if (FilterTypes.Contains("Дохід") &&
!FilterTypes.Contains("Витрата"))
            {
                var incomeCategories = new List<string>
                {
                    "Стипендія", "Зарплата", "Премія",
"Подарунок", "Дивіденди", "Проценти", "Виграш", "Знахідка",
"Інше"
                };
                CategorySelectList = new
SelectList(incomeCategories);
            }
            else
            {
                CategorySelectList = new
SelectList(allCategoriesList);
            }
            if ((ChartData == null || ChartLabels == null ||
!ChartData.Any() || !ChartLabels.Any()) &&
FilterTypes.Contains("Витрата"))
            {
                ChartData = new List<decimal> { 0 };
                ChartLabels = new List<string> { "Немає даних"
};
            }

            if ((IncomeData == null || IncomeLabels == null ||
!IncomeData.Any() || !IncomeLabels.Any()) &&
FilterTypes.Contains("Дохід"))
            {
                IncomeData = new List<decimal> { 0 };
                IncomeLabels = new List<string> { "Немає даних"
};
            }
        }
    }
}
```

EfficiencyAnalysis/ EfficiencyAnalysis.cshtml.cs

```
using Microsoft.AspNetCore.Mvc.RazorPages;
using Microsoft.EntityFrameworkCore;
using FinanceOblik.Models;
using Microsoft.Extensions.Logging;
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;
using System;
using Microsoft.AspNetCore.Mvc;

namespace FinanceOblik.Pages
{
    public class EfficiencyAnalysisModel : PageModel
    {
        private readonly FinanceOblikContext _context;
        private readonly ILogger<EfficiencyAnalysisModel>
_logger;

        public EfficiencyAnalysisModel(FinanceOblikContext
context, ILogger<EfficiencyAnalysisModel> logger)
        {
            _context = context;
            _logger = logger;
        }

        public decimal TotalIncome { get; set; }
        public decimal TotalExpenses { get; set; }

        [BindProperty(SupportsGet = true)]
        public DateTime? StartDate { get; set; }

        [BindProperty(SupportsGet = true)]
        public DateTime? EndDate { get; set; }

        [BindProperty(SupportsGet = true)]
        public string Period { get; set; } = "month";

        public async Task OnGetAsync()
        {
            var transactions =
_context.Transactions.AsNoTracking().AsQueryable();

            DateTime now = DateTime.Now;
            switch (Period.ToLower())
            {
                case "day":
                    transactions = transactions.Where(t =>
t.Date.Date == now.Date);
                    break;
                case "week":
```

```
                    var startOfWeek = now.AddDays(-
(int)now.DayOfWeek);
                    transactions = transactions.Where(t =>
t.Date >= startOfWeek.Date);
                    break;
                case "month":
                default:
                    var startOfMonth = new DateTime(now.Year,
now.Month, 1);
                    transactions = transactions.Where(t =>
t.Date >= startOfMonth);
                    break;
            }

            if (StartDate.HasValue && EndDate.HasValue)
            {
                transactions = transactions.Where(t => t.Date >=
StartDate.Value && t.Date <= EndDate.Value);
            }

            var filteredTransactions = await
transactions.ToListAsync();

            _logger.LogInformation($"Total Transactions after
filtering: {filteredTransactions.Count}");

            TotalIncome = filteredTransactions.Where(t => t.Type
== "Дохід").Sum(t => (decimal?)t.Amount) ?? 0;
            TotalExpenses = filteredTransactions.Where(t =>
t.Type == "Витрата").Sum(t => (decimal?)t.Amount) ?? 0;

            _logger.LogInformation($"Total Income:
{TotalIncome}");
            _logger.LogInformation($"Total Expenses:
{TotalExpenses}");
        }
    }
}
```