НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ БІОРЕСУРСІВ І ПРИРОДОКОРИСТУВАННЯ УКРАЇНИ

Факультет інформаційних технологій

ДОПУСКАЄТЬСЯ ДО ЗАХИСТУ

Завідувач кафедри
комп'ютерних наук
(назва кафедри)

_____ / Голуб Б.Л. /
(підпис) (ПІБ)

"___" _____ 2025 р.

БАКАЛАВРСЬКА КВАЛІФІКАЦІЙНА РОБОТА
на тему

«Система управління особистими фінансами: Розробка веб-додатку для обліку доходів та витрат»

Спеціальність 121 – «Інженерія програмного забезпечення»

Гарант освітньої програми
к.т.н, доцент _____ Вайганг Г.О.
(науковий ступінь та вчене звання) (підпис) (ПІБ)

Керівник бакалаврської кваліфікаційної роботи
асистент _____ _____ Баранова Т. А.
(науковий ступінь та вчене звання) (підпис) (ПІБ)

Консультант бакалаврської кваліфікаційної роботи
к.т.н, доцент _____ _____ Даков С.Ю.
(науковий ступінь та вчене звання) (підпис) (ПІБ)

Виконав _____ Вільчинський Валентин Ігорович
(підпис) (ПІБ студента)

КИЇВ – 2025

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ БІОРЕСУРСІВ І ПРИРОДОКОРИСТУВАННЯ УКРАЇНИ
Факультет інформаційних технологій

ЗАТВЕРДЖУЮ
Завідувач кафедри
_____комп'ютерних наук_____
(назва кафедри)

___к.т.н., доцент_____ _____Голуб Б.Л.
(науковий ступінь, вчене звання) (підпис) (ПІБ)
"___" _____ 2025 р.

З А В Д А Н Н Я
на виконання бакалаврської кваліфікаційної роботи студенту
Вільчинський Валентин Ігорович

Спеціальність 121 – «Інженерія програмного забезпечення»

1. Тема бакалаврської кваліфікаційної роботи «Система управління особистими фінансами: Розробка веб-додатку для обліку доходів та витрат» затверджена наказом ректора НУБіП України від 16.12.2024 № 2248 «С»
2. Термін подання завершеної роботи на кафедру_____
(рік, місяць, число)
3. Вихідні дані до роботи: надання інформації про облік власних доходів та витрат у вигляді категорій, а також аналіз своїх фінансових даних у вигляді звітності.
4. Перелік питань, що розглядаються:
   - Аналіз проблемної області
   - Моделювання предметної області
   - Проєктування програмної системи
   - Впровадження та експлуатація системи

Дата видачі завдання "16"___грудня____ 2024 р.

Керівник бакалаврської кваліфікаційної роботи
___асистент _____ _____ _Баранова Т. А._
(науковий ступінь та вчене звання) (підпис) (ПІБ)

Консультант бакалаврської кваліфікаційної роботи
____к.т.н, доцент _____ _____ _ Даков С.Ю.
(науковий ступінь та вчене звання) (підпис) (ПІБ)

Завдання прийняв до виконання:_____/ Вільчинський В.І. /
(підпис) (прізвище та ініціали)

ЗМІСТ

ВСТУП

У сучасному швидкоплинному цифровому суспільстві особиста фінансова обізнаність стає дедалі важливішою. Люди стикаються з широким спектром фінансових обов'язків, включаючи складання бюджету, відстеження витрат, заощадження та планування майбутнього. Однак багато людей досі покладаються на ручні або непослідовні методи управління своїми фінансами, що часто призводить до поганих фінансових рішень, перевитрат та відсутності заощаджень.

Швидкий розвиток веб-технологій відкрив нові можливості для створення інтуїтивно зрозумілих та доступних інструментів, що допомагають у фінансовому самокеруванні. У відповідь на цю зростаючу потребу, у цій роботі представлено проєктування та впровадження веб-системи управління особистими фінансами. Система надає користувачам централізовану платформу для відстеження їхніх доходів та витрат, управління категоріями транзакцій та аналізу їхніх фінансових звичок з часом.

Основні цілі системи:

- забезпечити безпечний та зручний інтерфейс для запису фінансових транзакцій;
- дозволити користувачам визначати власні категорії доходів та витрат;
- забезпечити детальний аналіз та візуалізацію фінансових даних;
- підтримка прийняття обґрунтованих фінансових рішень шляхом відстеження бюджету та зведень.

Цей веб-застосунок створено за допомогою PHP та фреймворку Symfony, що забезпечує масштабовану та добре структуровану архітектуру бекенду. База даних MySQL використовується для надійного зберігання та управління даними. Фронтенд-технології, такі як Twig та Chart.js,

використовуються для візуалізації динамічних, адаптивних та інтерактивних інтерфейсів користувача.

Кваліфікаційна робота структурована таким чином, щоб охопити весь процес розробки, включаючи аналіз ринку існуючих фінансових інструментів, збір вимог, проєктування системи, впровадження, тестування та можливі майбутні вдосконалення. Кінцевий продукт має на **меті** допомогти користувачам краще контролювати свої особисті фінанси, застосувавши сучасні методи веб-розробки для вирішення повсякденних проблем.

Актуальність цього проєкту додатково підтверджується зростаючою тенденцією до фінансової грамотності та незалежності, особливо серед молодих користувачів та тих, хто керує фріланс-працею або має нерегулярні джерела доходу. Завдяки інтеграції сучасних веб-технологій та застосуванню передового досвіду в галузі безпеки та зручності використання, ця робота демонструє, як доступна платформа особистих фінансів може бути одночасно освітньою та розширювати можливості. Система, розроблена в рамках цього дослідження, має на меті подолати розрив між фінансовою обізнаністю та зручністю щоденного використання, пропонуючи практичний інструмент, який допомагає користувачам контролювати своє економічне благополуччя.

Незважаючи на існування різних комерційних та відкритих програм для особистих фінансів, багатьом з них або бракує гнучкості, або є надмірно складними для пересічних користувачів, або накладають обмеження щодо налаштування та доступності. Деякі інструменти вимагають від користувачів адаптації до попередньо визначених категорій, тоді як інші обмежують повну функціональність за платними вікнами. Цей проєкт вирішує ці проблеми, зосереджуючись на простоті, налаштованості та вільному доступі. Користувачі мають можливість створювати власні категорії, фільтрувати транзакції за часом та типом, а також миттєво візуалізувати свої фінансові моделі, і все це в чистому та інтуїтивно зрозумілому інтерфейсі.

1 АНАЛІЗ ПРОБЛЕМНОЇ ОБЛАСТІ

1.1 Опис предметної області

Ефективне управління особистими фінансами є вирішальним компонентом фінансового благополуччя людини. Воно включає планування, організацію та контроль фінансової діяльності, такої як заробіток, витрати, заощадження та інвестування. Незважаючи на зростаючу важливість фінансової обізнаності, багато людей намагаються вести точний облік своїх доходів і витрат, що призводить до поганого складання бюджету, неконтрольованих боргів та недосягнення фінансових цілей. У багатьох випадках фінансові проблеми виникають не через недостатній дохід, а через брак прозорості щодо витратних звичок та погане планування.

Традиційно люди використовували блокноти, електронні таблиці або прості мобільні нотатки для запису своїх фінансів. Хоча ці методи пропонують гнучкість, їм бракує автоматизації, цілісності даних, аналітичних можливостей і часто займають багато часу. Крім того, вони не забезпечують зворотного зв'язку в режимі реального часу або тенденцій, які можуть допомогти користувачам приймати обґрунтовані фінансові рішення. В результаті користувачі часто втрачають інтерес до постійного ведення таких записів.

За останні роки з'явилося безліч цифрових рішень, включаючи мобільні додатки, інструменти онлайн-банкінгу та програмне забезпечення для настільних комп'ютерів, які спрямовані на спрощення фінансового відстеження. Однак багато з цих інструментів є або занадто складними для звичайних користувачів, або не мають можливості налаштування, або вимагають оплати за доступ до основних функцій. Крім того, мало хто пропонує повну прозорість щодо обробки даних або надає опції експорту та аналізу даних, що виходять за рамки базових діаграм.

Зростає попит на настроювані, зручні для користувача та безпечні системи, які дозволяють людям адаптувати фінансове відстеження до своїх

конкретних потреб. Система повинна підтримувати користувацькі категорії доходів і витрат, забезпечувати комплексну візуалізацію даних та працювати на різних пристроях. Вона також повинна забезпечувати конфіденційність та безпеку фінансової інформації, особливо в контексті зростання кіберзагроз та побоювань щодо неправомірного використання даних [1].

Ця робота розглядає ці проблеми, пропонуючи розробку веб-системи управління особистими фінансами. Система має на меті поєднати простоту, налаштування та потужні аналітичні інструменти, надаючи користувачам повний контроль над своїми фінансовими даними в безпечному та доступному середовищі.

Тематика цієї роботи лежить в галузі управління особистими фінансами, яка охоплює процеси планування, відстеження та аналізу фінансової діяльності людини. Основна мета особистих фінансів полягає в тому, щоб допомогти людям ефективно керувати своїми доходами, витратами, заощадженнями та інвестиціями для досягнення фінансової стабільності та довгострокових цілей. Ця галузь включає такі поняття, як бюджетування, фінансове планування, відстеження витрат, управління боргами та оптимізація заощаджень.

Зі зростанням складності сучасного життя та зростанням цифрових транзакцій людям стає все складніше підтримувати точний огляд свого фінансового становища. Витрати часто розподіляються по різних каналах — готівка, картки, мобільні додатки, онлайн-підписки — що робить ручне відстеження неефективним та схильним до помилок. Як наслідок, існує очевидна потреба в цифрових інструментах, які можуть допомогти агрегувати та організовувати фінансові дані централізованим та зручним для користувача способом [2].

Система управління особистими фінансами має на меті заповнити цю прогалину, пропонуючи користувачам інтерфейс, де вони можуть:

- реєструвати та класифікувати доходи та витрати;
- переглядати свій поточний баланс та історичні тенденції;

- аналізувати свою фінансову поведінку за допомогою статистики та візуалізації;
- встановлювати особисті бюджети або фінансові цілі;
- експорт фінансових звітів для використання офлайн або подальшого аналізу.

Веб-додаток, розроблений у цій роботі, зосереджений на забезпеченні такої функціональності гнучким та інтуїтивно зрозумілим способом. На відміну від корпоративних бухгалтерських систем, які вимагають складних знань бухгалтерського обліку, ця система адаптована для звичайних користувачів з невеликим досвідом або без нього у фінансовій сфері. Вона дозволяє користувачам визначати власні категорії доходів і витрат, адаптувати систему до своїх фінансових звичок та отримувати доступ до фінансової інформації в режимі реального часу.

Окрім зручності використання, безпека даних та конфіденційність є критичними аспектами предметної області. Оскільки система має справу з конфіденційними фінансовими даними, вона повинна впроваджувати надійні заходи безпеки, включаючи безпечну автентифікацію, шифрування даних та належні механізми контролю доступу. Ці міркування є важливими для формування довіри користувачів та забезпечення дотримання загальних стандартів захисту даних.

Детальний опис класів та атрибутів предметної області наведено у таблиці 1.1.

Таблиця 1.1

Опис атрибутів класів предметної області

Клас предметної області	**Атрибут**	**Опис**
Користувач	Ім’я користувача	Ім’я або псевдонім, який обирає користувач
	Електронна адреса	Адреса для входу в систему та комунікації
	Секретний код	Засіб захисту облікового запису (пароль)
	Фото профілю	Зображення, що персоніфікує обліковий запис
	Дата реєстрації	Коли користувач вперше зареєструвався в системі
Категорія	Назва	Назва групи доходів або витрат, яку бачить користувач
	Призначення	Визначає, чи це категорія для доходів чи для витрат
	Власник	Користувач, який створив і використовує цю категорію
Транзакція	Сума	Обсяг коштів, що надійшли або були витрачені
	Дата операції	День, коли відбулася відповідна фінансова дія
	Опис	Коротка примітка чи пояснення щодо транзакції
	Категорія	Група (доходів/витрат), до якої належить ця транзакція
	Автор запису	Користувач, який додав цей запис у систему

1.2 Огляд існуючих рішень

Щоб краще зрозуміти потреби користувачів та функціональний обсяг інструментів для роботи з особистими фінансами, важливо проаналізувати існуючі рішення в цій галузі. На ринку доступні численні програми та платформи, кожна з яких пропонує різні рівні складності, функції та цільову аудиторію. Найвідоміші інструменти включають CoinKeeper, YNAB (You Need A Budget), Spendee та 1Money, а також функції, інтегровані в банківські програми.

CoinKeeper – це широко використовуваний мобільний додаток для управління особистими фінансами, особливо відомий своїм візуально привабливим та інтерактивним підходом до складання бюджету та відстеження витрат. На відміну від багатьох традиційних інструментів, CoinKeeper представляє фінанси за допомогою інтерфейсу перетягування монет, де користувачі розподіляють кошти з джерел доходу на різні категорії витрат та заощаджень. Цей метод робить процес складання бюджету більш відчутним та зручним для користувача, особливо для тих, хто навчається візуально [3].

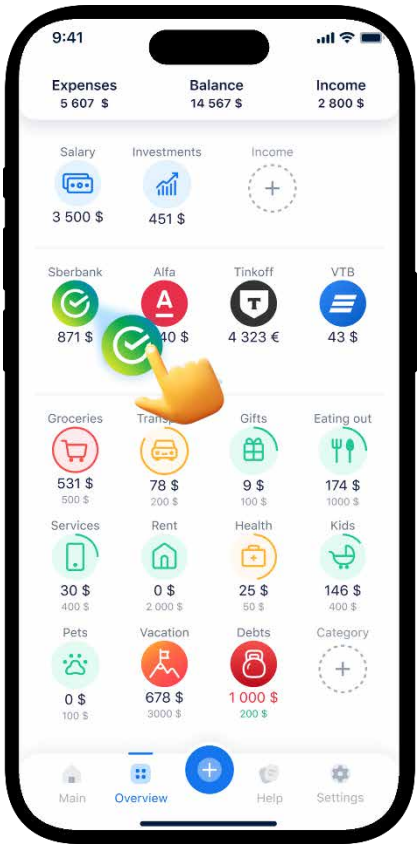

Рис. 1 Мобільний застосунок “CoinKeeper”

Система дозволяє користувачам створювати різні гаманці, такі як банківські рахунки, готівка або заощадження, та контролювати їхні баланси в режимі реального часу. Кожна категорія витрат (наприклад, їжа, транспорт, розваги) представлена окремим розділом на головному екрані, що дозволяє користувачам одразу бачити, куди йдуть їхні гроші. Користувачі можуть встановлювати щомісячні ліміти для категорій та отримувати сповіщення, коли вони наближаються до цих лімітів або перевищують їх, заохочуючи до більш усвідомлених витрат.

Окрім ручного введення, CoinKeeper підтримує заплановані повторювані транзакції, допомагаючи користувачам відстежувати фіксовані витрати, такі як орендна плата або підписки. Він також включає базову аналітику та звіти, які показують тенденції витрат з часом, допомагаючи користувачам виявляти проблемні області у своїй фінансовій поведінці [3].

Додаток доступний для iOS та Android і пропонує синхронізацію між кількома пристроями з обліковим записом CoinKeeper. Однак, хоча

безкоштовна версія забезпечує основний функціонал, більшість розширених функцій, таких як детальна статистика, підтримка кількох пристроїв та синхронізація банківських рахунків, заблоковані за преміум-підпискою. Це обмежує доступність для користувачів, які не готові платити за повну версію або яким потрібні лише базові функції відстеження.

Ще одним недоліком є обмежена доступність банківських інтеграцій, оскільки CoinKeeper в першу чергу орієнтований на певні ринки. Користувачам за межами підтримуваних регіонів фінансові дані необхідно вводити вручну, що може займати багато часу та бути менш ефективним. Більше того, хоча барвистий інтерфейс приваблює багатьох користувачів, інші можуть вважати його менш придатним для професійного або довгострокового фінансового планування.

Попри ці обмеження, CoinKeeper залишається сильним вибором для користувачів, які шукають візуальний, спрощений та привабливий інструмент для управління своїми щоденними фінансами. Однак його залежність від платної моделі та обмежена гнучкість для користувацьких функцій роблять його менш ідеальним для тих, хто надає пріоритет повному контролю, портативності даних або безкоштовному доступу — ключовим міркуванням, які розглядаються під час розробки системи, запропонованої в цій роботі.

Розглянемо інше програмне рішення на ринку.

1Money – це додаток для управління особистими фінансами, який спрощує процес відстеження доходів і витрат. Його головна привабливість полягає в зручному інтерфейсі, що дозволяє людям легко керувати своїми фінансами, не відчуваючи себе перевантаженими складними функціями. Розроблений для простоти, додаток дозволяє користувачам створювати кілька облікових записів або гаманців для різних типів фінансів, таких як готівка, банківські рахунки та цифрові гаманці. Ця функція дозволяє користувачам відстежувати свою фінансову активність з різних джерел без необхідності використовувати окремі додатки для кожного облікового запису [4].

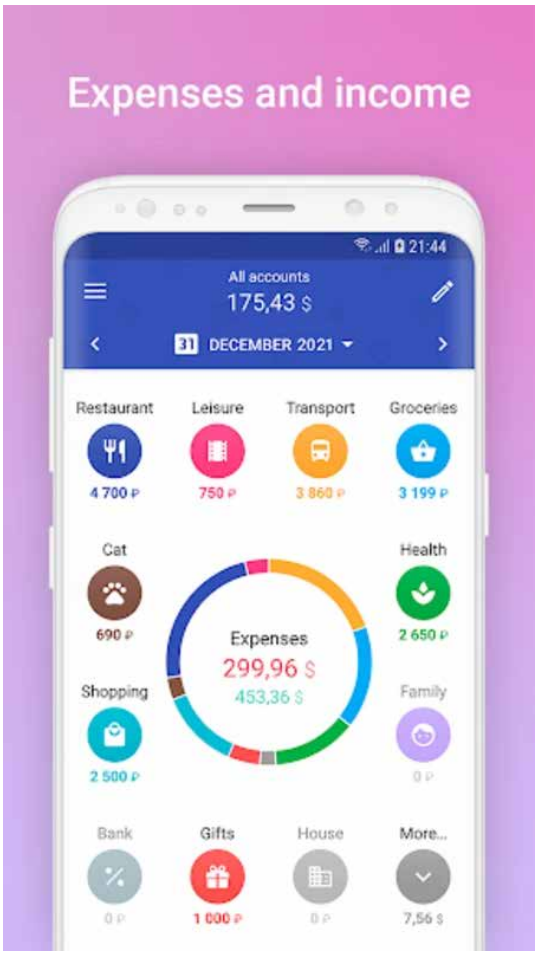

Рис. 2 Мобільний застосунок “1Money”

Однією з ключових переваг 1Money є його проста категоризація витрат. Користувачі можуть створювати власні категорії або використовувати попередньо визначені, такі як «їжа», «транспорт» та «розваги». Це допомагає користувачам залишатися організованими та легко відстежувати свої витрати в різних сферах свого життя. Додаток також підтримує кілька валют, що особливо корисно для тих, хто подорожує або керує фінансами в різних регіонах. Крім того, 1Money дозволяє вручну вводити транзакції, а також повторювані транзакції для регулярних витрат, таких як рахунки або підписки, що спрощує автоматизацію та управління послідовними фінансовими зобов'язаннями.

Функції бюджетування програми є ще однією родзинкою, що дозволяє користувачам встановлювати ліміти витрат для кожної категорії. Сповіщення спрацьовують, коли витрати наближаються до цих лімітів або перевищують їх, допомагаючи користувачам контролювати свої фінанси. Чистий та простий

дизайн поєднується з візуальними зведеннями, такими як діаграми та графіки, які забезпечують легкий для розуміння момент витрачання грошей. Ця простота та зрозумілість роблять його чудовим інструментом для людей, які віддають перевагу простому підходу до управління своїми особистими фінансами [4].

Однак, хоча 1Money ефективний для базового фінансового відстеження, є деякі обмеження, які слід враховувати. Функція синхронізації банків не така бездоганна, як деякі інші фінансові програми, а це означає, що користувачам, можливо, все одно доведеться вручну вводити свої транзакції, якщо програма не підтримує автоматичну синхронізацію з місцевими банками. Крім того, хоча безкоштовна версія пропонує основні функції, багато розширених інструментів, таких як синхронізація кількох пристроїв та більш детальні фінансові звіти, заблоковані за преміум-підпискою. Це може бути недоліком для користувачів, які не готові платити за додаткові функції.

Хоча 1Money може не надавати такого ж рівня глибокого фінансового аналізу чи складних інструментів, як конкуренти, такі як YNAB чи Mint, його простий дизайн та простота роблять його чудовим вибором для тих, хто новачок в управлінні особистими фінансами. Він особливо підходить для людей, яким потрібен простий та ефективний спосіб відстеження своїх щоденних витрат без потреби в розширених функціях бюджетування чи обширних фінансових звітах.

На завершення, 1Money пропонує доступне та практичне рішення для управління особистими фінансами. Його простий та інтуїтивно зрозумілий інтерфейс, а також корисні інструменти категоризації та бюджетування роблять його надійним варіантом для користувачів, які хочуть відстежувати свої фінанси без складності більш просунутих платформ.

1.3 Постановка завдання

Ця програма відповідає за ведення обліку доходів та витрат. Це включає відстеження різних фінансових показників, таких як щомісячний дохід, категорії витрат, цілі заощаджень та бюджетні ліміти. Метою системи є створення комплексної бази даних, де користувачі можуть легко вводити свої фінансові дані та отримувати персоналізовані сповіщення, щоб залишатися в межах своїх фінансових цілей.

Основна програмна система, що розробляється, надасть користувачам можливість швидко отримувати інформацію про:

- ефективність стратегій бюджетування;
- категоризацію та відстеження доходів та витрат;
- стан цілей та завдань заощаджень;
- сповіщення про перевитрат коштів у різних фінансових категоріях.

Система буде налаштована для роботи в діалоговому інтерфейсі на основі зручного меню. Користувачі зможуть взаємодіяти з програмним забезпеченням за допомогою простих підказок та команд. На основі введених користувачем даних система відображатиме відповідні дані, такі як зведення доходів, розбивка витрат та сповіщення про будь-які фінансові розбіжності. Користувачі також можуть вносити корективи до своїх записів, якщо необхідно, скасовуючи або змінюючи раніше зареєстровані транзакції чи цілі.

Така конфігурація забезпечує легку навігацію та персоналізоване фінансове відстеження, гарантуючи, що користувачі контролюють свої фінанси та можуть приймати обґрунтовані рішення на основі даних у режимі реального часу.

1.4 Функціональні та нефункціональні вимоги

Функціональні вимоги:

1. система повинна дозволяти користувачам реєструватися, надаючи свої особисті дані (ім'я, електронну пошту, пароль) та проходити автентифікацію за допомогою підтвердження електронної пошти. Тільки автентифіковані користувачі повинні мати доступ до своїх фінансових даних та керувати ними;
2. користувачі повинні мати можливість додавати доходи та витрати, вказувати категорії (наприклад, зарплата, продукти харчування, транспорт), а також вводити суму, дату та короткий опис. Система повинна дозволяти користувачам редагувати або видаляти будь-які записи за потреби;
3. користувачі повинні мати можливість створювати власні категорії для доходів та витрат. Попередньо визначені категорії (наприклад, комунальні послуги, розваги) також повинні бути доступні для оптимізації категоризації;
4. система повинна дозволяти користувачам встановлювати щомісячний бюджет для кожної категорії витрат. Сповіщення повинні спрацьовувати, коли користувач наближається до або перевищує свій бюджетний ліміт, забезпечуючи активне управління фінансами;
5. додаток повинен надавати аналітику у вигляді кругових діаграм або гістограм, які відображають розподіл доходів та витрат за категоріями, допомагаючи користувачам візуалізувати свої фінансові тенденції [5];

6. система повинна дозволяти користувачам створювати фінансові звіти за певні періоди (наприклад, щоденні, щомісячні, щорічні) для перегляду своїх доходів, витрат та заощаджень;
7. користувачі повинні мати можливість переглядати повну історію своїх транзакцій, включаючи доходи та витрати, з можливістю фільтрації за датою, категорією або сумою.

Нефункціональні вимоги:

1. система повинна бути адаптивною та мати можливість обробляти зростаючу кількість транзакцій з мінімальними затримками під час завантаження та обробки. Додаток має бути оптимізований для обробки великих наборів даних без шкоди для швидкості;
2. система повинна впроваджувати надійні заходи безпеки, включаючи шифрування конфіденційних даних (наприклад, паролів, фінансових транзакцій) як під час передачі, так і в стані спокою. Автентифікація користувачів повинна відповідати найкращим практикам, таким як двофакторна автентифікація (2FA), для запобігання несанкціонованому доступу [6];
3. система повинна мати інтуїтивно зрозумілий та зручний інтерфейс. Користувачі повинні мати можливість без проблем переміщатися між розділами (наприклад, управління доходами, створення звітів, бюджетування). Для нових користувачів повинні бути надані чіткі інструкції та підказки, щоб вони могли зрозуміти, як ефективно використовувати систему;
4. система повинна мати можливість масштабуватися та обробляти зростаючу кількість користувачів і транзакцій з часом без значного падіння продуктивності;
5. веб-застосунок має бути сумісним з основними браузерами (наприклад, Chrome, Firefox, Safari, Edge) та адаптуватися до

екранів різних розмірів та пристроїв (настільний комп’ютер, планшет, мобільний телефон);

6. система повинна містити надійний механізм резервного копіювання, щоб запобігти втраті даних у разі збою. Користувачі повинні мати можливість відновлювати свої фінансові дані у разі випадкового видалення або системних помилок.

Ці вимоги визначають як основну функціональність, так і загальні очікування щодо якості Системи управління особистими фінансами, забезпечуючи її безперебійну, безпечну та ефективну роботу для своїх користувачів.

1.5 Вимоги до інтерфейсу користувача

Інтерфейс користувача (UI) Системи управління особистими фінансами є ключовим компонентом у забезпеченні інтуїтивно зрозумілого та безперебійного досвіду для користувачів, що дозволяє їм ефективно керувати своєю фінансовою інформацією. Дизайн повинен пріоритезувати простоту та ясність, гарантуючи, що користувачі можуть безперешкодно переміщатися по системі. Кожна сторінка повинна служити чіткій меті, а інформація має бути представлена в добре організованому та легкому для читання форматі. Загальний макет повинен уникати візуального безладу, зосереджуючись лише на основних функціях, пропонуючи користувачам зручний простір для управління своїми фінансами.

Навігація по всій системі має бути інтуїтивно зрозумілою та простою. Користувачі повинні мати легкий доступ до ключових функцій, таких як управління доходами, витратами, бюджетами та створення звітів, і все це без труднощів. Чітка та послідовна структура навігації — чи то через головне меню, панель інструментів чи бічну панель — гарантує, що користувачі можуть швидко знайти те, що їм потрібно. Кнопки та інтерактивні елементи

повинні бути чітко позначені, а підказки або візуальні підказки можуть допомогти користувачам зрозуміти, як взаємодіяти з системою. Це допомагає уникнути будь-яких розчарувань під час взаємодії користувача.

Дизайн має бути повністю адаптивним, адаптуючись до різних пристроїв та розмірів екранів. Незалежно від того, чи користувач отримує доступ до системи на настільному комп'ютері, планшеті чи смартфоні, інтерфейс повинен плавно адаптуватися, забезпечуючи доступність та функціональність усіх функцій незалежно від платформи. Ця гнучкість є ключовою для надання користувачам можливості керувати своїми фінансами в дорозі, забезпечуючи стабільний та плавний досвід роботи на всіх пристроях.

Естетика відіграє значну роль в інтерфейсі користувача. Додаток повинен мати сучасний та професійний вигляд, з ретельно підібраними колірними схемами, шрифтами та піктограмами, що забезпечують візуальну гармонію. Така увага до дизайну гарантує, що інтерфейс буде не лише функціональним, але й приємним у використанні. Послідовний брендинг у всьому додатку допоможе підкреслити ідентичність платформи, зробивши її впізнаваною та надійною для користувачів [7].

Фінансові дані повинні бути представлені чітко та зрозуміло. Користувачі повинні мати можливість легко вводити та переглядати транзакції, такі як доходи та витрати, з можливістю їх ефективної категоризації. Графіки та діаграми можуть пропонувати візуальне представлення фінансових даних, надаючи користувачам швидкий огляд їхніх доходів та витрат. Наприклад, кругові діаграми або стовпчикові діаграми можуть показувати розподіл різних витрат, дозволяючи користувачам візуалізувати, куди йдуть їхні гроші.

Система також повинна забезпечувати персоналізований користувацький досвід. Після входу в систему користувачі побачать налаштовану панель інструментів, яка надає огляд їхнього фінансового стану. Тут мають бути представлені ключові дані, такі як загальний дохід, витрати та стан бюджету, з швидким доступом до детальних переглядів для глибшого

занурення у фінансову діяльність. Сповіщення та повідомлення про обмеження бюджету, майбутні платежі або очікувані транзакції повинні відображатися ненав'язливо, щоб користувачі були в курсі подій, не порушуючи їхнього робочого процесу.

Критичним аспектом системи є забезпечення доступності для всіх користувачів. Інтерфейс користувача повинен відповідати стандартам доступності, що робить його зручним для людей з інвалідністю. Такі функції, як навігація за допомогою клавіатури, сумісність з програмою зчитування з екрана та опції зміни розміру тексту, забезпечують комфортну взаємодію всіх користувачів з платформою. Дизайн також повинен враховувати колірний контраст для людей з вадами зору, забезпечуючи розрізнення тексту та кнопок для всіх.

У систему мають бути інтегровані можливості пошуку та фільтрації, що дозволить користувачам швидко знаходити певні транзакції, категорії або звіти. Фільтруючи дані за різними критеріями, такими як дата, категорія або сума, користувачі можуть легко керувати своїми фінансовими записами. Крім того, інтерфейс користувача повинен підтримувати узгодженість на всіх сторінках, забезпечуючи однаковий стиль таких елементів, як кнопки, форми та шрифти, для створення цілісного користувацького досвіду.

Підсумовуючи, інтерфейс користувача Системи управління особистими фінансами повинен пріоритезувати зручність використання, зрозумілість та доступність. Він повинен поєднувати чистий дизайн із надійною функціональністю, роблячи управління фінансами максимально простим. Забезпечуючи безперебійний досвід на всіх пристроях та гарантуючи, що платформа є інтуїтивно зрозумілою, безпечною та візуально привабливою, програма допоможе користувачам впевнено контролювати свої особисті фінанси.

2 МОДЕЛЮВАННЯ ПРЕДМЕТНОЇ ОБЛАСТІ

2.1 Загальні відомості

UML, або Уніфікована мова моделювання, – це широко використовуваний інструмент у розробці програмного забезпечення, призначений для забезпечення стандартизованого способу візуалізації, визначення та документування структури та поведінки програмної системи. Він відіграє вирішальну роль, особливо в об'єктно-орієнтованій розробці, допомагаючи розробникам, дизайнерам та іншим зацікавленим сторонам краще зрозуміти, як система працює та взаємодіє. UML складається з різних діаграм, кожна з яких має свою мету, від ілюстрації статичних структур до зображення динамічних взаємодій у системі [8].

Структурні діаграми зосереджені на представленні статичних аспектів системи. Вони демонструють архітектуру та компоненти системи, виділяючи зв'язки між ними. Діаграма класів – одна з найважливіших структурних діаграм, що ілюструє класи системи, їх атрибути, методи та зв'язки між ними. Аналогічно, діаграми компонентів представляють фізичні компоненти або модулі системи, показуючи, як вони пов'язані. Діаграми об'єктів корисні для зображення екземплярів об'єктів у певний момент часу, тоді як діаграми розгортання окреслюють, як програмні компоненти розгортаються на апаратному забезпеченні. Діаграми пакетів, з іншого боку, групують класи або компоненти в пакети, що відображає організацію системи.

На відміну від цього, діаграми поведінки стосуються динамічної поведінки системи. Ці діаграми зосереджені на ілюструванні того, як система функціонує з часом, з акцентом на взаємодії та процесах. Діаграми варіантів використання є ключовим прикладом, що відображає функціональні вимоги системи з точки зору користувача. Вони показують учасників (або користувачів, або інші системи) та їхню взаємодію з системою. Діаграми послідовностей надають детальне уявлення про взаємодію між об'єктами з

часом, підкреслюючи хронологічний порядок подій. Діаграми активності допомагають моделювати робочий процес у системі, ілюструючи послідовність дій, точок прийняття рішень та одночасних процесів. Діаграми станів відстежують різні стани, в яких може перебувати об'єкт, та переходи між цими станами на основі певних подій [9].

Діаграми взаємодії, ще одна підмножина UML, візуалізують, як об'єкти взаємодіють один з одним. Вони зосереджені на взаємодії об'єктів з часом, показуючи, як повідомлення передаються між ними. Діаграми зв'язку належать до цієї категорії, виділяючи структуру зв'язку між об'єктами.

UML корисний на різних етапах розробки програмного забезпечення, від початкової фази аналізу до остаточного розгортання. Під час фази аналізу діаграми варіантів використання та активності допомагають уточнити функціональні вимоги та потоки процесів. На етапі проєктування діаграми класів, компонентів та послідовностей допомагають у викладанні структури системи та її компонентів. Під час впровадження UML служить орієнтиром для розробників, щоб зрозуміти, як взаємодіють класи та об'єкти. Нарешті, UML відіграє значну роль у документації та тестуванні, надаючи чіткі моделі, які допомагають забезпечити відповідність системи своїм специфікаціям.

Переваги використання UML численні. Він дозволяє чітко візуалізувати складні системи, що робить їх легшими для розуміння. Дотримуючись стандарту, UML гарантує, що всі зацікавлені сторони знаходяться на одній хвилі, що є важливим для ефективної комунікації. Він також підвищує гнучкість, оскільки діаграми UML можна використовувати на різних етапах життєвого циклу розробки програмного забезпечення, будь то планування, проєктування чи обслуговування системи. Крім того, UML пропонує спільну мову, яка зменшує розрив між технічними та нетехнічними членами команди, забезпечуючи безперебійну співпрацю в рамках проєкту.

Що стосується діаграм, UML включає кілька типів, кожен з яких служить певній меті. Діаграми варіантів використання зосереджені на функціональності користувача, тоді як діаграми класів детально описують

структуру системи, показуючи класи та їх зв'язки. Діаграми послідовностей ілюструють, як об'єкти взаємодіють з часом, а діаграми діяльності моделюють потік керування в системі. Діаграми станів зображують можливі стани об'єкта та те, як він переходить між ними [10].

На завершення, UML є важливим інструментом в об'єктно-орієнтованій розробці програмного забезпечення. Він забезпечує чіткий, стандартизований метод представлення як структури, так і поведінки системи, допомагаючи в комунікації, документуванні та проєктуванні системи. У випадку системи управління особистими фінансами, діаграми UML можна ефективно використовувати для проєктування архітектури системи, потоку даних, взаємодії з користувачами та загальної функціональності, забезпечуючи, щоб система була добре структурованою та відповідала потребам користувачів.

2.2 Об’єктне та функціональне моделювання

2.2.1 Діаграма прецедентів. Діаграма варіантів використання є ключовою частиною Уніфікованої мови моделювання (UML), яка використовується для візуального представлення взаємодії між системою та її користувачами або іншими системами. Цей тип діаграми висвітлює функціональні вимоги до системи, надаючи чіткий та простий огляд того, що система повинна робити з точки зору користувача. Це важливий інструмент на ранніх етапах проєктування програмного забезпечення, який допомагає як технічним, так і нетехнічним зацікавленим сторонам зрозуміти очікувану поведінку системи [9].

В основі діаграми варіантів використання лежать актори та варіанти використання. Актори – це сутності, які взаємодіють із системою, це можуть бути люди, інші системи або пристрої. Наприклад, у системі управління особистими фінансами актори можуть включати користувача (особу, яка керує своїми фінансами), адміністратора (відповідального за керування

налаштуваннями системи) або навіть сторонню службу, таку як платіжний шлюз. Варіанти використання представляють конкретні завдання або функції, які система виконує у відповідь на взаємодію актора. Наприклад, варіанти використання в системі управління особистими фінансами можуть включати «Додати витрати», «Переглянути звіт про доходи» або «Створити фінансовий звіт».

Окрім акторів та варіантів використання, діаграма також містить зв'язки, що пов'язують акторів з варіантами використання. Ці зв'язки допомагають визначити характер взаємодії. Асоціація – це базовий зв'язок, який відображається як проста лінія, що з'єднує актора з варіантом використання. Зв'язок включення означає, що один варіант використання завжди є частиною іншого, наприклад, вимагає варіанту використання «Автентифікація користувача» перед доступом до варіанту використання «Переглянути звіт». Зв'язок розширення вказує на те, що варіант використання може додавати необов'язкову або умовну поведінку до іншого. Нарешті, узагальнення представляє одного актора або варіант використання як більш конкретну версію іншого, наприклад, актор «Адміністратор», який має більше привілеїв, ніж загальний «Користувач».

Розроблена діаграма прецедентів використання представлена на рис. 3.

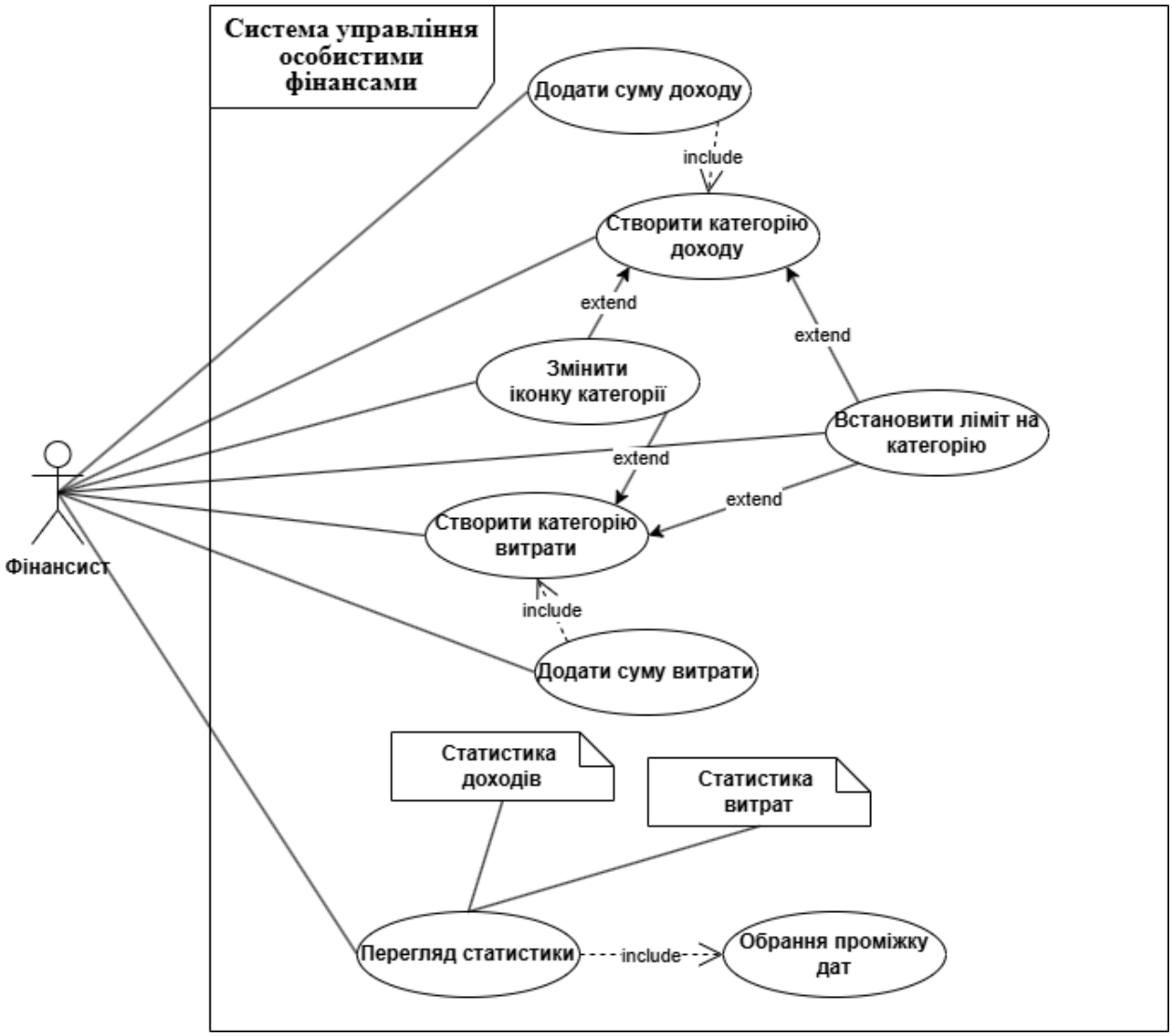

Рис. 3 Діаграма прецедентів

Створена діаграма прецедентів містить акторів:

- “Фінансист”.

Актор «Фінансист» включає такі прецеденти:

- створити категорію доходу;
- додати суму доходу;
- змінити іконку категорії;
- встановити ліміт на категорію;
- створити категорію витрати;
- додати суму витрати;
- перегляд статистики;

- обрання проміжку дат.

Прецеденти певним чином залежать одне від одного.

Розглянемо детальніше вищеописані прецеденти.

Сценарій: Створення категорії доходу

Виконавець: Фінансист

Опис: Фінансист відповідає за керування категоріями в системі, включаючи створення нових категорій доходу, щоб користувачі могли правильно класифікувати свої джерела доходу. Ця функціональність гарантує, що користувачі можуть точно відстежувати та категоризувати свої доходи, допомагаючи їм ефективніше керувати своїми фінансами.

Послідовність дій:

1. фінансист отримує доступ до адміністративної панелі програми;
2. він переходить до розділу «Категорії доходу», де перелічені існуючі категорії доходу та пропонується можливість створити нову категорію;
3. фінансист натискає кнопку «Створити нову категорію»;
4. з’являється форма, яка пропонує фінансисту ввести інформацію про нову категорію доходу;
5. фінансист заповнює необхідні поля та перевіряє деталі;
6. після перевірки інформації фінансист натискає кнопку «Зберегти» або «Надіслати»;
7. система перевіряє введену інформацію, щоб переконатися, що немає порожніх полів або дублікатів назв категорій;
8. якщо дані дійсні, система зберігає нову категорію доходу в базі даних;
9. після успішного створення система відображає підтвердження, яке сповіщає Фінансиста про успішне додавання нової категорії доходу;
10. щойно створена категорія тепер відображається у списку доступних категорій доходу.

Альтернативні потоки:

1. якщо Фінансист вводить недійсні дані (наприклад, залишає назву категорії порожньою або вводить дублікат назви), система відображає повідомлення про помилку із пропозицією виправити проблему;
2. якщо Фінансист не має необхідних дозволів для створення категорій, система заборонить доступ до цієї функції та покаже відповідне повідомлення.

Цей сценарій гарантує, що Фінансист може керувати категоріями доходу в системі, дозволяючи користувачам правильно класифікувати свої доходи та вести організований фінансовий облік.

Розглянемо ще один сценарій використання.

Сценарій: Встановлення ліміту для категорії

Виконавець: Фінансист

Опис: Фінансист відповідає за встановлення лімітів витрат на певні категорії доходів або витрат у системі. Встановлення ліміту допомагає користувачам контролювати свої фінанси, гарантуючи, що вони не перевищують попередньо визначені пороги для різних фінансових категорій. Ця функціональність дозволяє користувачам залишатися в межах своїх фінансових цілей та приймати обґрунтовані рішення щодо витрат.

Потік подій:

1. фінансист переходить до розділу «Ліміти категорій» з адміністративної панелі;
2. система відображає список усіх існуючих категорій, включаючи категорії доходів та витрат;
3. кожна категорія має опцію встановлення або зміни ліміту витрат або доходу;
4. фінансист вибирає конкретну категорію, для якої він хоче встановити ліміт (наприклад, «Продукти харчування»,

«Розваги», «Комунальні послуги»);

5. система відображає сторінку або модальне вікно з деталями цієї категорії, включаючи її поточний ліміт (якщо такий є) та іншу пов’язану інформацію;
6. фінансист вводить бажане лімітуюче значення для категорії у вказане поле (наприклад, «8000 грн. на місяць» для категорії «Продукти»);
7. фінансист може вибирати між різними типами лімітів (наприклад, щоденними, щотижневими, щомісячними) залежно від характеру категорії;
8. можна встановити додатковий поріг сповіщення, коли система повідомлятиме користувача, коли він наближається до ліміту або перевищує його;
9. після введення відповідної інформації про ліміт Фінансист переглядає дані;
10. фінансист натискає кнопку «Зберегти» або «Встановити ліміт», щоб зберегти новий ліміт для категорії;
11. система перевіряє введені дані, переконуючись, що ліміт є додатним числом і що для інших категорій не встановлено конфліктуючих лімітів;
12. після успішного збереження ліміту система відображає Фінансистові повідомлення з підтвердженням, яке вказує на те, що ліміт успішно застосовано до вибраної категорії;
13. оновлена категорія тепер відображає щойно встановлений ліміт, і він буде відстежуватися в системі для подальшого використання;
14. коли користувачі вводять свої витрати або доходи в цій категорії, система порівнює суми з встановленим лімітом;
15. якщо користувач наближається до ліміту або перевищує його, система надсилатиме йому сповіщення (наприклад,

попередження, коли досягнуто 90% ліміту).

Альтернативні потоки:

1. якщо Фінансист вводить недійсний ліміт (наприклад, від'ємне число або нуль), система відображатиме повідомлення про помилку та запропонує Фінансистові ввести дійсне значення;
2. якщо Фінансист не має необхідних дозволів для зміни лімітів категорій, система заборонить доступ до цієї функції та відобразить відповідне повідомлення про помилку.

Цей сценарій гарантує, що Фінансист може встановити відповідні ліміти витрат або доходів для різних категорій, що може значно покращити фінансову обізнаність користувачів та допомогти їм дотримуватися своїх бюджетних обмежень.

2.2.2 Діаграма послідовності. Діаграма послідовності — це важливий тип діаграми взаємодії в уніфікованій мові моделювання (UML), призначений для ілюстрації того, як об'єкти або компоненти взаємодіють один з одним з часом. Ця діаграма надає чітке уявлення про послідовність, у якій повідомлення або події обмінюються між цими сутностями для виконання певного завдання або процесу. Діаграма підкреслює потік керування та даних, де вертикальні лінії представляють об'єкти, а горизонтальні стрілки позначають повідомлення або взаємодії [11].

У контексті системи управління особистими фінансами діаграми послідовності можуть бути надзвичайно корисними для демонстрації того, як користувачі, система та її різні компоненти взаємодіють під час певних операцій, таких як налаштування категорій доходів і витрат, запис транзакцій або застосування фінансових лімітів.

Ключовими елементами діаграми послідовності є актори, об'єкти, повідомлення, лінії життя та смуги активації. Актори, такі як користувачі або зовнішні системи, є учасниками взаємодії. Об'єкти представляють екземпляри класів, які виконують завдання, такі як інтерфейс користувача

або база даних. Стрілки між об'єктами представляють обмінповідомленнями, зазвичай вказуючи на виклик функції або подію, що запускає певну дію. Лінії життя — це вертикальні пунктирні лінії, що представляють активний період часу об'єкта або актора. Смуги активації, що виглядають як тонкі прямокутники на лініях життя, показують час, протягом якого об'єкт активно виконує операцію або завдання.

За повідомленнями на діаграмі послідовності також можуть йти повідомлення-відповіді, які зазвичай представлені пунктирними стрілками, що вказують на повернення керування після завершення дії.

Діаграма послідовності, зображена на рис. 4, включає наступні об'єкти:

- “Фінансист”;
- “Доходи”;
- “Витрати”;
- “Ліміт”.

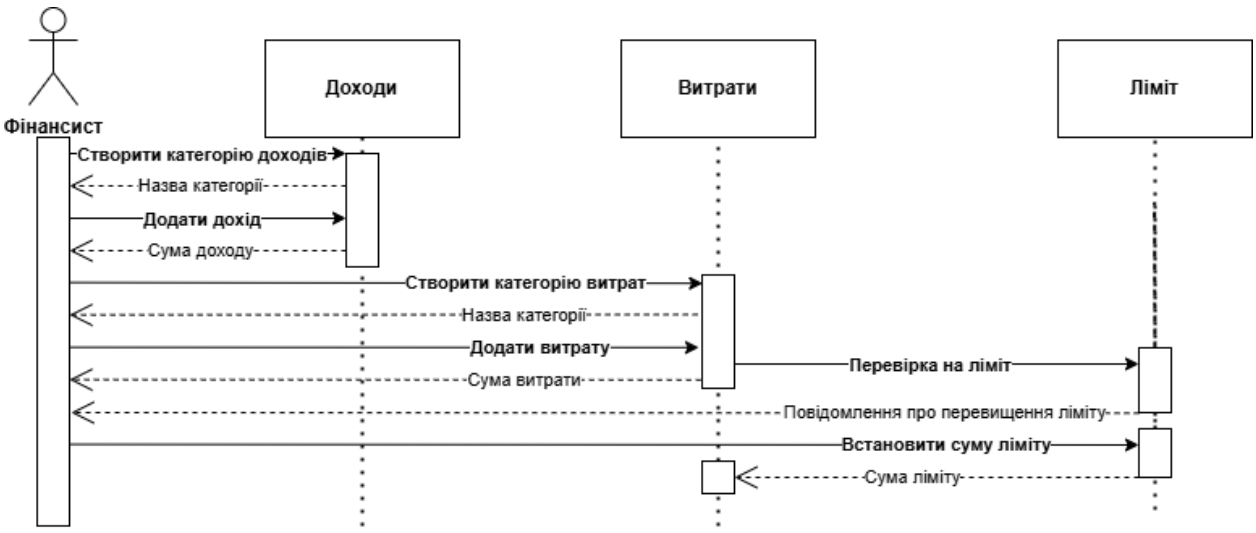

Рис. 4 Діаграма послідовності

Діаграма послідовності ілюструє робочий процес управління фінансами, зосереджуючись на категоризації та моніторингу доходів і витрат. Він починається зі створення фінансовим менеджером категорії для доходу, вказуючи її назву та суму, а потім додає відповідний дохід до цієї категорії. Далі аналогічний процес розгортається для витрат, де визначаються категорії,

а витрати призначаються відповідно. Ці витрати потім перевіряються, щобпереконатися, що вони відповідають заздалегідь визначеним лімітам. Якщо витрати перевищують встановлені межі, спрацьовує сповіщення, щоб висвітлити цю проблему. Для підтримки фінансового контролю та дисципліни також можна проактивно встановлювати ліміти для категорій витрат. Цей процес забезпечує систематичну фінансову організацію та сприяє ефективному моніторингу та управлінню.

2.2.3 Діаграма активності. Діаграма діяльності – це фундаментальний інструмент уніфікованої мови моделювання (UML), який використовується для представлення потоку процесів або робочих процесів у системі. Вона візуально демонструє, як пов'язані дії або дії, показуючи порядок, у якому вони відбуваються, та як керування переходить від одного кроку до іншого. Цей тип діаграми особливо цінний для ілюстрації складних бізнес-процесів, взаємодії користувачів або системних функцій, що включають кілька послідовних або умовних кроків [].

Ключові компоненти діаграми діяльності включають дії, які зображуються у вигляді заокруглених прямокутників та представляють окремі завдання або дії в процесі. Переходи, показані у вигляді стрілок, з'єднують ці дії та вказують, як керування переходить від однієї дії до наступної. Діаграма починається з початкового вузла, представленого зафарбованим чорним колом, що позначає початок процесу. В кінці процесу кінцевий вузол означає висновок, зображений у вигляді зафарбованого чорного кола в кільці.

Вузли рішень, представлені у вигляді ромбів, показують, де процес може розгалужуватися на основі певних умов. Шляхи, що ведуть від цих вузлів, залежать від умов захисту, які зазвичай є логічними виразами, що визначають, яку гілку слідувати. Після прийняття рішень вузли злиття об'єднують різні шляхи назад в один потік.

Для процесів, що включають кілька паралельних завдань, використовуються вузли розгалуження та з'єднання. Вузли розгалуження

розділяють потік на кілька одночасних дій, тоді як вузли з'єднання синхронізують ці дії назад в одну. Крім того, доріжки можна використовувати для візуального розділення дій відповідно до залучених

учасників або компонентів. Це допомагає уточнити, яка сутність, така як користувач чи система, відповідає за кожну частину процесу.

Розроблена діаграма активності представлена на рис. 5

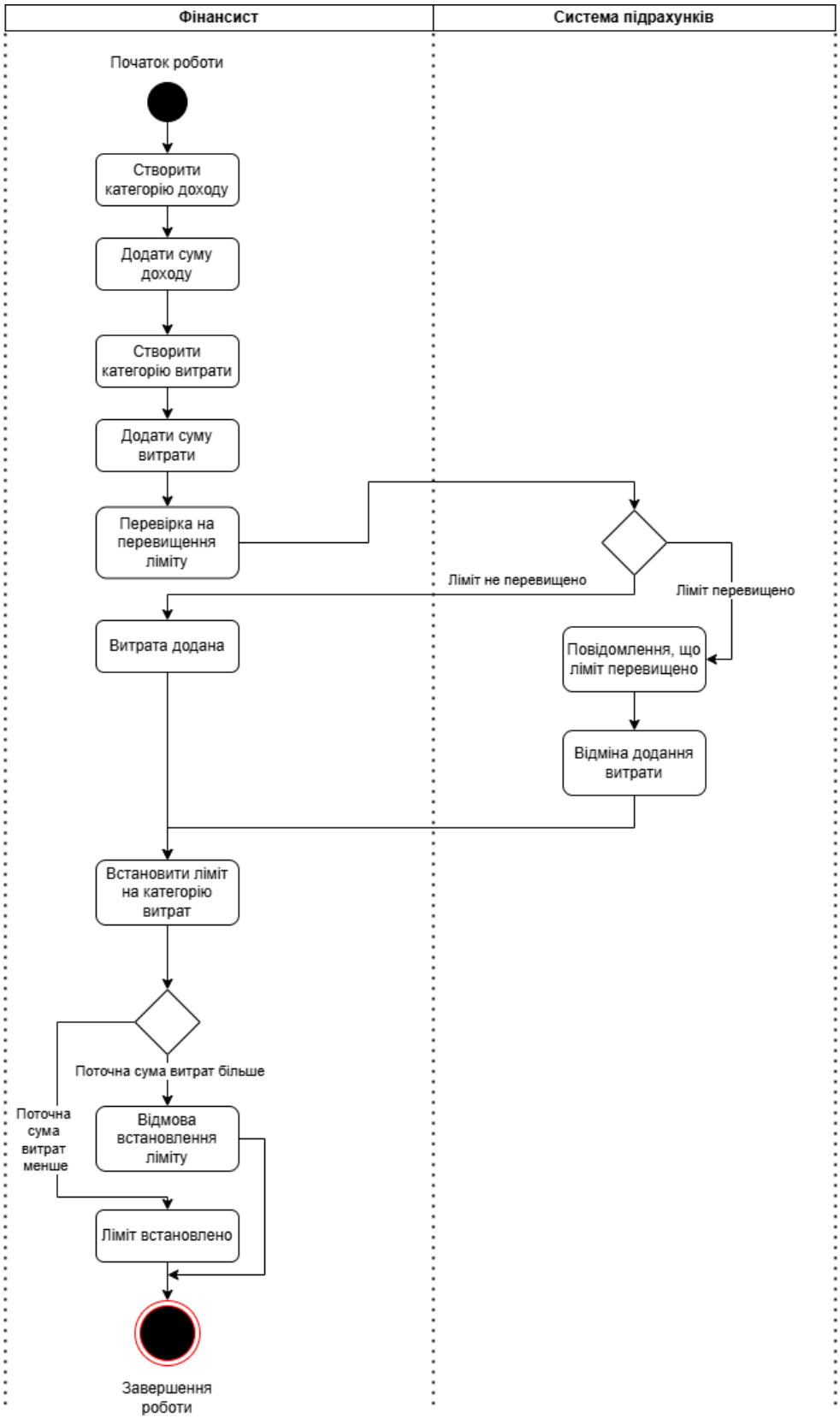

Рис. 5 Діаграма активності

Ця діаграма активності демонструє процес взаємодії фінансиста з системою підрахунків. Початковий крок – початок роботи, де фінансист створює категорії для доходів і витрат. У кожну категорію він додає відповідні суми.

Система підрахунків відіграє ключову роль у перевірці витрат щодо встановлених лімітів. Якщо витрати перевищують межі, система повідомляє про це і не дозволяє додати перевищену суму. Фінансист також може встановлювати ліміти для витрат. У випадку, коли поточні витрати вже перевищують бажаний ліміт, встановлення буде скасовано. В іншому разі ліміт успішно додається.

Завершується процес закінченням роботи фінансиста, після того як усі дії виконано. Така діаграма чітко ілюструє розподіл відповідальностей між користувачем і системою, забезпечуючи контроль і дисципліну у фінансовому управлінні.

2.3 Абстракції предметної області

Абстракції предметної області є ключовим поняттям у проєктуванні та моделюванні систем. Вони слугують спрощеним представленням складних реальних сутностей або процесів, що дозволяє розробникам та зацікавленим сторонам краще розуміти систему та працювати з нею. У контексті системи управління особистими фінансами абстракції допомагають зосередитися на найважливіших аспектах системи, зменшуючи непотрібні деталі та виділяючи основні елементи, необхідні для досягнення цілей системи [12].

Абстракції допомагають визначити основні компоненти та їх взаємодію в системі. Вони створюють більш керовану та зрозумілу основу для розробки, розбиваючи складну систему на менші, більш цілісні одиниці. У цьому випадку абстракції включатимуть такі сутності, як Дохід, Витрати, Категорії, Профілі користувачів та Транзакції, кожна з яких представляє значну частину процесу управління фінансами.

Наприклад, Дохід та Витрати можна абстрагувати як окремі сутності з різними атрибутами, такими як суми, дати, описи та категорії. Абстракція категорій дозволяє користувачам ефективніше організовувати свої фінанси, групуючи пов'язані статті доходів або витрат разом. У свою чергу, транзакції можна абстрагувати як окремі фінансові дії, такі як зареєстровані витрати чи доходи, які містять посилання на певні категорії та суми.

Абстрагуючи профіль користувача, система може відокремити персональні дані користувачів від фінансових операцій, зосереджуючись на таких атрибутах, як уподобання користувача, ролі (наприклад, адміністратор, звичайний користувач) та їхні конкретні фінансові цілі. Це також дозволяє системі застосовувати різні дозволи та обмеження, пропонуючи різні рівні доступу залежно від ролі користувача.

У контексті зберігання та управління даними моделі баз даних виступають як абстракції, що представляють, як дані будуть організовані, збережені та отримані. Структура може включати такі сутності, як користувачі, транзакції, категорії та бюджети в реляційних таблицях, з чітко визначеними їхніми відповідними атрибутами та зв'язками.

Ці абстракції допомагають у проєктуванні системи, спрощуючи базову логіку, забезпечуючи ефективне впровадження, підтримку та масштабування системи. Вони також забезпечують загальне уявлення, яке легше зрозуміти зацікавленим сторонам, гарантуючи виконання бізнес- та функціональних вимог без заглиблення в деталі впровадження. Зрештою, абстракції дозволяють створити більш гнучку, адаптивну та зручну в обслуговуванні архітектуру системи, а також слугують основою для подальших рішень щодо проєктування та впровадження.

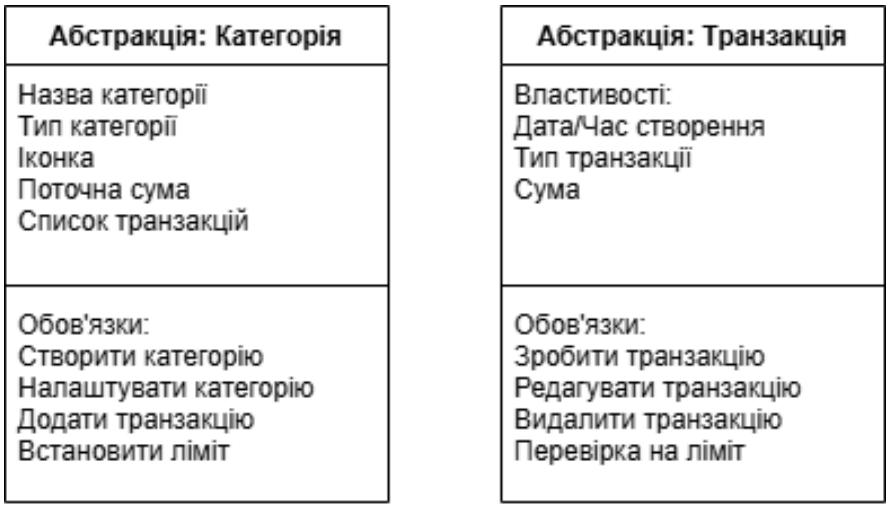

Рис. 6 Абстракції предметної області

Ці абстракції представляють основні елементи предметної області, розподілені між категоріями та транзакціями.

Категорія включає такі властивості, як назва, тип, іконка, поточна сума та список пов'язаних транзакцій. Її обов'язки охоплюють створення та налаштування категорій, додавання транзакцій та встановлення лімітів на витрати.

Транзакція описується характеристиками, як-от дата й час створення, тип транзакції та сума. Вона виконує роль здійснення, редагування, видалення транзакцій і перевірки їх на відповідність заданим лімітам.

Ця структура дозволяє чітко визначити взаємодії між об'єктами системи та забезпечує гнучкість в управлінні фінансовими даними.

2.4 Діаграма класів

Діаграма класів є ключовим компонентом об'єктно-орієнтованого проєктування, зокрема в рамках уніфікованої мови моделювання (UML). Вона забезпечує візуальне представлення статичної структури системи, зосереджуючись на класах та зв'язках між ними. Діаграма використовується для визначення об'єктів системи, їхніх атрибутів, поведінки та взаємодій, що відбуваються між ними [13].

В основі діаграми класів лежать класи, які зображуються у вигляді

прямокутників, розділених на три секції. Верхня секція містить назву класу, середня секція містить атрибути (поля даних класу), а нижня секція містить методи (або функції), які може виконувати клас. Кожен клас служить шаблоном для створення об'єктів, вказуючи, які дані вони зберігають, та операції, які вони можуть виконувати.

Атрибути представляють характеристики класу, такі як дані або властивості, які міститиме об'єкт цього класу. Наприклад, у класі, такому як User, атрибути можуть включати ім'я, електронну пошту та пароль. Методи визначають, які дії може виконувати клас. Це операції, які може виконувати об'єкт, такі як recordIncome(), addExpense() або generateReport() у класі Transaction.

Діаграми класів також ілюструють різні типи зв'язків між класами. Одним з найпоширеніших є асоціація, яка показує, як пов'язані два класи. Користувач може бути пов'язаний з багатьма Транзакціями, що вказує на зв'язок "один до багатьох". Діаграма може вказувати множинність, щоб вказати, скільки екземплярів одного класу пов'язано з іншим, наприклад, один Користувач має кілька Транзакцій.

Ще одним важливим зв'язком у діаграмах класів є успадкування. Воно використовується, коли один клас (підклас) розширює інший клас (суперклас), успадковуючи його властивості та поведінку. Цей зв'язок візуально представлений лінією з порожнистим трикутником, що вказує на суперклас. Існують також більш спеціалізовані зв'язки, такі як агрегація та композиція. Агрегація являє собою зв'язок "ціле-частина", де частина може існувати незалежно, тоді як композиція є сильнішим зв'язком, де частина не може існувати без цілого.

Для цього проєкту було створено спеціальну діаграму класів з використанням об’єктно-орієнтованого підходу (рис. 7).

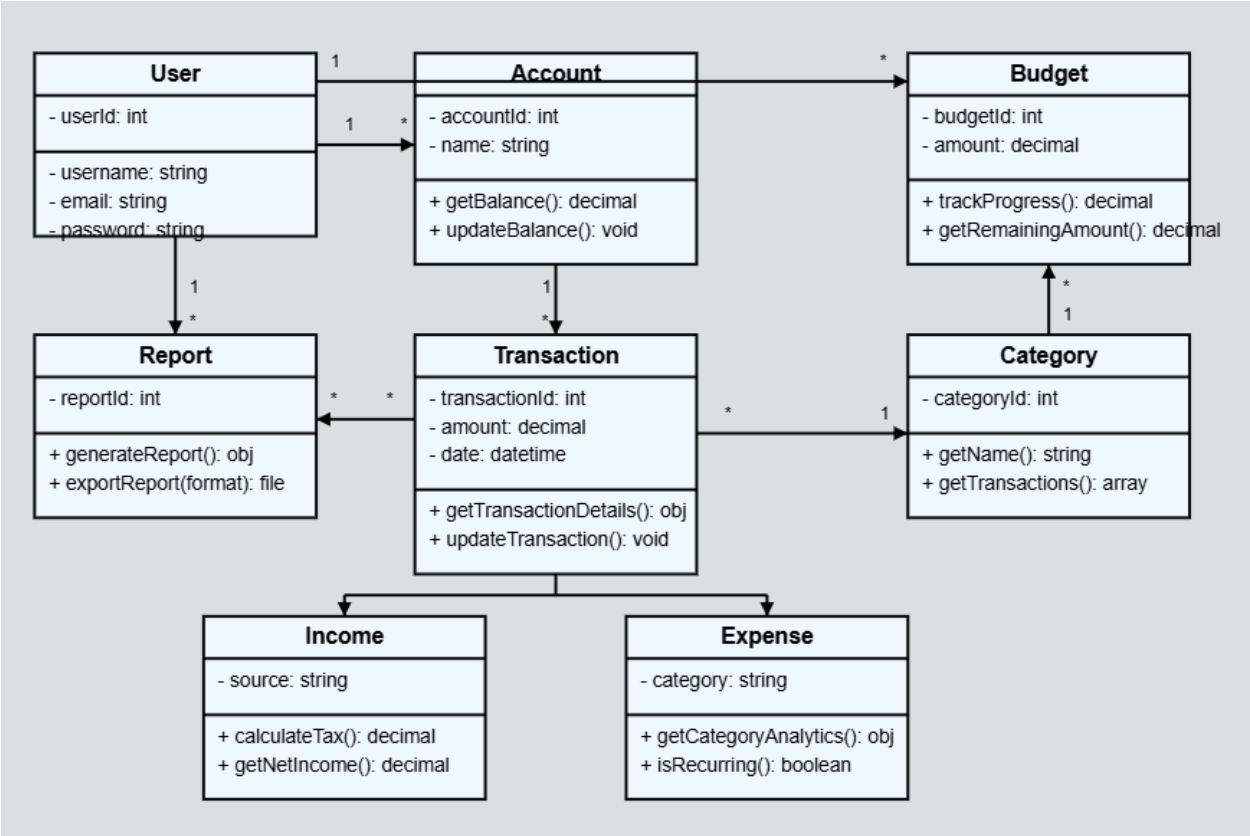

Рис. 7 Діаграма класів

Ця діаграма класів окреслює структурну основу системи фінансового управління, демонструючи зв'язки, атрибути та методи її ключових компонентів. В основі класу User є такі атрибути, як userId, ім'я користувача, електронна пошта та пароль, які підтримують автентифікацію та пов'язують одного користувача з його обліковим записом. Клас Account відповідає за управління балансами, пропонуючи методи їх оновлення та отримання. Він підключається до звітів, транзакцій та бюджетів, забезпечуючи цілісну систему фінансового відстеження.

Клас Budget забезпечує ефективне планування, зберігаючи такі атрибути, як budgetId та amount, а також пропонуючи інструменти для моніторингу прогресу та залишку коштів. Тим часом клас Report спрощує створення та експорт фінансових зведень у різних форматах, допомагаючи користувачам відстежувати їхню фінансову діяльність. Ще одним важливим елементом є клас Transaction, який характеризується transactionId, amount та датою, а також оснащений методами для управління та редагування

транзакцій. Транзакції також пов'язані з класом Category, який організовує їх за певними класифікаціями, надаючи методи доступу до деталей транзакцій.

Спеціалізовані класи розширюють концепцію транзакцій. Клас Income, похідний від класу Transaction, включає додаткові атрибути, такі як source, та пропонує функціональність для розрахунку податків та відстеження чистого доходу. Аналогічно, клас Expense, який також розширює клас Transaction, містить функції для аналізу моделей витрат та категоризації періодичних витрат. Ця взаємопов'язана модель забезпечує безперебійну роботу системи, забезпечуючи структурований, але динамічний підхід до управління фінансами.

3 ПРОЄКТУВАННЯ ПРОГРАМНОЇ СИСТЕМИ

3.1 Логічна модель даних

Логічна модель даних — це абстрактне представлення структури даних, яке зосереджене на організації та зв'язках між даними, не вдаючись у специфіку базової технології чи деталі фізичного зберігання. На відміну від фізичної моделі даних, яка стосується того, як дані фактично зберігаються в базі даних, логічна модель даних представляє бізнес-правила та вимоги до даних системи в більш концептуальній формі. Ця модель служить мостом між бізнес-вимогами та фізичною реалізацією бази даних [14].

Ключові компоненти логічної моделі даних включають сутності, атрибути та зв'язки.

Сутності представляють об'єкти або концепції в системі, які потрібно відстежувати. Наприклад, у системі управління особистими фінансами сутності можуть включати Користувача, Дохід, Витрати, Категорію та Транзакцію. Кожна сутність відповідає основному об'єкту в системі, який зберігає або обробляє дані.

Атрибути — це конкретні деталі або характеристики сутності. Наприклад, сутність Користувач може мати такі атрибути, як ім'я, електронна пошта та пароль. Ці атрибути допомагають визначити властивості, необхідні для виконання сутністю своєї ролі в системі.

Зв'язки визначають, як сутності пов'язані одна з одною. У логічній моделі даних ці зв'язки виражаються через кардинальність, яка показує, скільки екземплярів однієї сутності може бути пов'язано з екземплярами іншої. Наприклад, Користувач може мати багато Транзакцій, тому зв'язок між цими двома сутностями можна описати як «один до багатьох». Аналогічно, Витрати можуть належати до певної Категорії, утворюючи зв'язок «багато до одного».

Логічна модель даних часто використовує нормалізацію для організації даних таким чином, щоб зменшити надлишковість та забезпечити

цілісність даних. Метою нормалізації є розділення великих таблиць на менші, керовані частини та забезпечення узгодженості зв'язків між елементами даних. Цей процес зазвичай включає створення різних таблиць для кожної сутності та визначення їхніх зв'язків на основі бізнес-правил.

Логічна модель даних зазвичай представлена за допомогою діаграм «сутність-зв'язок» (ER), які візуально ілюструють сутності, їх атрибути та зв'язки між ними. Ці діаграми корисні для відображення структури системи та забезпечення того, щоб усі необхідні елементи даних враховувалися та організовувалися таким чином, щоб підтримувати цілі системи [14].

У контексті системи управління особистими фінансами логічна модель даних відображатиме, як користувачі взаємодіють з категоріями доходів і витрат, як відстежуються транзакції та як можуть створюватися звіти. Вона також визначатиме зв'язки між користувачами, категоріями та транзакціями, допомагаючи забезпечити логічну організацію даних відповідно до потреб програми.

Підсумовуючи, логічна модель даних є важливим кроком у процесі проєктування бази даних, оскільки вона забезпечує чітку структуру даних, гарантуючи, що система може ефективно зберігати та керувати необхідною інформацією. Вона забезпечує основу для перетворення бізнес-вимог у технічні специфікації, які пізніше можна реалізувати у фізичній базі даних.

Логічна модель системи представлена на рисунку 8.

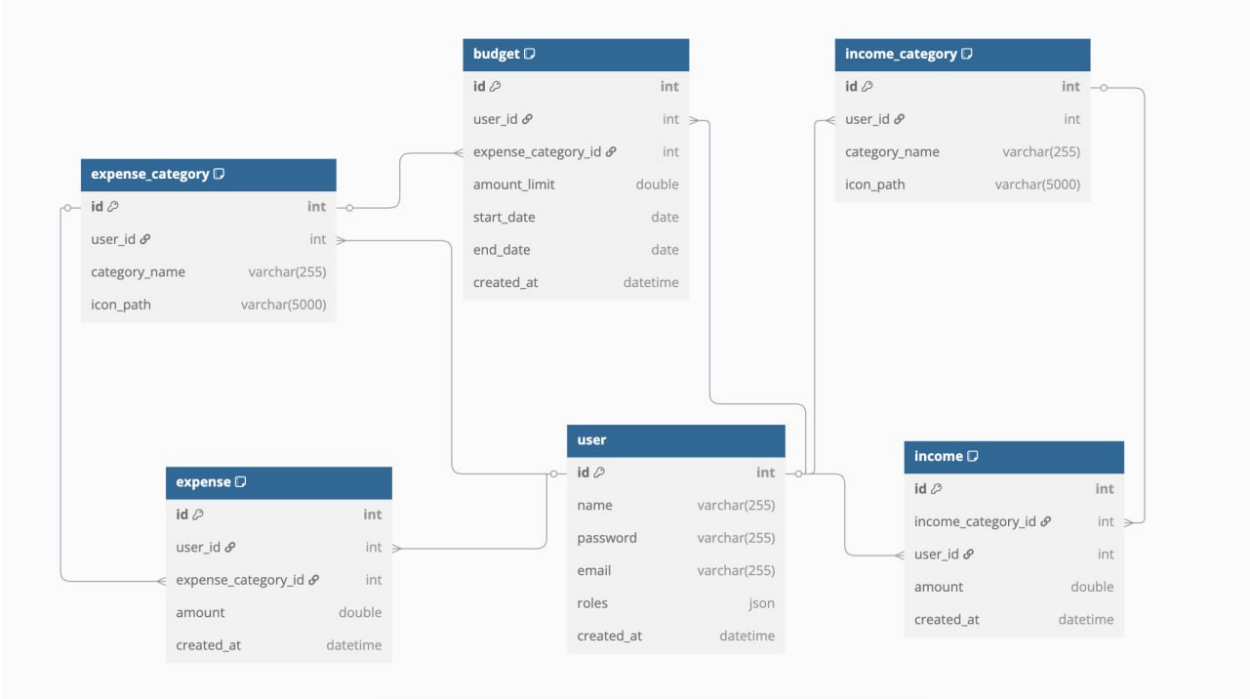

Рис. 8 ER-діаграма «cashflow_db»

Ця ER-діаграма відображає базу даних для управління фінансами, включаючи таблиці, їх атрибути та взаємозв'язки між ними.

Таблиці:

1. expense_category: містить інформацію про категорії витрат із такими атрибутами, як id, user_id, category_name, icon_path;
2. expense: зберігає витрати, описані через id, user_id, expense_category_id, amount, та created_at;
3. budget: відповідає за ліміти витрат, включаючи id, user_id, expense_category_id, amount_limit, start_date, end_date, і created_at;
4. user: представляє користувачів системи із атрибутами id, name, password, email, roles, created_at;
5. income_category: організує категорії доходів і включає id, user_id, category_name, icon_path;
6. income: деталізує доходи через id, income_category_id, user_id, amount, та created_at.

Зв'язки:

1. expense_category зв'язана з таблицею expense через атрибут expense_category_id;
2. budget має зв'язки з таблицею expense_category за допомогою expense_category_id і таблицею user через user_id;
3. expense також зв'язана з таблицею user, використовуючи атрибут user_id;
4. income_category під'єднана до таблиці income через income_category_id;
5. income має зв'язок із таблицею user за допомогою user_id.

Ця структура забезпечує чітке відображення фінансових операцій, їх класифікацію та контроль над лімітами.

3.2 Вибір системи управління базою даних та її реалізація

Під час розробки веб-застосунку для управління особистими фінансами вибір відповідної системи керування базами даних (СКБД) є вирішальним кроком. Вибір СКБД безпосередньо впливає на продуктивність, масштабованість та простоту обслуговування застосунку. У випадку системи управління особистими фінансами, де цілісність даних, безпека та ефективне запитування є важливими, важливо врахувати кілька факторів перед вибором СКБД.

Першим кроком у виборі СКБД є розуміння вимог до системи. Застосунок повинен зберігати та керувати великими обсягами транзакційних даних, включаючи доходи, витрати, категорії та інформацію про користувачів. Він повинен підтримувати можливість легкого отримання та маніпулювання даними на основі різних критеріїв, таких як створення звітів або запити транзакцій за категоріями чи датами. Крім того, система повинна ефективно обробляти складні зв'язки між користувачами, транзакціями та категоріями.

Враховуючи ці вимоги, реляційна система керування базами даних

(РСБД) є найбільш підходящим вибором для цієї системи. РСБД розроблені для обробки структурованих даних з чітко визначеними зв'язками, що робить їх ідеальними для таких застосунків, як управління особистими фінансами, де транзакції та дані користувачів зазвичай зберігаються в таблицях з встановленими зв'язками. Можливість визначати та забезпечувати зв'язки за допомогою зовнішніх ключів, а також підтримка складних запитів гарантує легке масштабування програми в міру додавання нових користувачів та транзакцій [16].

Для впровадження MySQL у системі управління особистими фінансами першим кроком має бути розробка схеми бази даних. Це включає визначення необхідних сутностей (таких як Користувачі, Транзакції, Категорії та записи про доходи/витрати) та зв'язків між ними. Схема має бути розроблена таким чином, щоб мінімізувати надлишковість та забезпечити ефективну роботу з запитами шляхом нормалізації даних, де це необхідно.

Таблиця Користувачі зберігатиме інформацію про користувача, таку як ім'я, електронна пошта, пароль та роль користувача. Таблиця Категорії зберігатиме різні категорії доходів та витрат із посиланнями на таблицю транзакцій. Таблиця Транзакції відстежуватиме дату, суму, тип (дохід чи витрати) та пов'язану категорію для кожної транзакції. Крім того, будуть створені індекси для часто запитуваних полів, таких як ідентифікатор користувача, ідентифікатор категорії та дата транзакції, щоб забезпечити швидкий час пошуку.

Після розробки схеми наступним кроком буде створення бази даних та таблиць за допомогою SQL-скриптів. Надійна підтримка SQL у MySQL дозволяє розробникам визначати обмеження, первинні та зовнішні ключі, а також інші правила цілісності безпосередньо в базі даних. Потім програма взаємодіятиме з MySQL через веб-фреймворк, такий як Symfony, використовуючи інструменти об'єктно-реляційного відображення (ORM), такі як Doctrine, для абстрагування операцій з базою даних та спрощення їх керування.

Крім того, база даних повинна обробляти автентифікацію користувачів та доступ на основі ролей. Конфіденційні дані, такі як паролі, зберігатимуться безпечно з використанням методів шифрування, таких як хешування, для захисту конфіденційності користувачів. Програма також повинна забезпечувати можливість додавання, редагування та видалення транзакцій, категорій та профілів користувачів з відповідною перевіркою та обробкою помилок, щоб запобігти невідповідності даних.

Після запуску системи буде впроваджено регулярне резервне копіювання бази даних, щоб забезпечити відновлення даних у разі системних збоїв. Вбудовані інструменти MySQL для резервного копіювання та відновлення дозволяють легко автоматизувати цей процес. Крім того, інструменти моніторингу продуктивності можна використовувати для відстеження продуктивності запитів та оптимізації індексів і запитів у міру зростання системи.

На завершення, вибір MySQL як системи керування базами даних для системи управління особистими фінансами забезпечує надійну, масштабовану та безпечну основу для управління даними користувачів, транзакціями та фінансовими записами. Завдяки потужним можливостям запитів, сильній підтримці спільноти та легкості інтеграції з PHP та Symfony, MySQL є чудовим вибором для цього проєкту. Завдяки ретельному проєктуванню бази даних та регулярному обслуговуванню система зможе впоратися з вимогами зростаючої бази користувачів, забезпечуючи при цьому цілісність та безпеку даних.

Таким чином було створено базу даних в середовищі MySQL Workbench (Рис. 9).

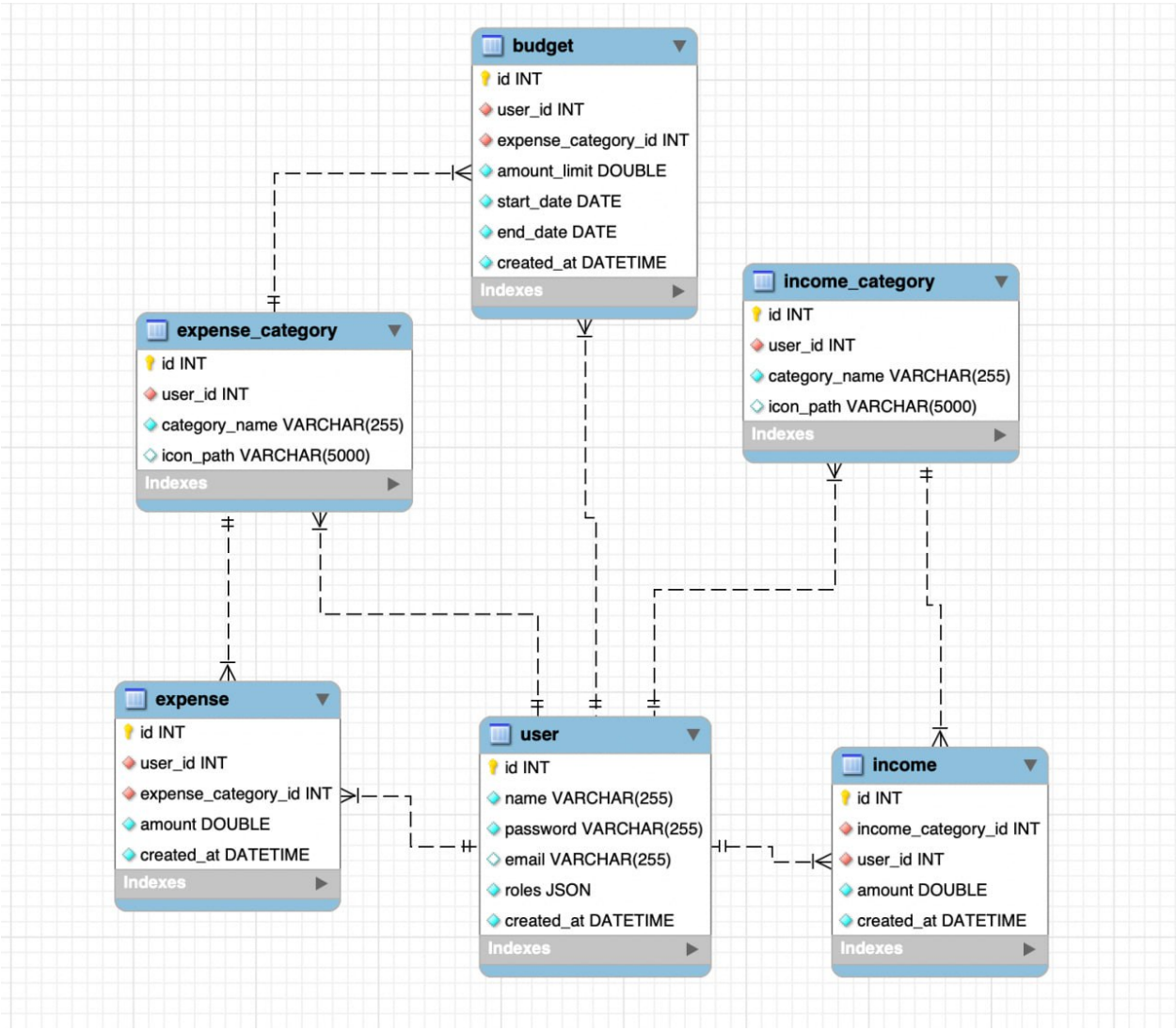

Рис. 9 База даних «cashflow_db»

Ця база даних розроблена для ефективного управління фінансовими операціями, інтегруючи різні взаємопов'язані таблиці, атрибути та зв'язки. В її основі лежить таблиця користувачів, яка зберігає важливі дані, такі як унікальні ідентифікатори, імена користувачів, паролі, адреси електронної пошти, ролі, відформатовані в JSON, та позначки часу створення облікового запису. Ця основа забезпечує безпечний та персоналізований доступ до системи.

Таблиця expense_category організовує категорії витрат, включаючи такі атрибути, як ідентифікатори, асоціації користувачів, назви категорій та шляхи до значків для візуального представлення. Паралельно з цим, таблиця income_category виконує аналогічну функцію для доходів, групуючи їх у класифікації з такими атрибутами, як ідентифікатори, посилання

користувачів, мітки категорій та шляхи до значків.

Таблиця витрат містить детальні записи про витрати, включаючи їхні унікальні ідентифікатори, пов'язані користувачів, пов'язані категорії витрат, суми та позначки часу створення. Доповнюючи це, таблиця доходів документує деталі доходів з такими атрибутами, як ідентифікатори, пов'язані категорії та користувачі, грошові суми та дати створення.

Для забезпечення фінансової дисципліни бюджетна таблиця впроваджує механізм встановлення лімітів витрат у межах певних категорій витрат. Він містить поля для ідентифікаторів, асоціацій користувачів, пов'язаних категорій витрат, визначених лімітів, дат початку та завершення, а також позначок часу, що сприяє структурованій системі контролю.

Зв'язки в базі даних утворюють цілісну структуру. Таблиця expense_category пов'язана з expense через ідентифікатор категорії, що дозволяє категоризувати витрати. Таблиця user слугує основою, пов'язуючи її з кількома іншими таблицями через ідентифікатори користувачів, гарантуючи, що кожен фінансовий запис є специфічним для користувача. Категорії доходів та транзакції взаємопов'язані через ідентифікатори категорій, що дозволяє безперешкодно організовувати дані про доходи. Нарешті, бюджетна таблиця поєднує категорії витрат та користувачів, щоб ефективно забезпечувати обмеження витрат.

Цей дизайн пропонує оптимізований підхід до управління фінансовою інформацією, сприяючи класифікації, аналізу та контролю над доходами та витратами. Він забезпечує гнучкість та надійність, зберігаючи при цьому надійну організаційну основу для фінансових операцій.

3.3 Архітектура програмного забезпечення

Архітектура системи управління особистими фінансами є фундаментальним аспектом забезпечення ефективності, гнучкості та зручності використання застосунку. Вона побудована на PHP-фреймворку

Symfony, який заохочує чіткий розподіл обов'язків завдяки реалізації шаблону Model-View-Controller (MVC). Такий підхід спрощує розробку, тестування та масштабування системи за потреби.

В основі архітектури лежить рівень даних, який визначає, як зберігається та керується така інформація, як користувачі, фінансові транзакції, категорії та ліміти витрат. За допомогою Doctrine ORM ці сутності домену безперешкодно підключаються до бази даних MySQL, забезпечуючи структуроване зберігання та пошук даних. Така конфігурація забезпечує узгодженість в обробці зв'язків між різними сутностями, наприклад, пов'язування певних транзакцій з категоріями, визначеними користувачем.

Рівень представлення відповідає за візуальний інтерфейс, з яким взаємодіють користувачі. Використовуючи шаблонний механізм Twig, Symfony створює динамічні та зрозумілі інтерфейси користувача, які включають фінансові панелі інструментів, форми транзакцій та візуальну аналітику. Цей рівень зосереджений на забезпеченні плавного та інтуїтивно зрозумілого користувацького досвіду, адаптуючи свій вивід на основі введених користувачем даних та стану застосунку.

Рівень додатку, представлений контролерами Symfony, обробляє вхідні запити користувачів та координує відповіді. Ці контролери діють як посередники, виконуючи бізнес-логіку, викликаючи сервіси та надаючи правильні представлення браузеру. Наприклад, коли користувач записує новий витрату, контролер обробляє запит, перевіряє дані, оновлює базу даних та оновлює інтерфейс новим балансом або підсумком.

Для покращення організації коду та зручності обслуговування введено спеціальний рівень сервісів для бізнес-процесів, які виходять за рамки базової обробки даних. Тут обробляються такі завдання, як створення щомісячних звітів, виявлення перевитрат у певних категоріях або надсилання сповіщень про бюджет. Це дозволяє контролерам залишатися легкими та зосередженими на маршрутизації та обробці відповідей.

Безпека є критично важливим питанням у будь-якій фінансовій

програмі. Symfony надає надійні інструменти для управління автентифікацією та контролем доступу, гарантуючи, що лише авторизовані користувачі можуть взаємодіяти з персональними даними. Такі функції, як шифрування паролів, керування сеансами та дозволи на основі ролей, утворюють безпечну основу для системи.

MySQL служить обраною реляційною базою даних завдяки своїй надійності та тісній інтеграції з програмами на базі PHP. Це дозволяє ефективну нормалізацію, використання обмежень зовнішніх ключів та ефективне запитування, що сприяє стабільній та продуктивній роботі серверної частини.

Система розроблена інтерактивною та адаптивною. Завдяки використанню AJAX та асинхронних операцій користувачі можуть отримувати оновлення та відгуки в режимі реального часу без необхідності перезавантажувати сторінки. Це покращує зручність використання та заохочує частішу взаємодію з фінансовими даними.

Модульна природа архітектури також сприяє майбутньому розширенню. Нові функції, такі як підтримка кількох валют, планування регулярних платежів або відстеження особистих фінансових цілей, можуть бути інтегровані без порушення існуючої функціональності. Контейнер для впровадження залежностей та система пакетів Symfony дозволяють легко додавати або змінювати компоненти масштабованим способом.

Система включає інструменти для ведення журналу, відстеження помилок та обробки подій. Вони забезпечують безперебійну роботу та спрощують обслуговування, надаючи інформацію про поведінку та продуктивність системи.

Загалом, архітектурний підхід зосереджений на чіткості, модульності та масштабованості. Він підтримує ефективну розробку та забезпечує безпечне, орієнтоване на користувача середовище для ефективного управління особистими фінансами.

Рис. 10 Багатошарова архітектура

Ця архітектура програмного забезпечення побудована за принципом багаторівневої структури, забезпечуючи чітке розмежування функціональних шарів для полегшення розробки, обслуговування та масштабованості системи.

На презентаційному рівні знаходиться інтерфейс користувача, включаючи веб-інтерфейс (WEB UI) та HTML-сторінки, які відповідають за взаємодію користувача з системою. Цей рівень забезпечує доступність і зрозумілість функціоналу для користувача через зручний дизайн і інтерактивність.

Рівень бізнес-логіки відповідає за основні функції програми, такі як обробка категорій доходів та витрат, управління бюджетами та обробка

транзакцій. Він виконує ключову роль у реалізації правил і поведінки системи, забезпечуючи виконання основних завдань.

Рівень доступу до даних фокусується на управлінні збереженням і доступом до інформації. Він включає управління користувачами, обробку категорій та виконання необхідних розрахунків. Цей шар забезпечує ефективну взаємодію з базою даних і оптимізацію роботи з великим обсягом даних.

На рівні бази даних знаходиться MySQL, яка виконує роль надійного сховища для всієї інформації системи. Цей рівень працює з таблицями, що структурують та упорядковують дані, забезпечуючи їхню цілісність і доступність.

Така структура сприяє модульності, зручності в обслуговуванні й гнучкості системи, дозволяючи легко впроваджувати нові функціональні можливості або масштабувати систему.

3.4 Організаційна структура програмного забезпечення

3.4.1 Діаграма пакетів. У контексті системи управління особистими фінансами, схема пакета надає загальне уявлення про те, як різні частини системи згруповані та організовані. Вона допомагає візуалізувати модульну структуру програми, показуючи залежності та зв'язки між логічними одиницями (пакетами), кожен з яких інкапсулює пов'язані класи або компоненти [18].

Система поділена на кілька основних пакетів, кожен з яких виконує певну функцію в рамках архітектури. Пакет керування користувачами містить усі функції, пов'язані з автентифікацією, реєстрацією, керуванням профілями та ролями користувачів. Цей пакет взаємодіє зі службами безпеки та гарантує, що лише авторизовані користувачі можуть отримувати доступ до своїх фінансових даних та керувати ними.

Далі пакет керування транзакціями обробляє записи доходів та витрат. Він включає логіку для створення, оновлення, видалення та переліку транзакцій. Цей пакет тісно пов'язаний з пакетом керування категоріями, який дозволяє користувачам визначати та змінювати власні категорії доходів та витрат для кращої організації.

Пакет контролю бюджету забезпечує підтримку для встановлення фінансових лімітів для кожної категорії та відстеження витрат у режимі реального часу. Він відповідає за генерацію сповіщень або візуальних індикаторів, коли ліміт наближається або перевищується. Цей пакет спирається як на дані транзакцій, так і на дані категорій для виконання своїх розрахунків та порівнянь.

Ще один важливий пакет – це Analytics and Reporting, який агрегує дані з плином часу та надає користувачам зведення, діаграми та фінансову аналітику. Він використовує дані з пакетів контролю транзакцій та бюджету для створення щомісячних, щотижневих або спеціальних звітів.

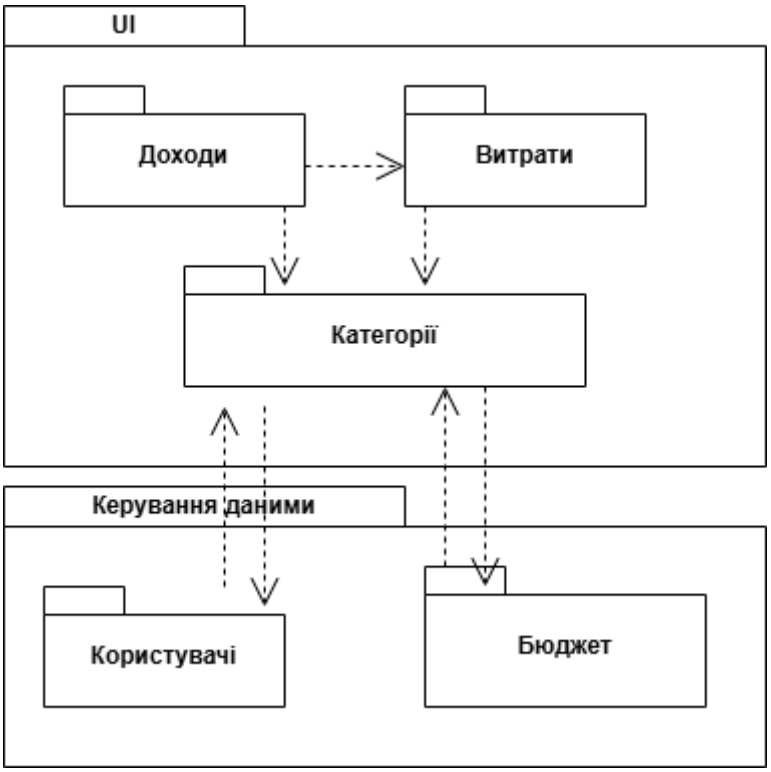

Рис. 11 Діаграма пакетів

Ця діаграма пакетів демонструє структуру системи, розділену на два основні блоки: "UI" (інтерфейс користувача) та "Керування даними".

У блоці "UI" представлені три пакети: "Доходи", "Витрати" та "Категорії". Пакети "Доходи" та "Витрати" мають прямий зв'язок із пакетом "Категорії", що вказує на їхню залежність від категорій. Крім того, між пакетами "Доходи" та "Витрати" існує пунктирний зв'язок, який може символізувати взаємодію або залежність між ними.

Блок "Керування даними" включає два пакети: "Користувачі" та "Бюджет". Пакет "Користувачі" має прямий зв'язок із пакетом "Категорії", що вказує на управління категоріями користувачами. Пакет "Бюджет" також пов'язаний із пакетом "Категорії", а пунктирний зв'язок із пакетом "Витрати" може вказувати на залежність або взаємодію між ними.

Ця діаграма чітко ілюструє взаємозв'язки між компонентами системи, забезпечуючи зрозумілу структуру для організації даних та функціоналу.

3.5 Вибір інструментарію для створення програмного забезпечення

Під час розробки системи управління особистими фінансами вибір відповідних інструментів має вирішальне значення для забезпечення надійності, масштабованості та зручності обслуговування. Обрані технології — PHP, Symfony, MySQL, Doctrine ORM, HTML, CSS та JavaScript — разом утворюють надійний та сучасний стек, що добре підходить для створення веб-додатків.

PHP служить основною мовою програмування застосунку, пропонуючи чудову підтримку серверної логіки, обробки даних та інтеграції з різними інструментами та бібліотеками. Він широко використовується у веб-розробці та надає багатий функціонал для обробки запитів користувачів, керування сеансами та безпечної обробки форм.

Symfony — це фреймворк PHP, обраний для цього проєкту. Він сприяє структурованій, модульній та тестованій кодовій базі, застосовуючи архітектуру MVC (Model-View-Controller). Symfony пропонує велику

екосистему компонентів повторного використання та вбудованих інструментів, таких як маршрутизація, обробка форм, валідація та безпека, що значно пришвидшує розробку та зменшує ймовірність помилок.

Для обробки зберігання та збереження даних MySQL використовується як система управління реляційними базами даних. Він відомий своєю продуктивністю, надійністю та підтримкою складних запитів, що робить його ідеальним для керування структурованими фінансовими даними, такими як транзакції, категорії та профілі користувачів.

Doctrine ORM (Об'єктно-реляційне відображення) інтегровано з Symfony для спрощення взаємодії між об'єктно-орієнтованим кодом PHP та реляційною базою даних MySQL. Це дозволяє розробникам працювати із записами бази даних як із об'єктами PHP, абстрагуючи низькорівневі SQL-запити та зменшуючи складність розробки.

На стороні клієнта HTML забезпечує структуру веб-сторінок, тоді як CSS відповідає за стилізацію та макет. Ці технології гарантують, що інтерфейс користувача є не лише функціональним, але й візуально привабливим та зручним для користувача.

JavaScript використовується для покращення інтерактивності та швидкості реагування. Він дозволяє використовувати такі функції, як перевірка форм у режимі реального часу, динамічне оновлення контенту, модальні вікна та інтерактивні діаграми для фінансового аналізу без необхідності перезавантаження сторінки.

Разом ці інструменти утворюють цілісне середовище, яке підтримує всі етапи розробки — від логіки бекенду та управління даними до презентації фронтенду та взаємодії з користувачем. Їхнє широке використання, обширна документація та підтримка спільноти роблять їх ідеальними для створення веб-додатку для особистих фінансів, який є ефективним, безпечним та простим в обслуговуванні.

4 ВПРОВАДЖЕННЯ ТА ЕКСПЛУАТАЦІЯ СИСТЕМИ

4.1 Вимоги до апаратного та програмного забезпечення

Для забезпечення успішної розробки та розгортання системи управління особистими фінансами важливо чітко визначити вимоги як до апаратного, так і до програмного забезпечення. Ці специфікації підтримують ефективну роботу програми, підвищують продуктивність та забезпечують масштабованість для майбутнього використання.

Що стосується програмного забезпечення, серверне середовище повинно працювати на сучасній та стабільній операційній системі, такій як Ubuntu, Windows 10 або macOS. Розробка бекенду спирається на PHP (версії 8.1 або вище), а фреймворк Symfony забезпечує структуру та організацію програми. Для управління базами даних рекомендується MySQL завдяки своїй надійності та широкій підтримці. Doctrine ORM буде використовуватися для оптимізації взаємодії між програмою та базою даних. Для управління залежностями Composer буде виконувати роль менеджера пакетів. Інструменти розробки, такі як Symfony CLI, PHPStorm або Visual Studio Code, а також MySQL Workbench або DBeaver, допоможуть у кодуванні та управлінні базами даних. Git буде займатися контролем версій, тоді як додаткові бібліотеки, такі як Chart.js, можуть бути використані для візуалізації даних, а фреймворки CSS, такі як Bootstrap або Tailwind CSS, покращать інтерфейс користувача. На фронтенді HTML5, CSS3 та сучасний JavaScript (ES6) забезпечують кросбраузерну сумісність, а додаткові фронтенд-бібліотеки, такі як Vue.js або React, підвищують інтерактивність.

Вимоги до апаратного забезпечення для розробників включають комп'ютер, оснащений щонайменше чотириядерним процесором (наприклад, Intel Core i5 або AMD Ryzen 5), 8 ГБ оперативної пам'яті (бажано 16 ГБ) та

твердотільний накопичувач (SSD) з достатньою ємністю. Для ефективної розробки та тестування також рекомендується монітор Full HD та стабільне підключення до Інтернету.

Для розгортання сервера вирішальне значення має надійне середовище хостингу. Хмарний або локальний сервер повинен мати щонайменше два віртуальні процесори, від 4 до 8 ГБ оперативної пам'яті та швидке SSD-сховище для розміщення даних користувачів та файлів програм. Мережева інфраструктура повинна забезпечувати високу доступність та достатню пропускну здатність для обробки одночасних запитів без затримок.

Кінцеві користувачі можуть отримати доступ до системи за допомогою будь-якого сучасного пристрою, такого як персональний комп'ютер, смартфон або планшет, з сучасним веб-браузером, таким як Chrome, Firefox, Safari або Edge. Базових апаратних можливостей та доступу до Інтернету достатньо для безперебійної взаємодії з програмою.

Ці вимоги закладають основу для стабільної, зручної у використанні та масштабованої веб-програми, здатної допомагати людям ефективно відстежувати та керувати своїми фінансами.

Також було зроблено діаграму розгортання для візуального огляду деплою системи (Рис. 12).

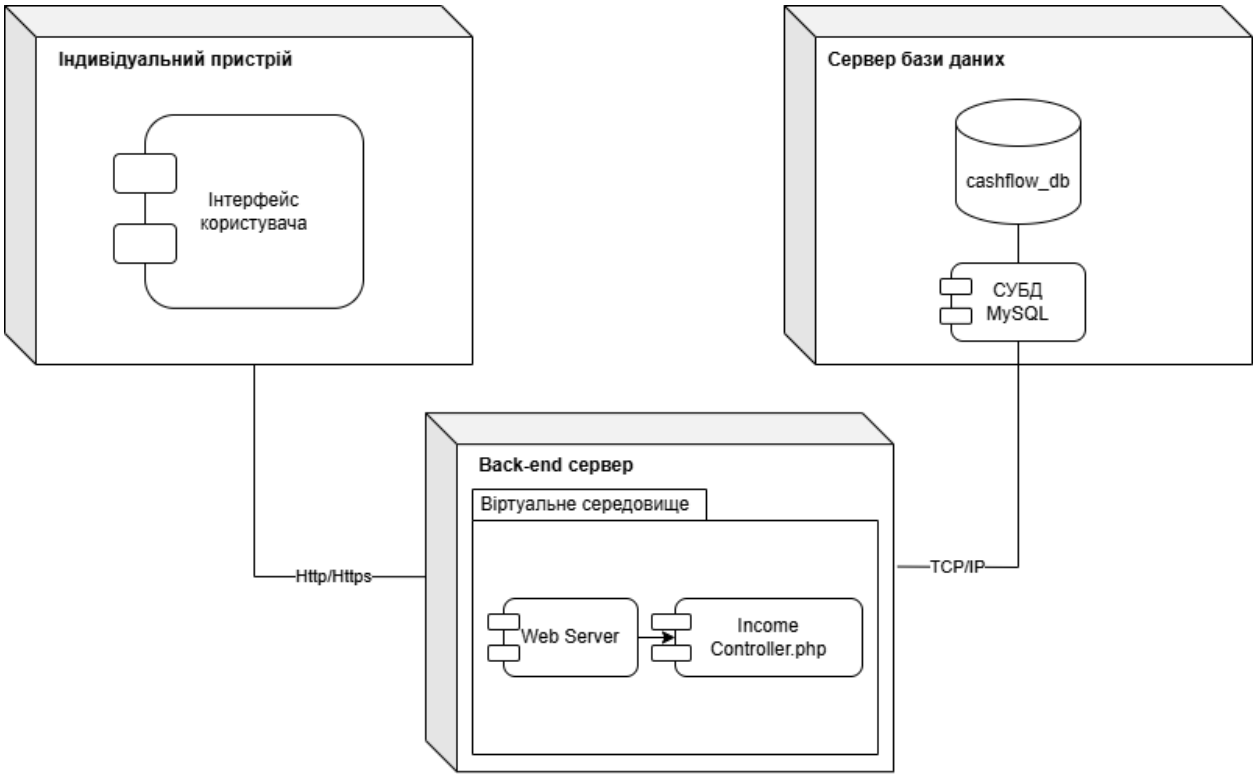

Рис. 12 Діаграма розгортання

4.2 Тестування системи

Запустивши систему, потрапляємо на форму авторизації, тут нам треба ввести логін та пароль користувача (рис. 13).

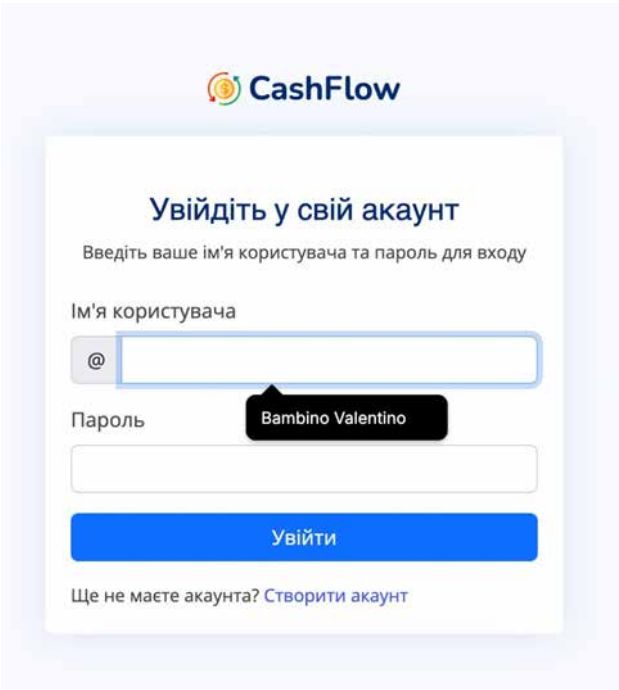

Рис. 13 Форма авторизації

Спочатку нас зустрічає основна сторінка. Тут показуються всі наші дані про фінанси у вигляді категорій доходів, категорій витрат, які суми на що витрачені, скільки зараз є грошей у володінні, який дохід та витрати у поточному місяці (Рис. 14).

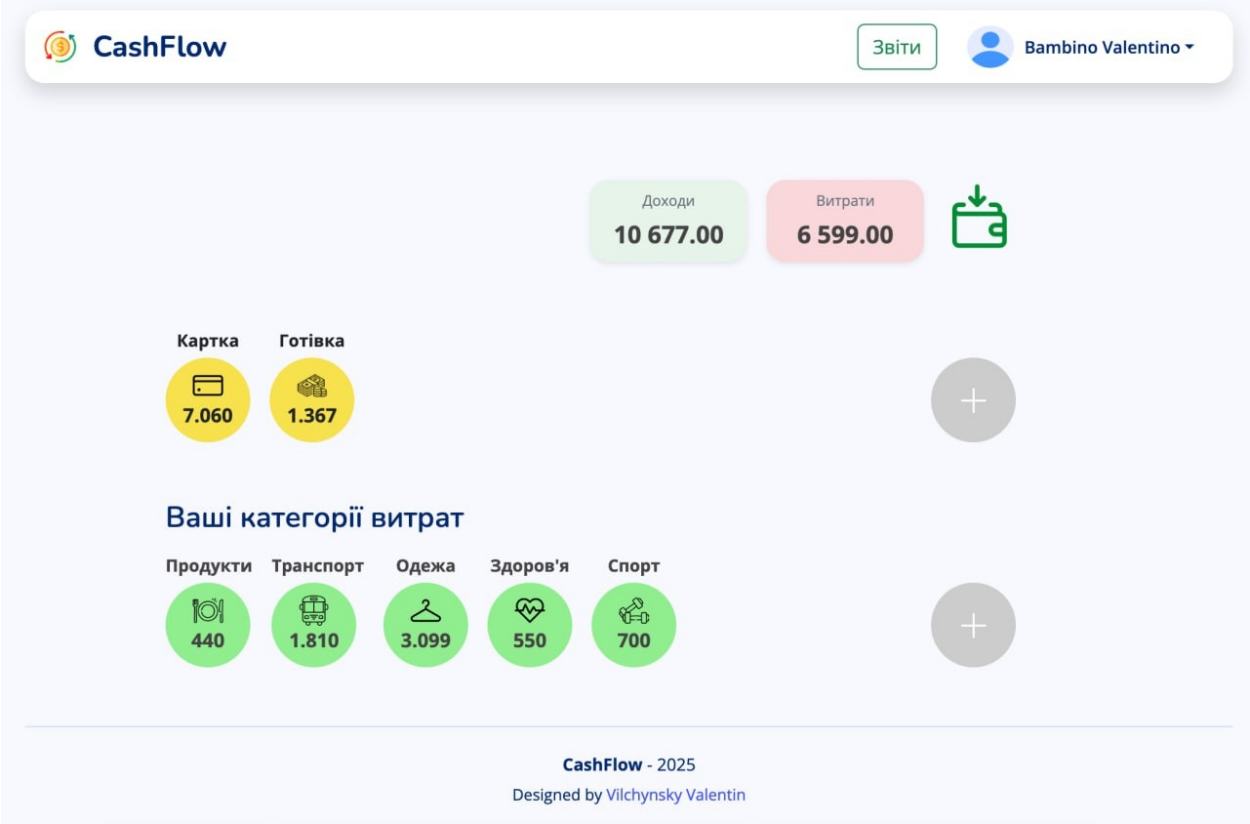

Рис. 14 Головна сторінка

Спробуємо зараз створити нову категорію доходів, для цього нам потрібно натиснути на сірий плюс з правого краю (Рис. 15) і у нас відкриється форма для заповнення даних про категорію доходу (Рис. 16).

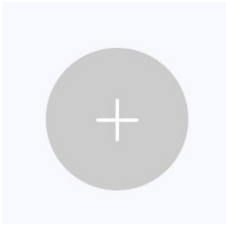

Рис. 15 Елемент створення

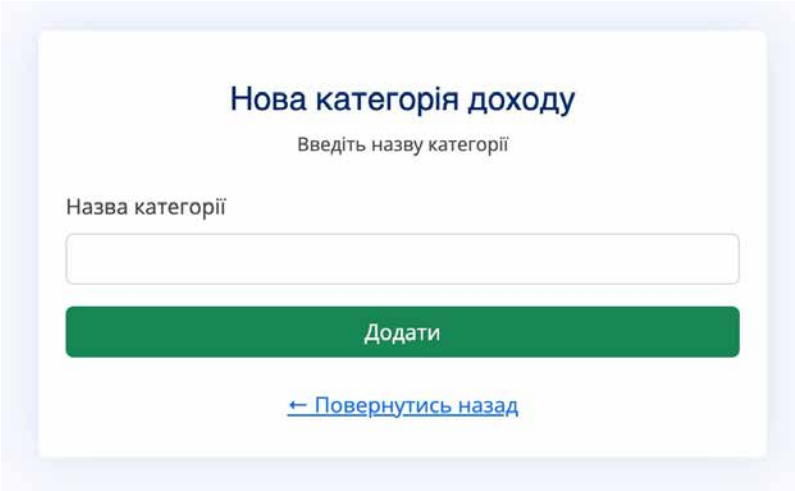

Рис. 16 Форма створення нової категорії

Тепер ми можемо додати певну суму до нашої категорії доходу, для цього нам потрібно натиснути на іконку зеленого гаманця по правому краю (Рис. 17), яка відкриємо нам форму внесення суми до обраної нами категорії (Рис. 18).

Рис. 17 Елемент додання доходу

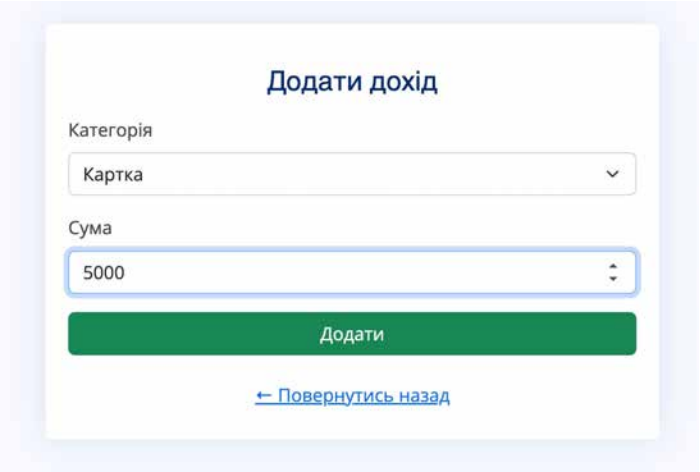

Рис. 18 Форма додання нового доходу

Після додавання суми до категорії доходу, у нас оновиться число в даній категорії (Рис. 19).

Рис. 19 Наша створена категорія доходу

Якщо натиснути на категорію доходу, у нас відкриється її налаштування, тут ми можемо змінити назву або встановити іконку на наш вибір із запропонованих (Рис. 20).

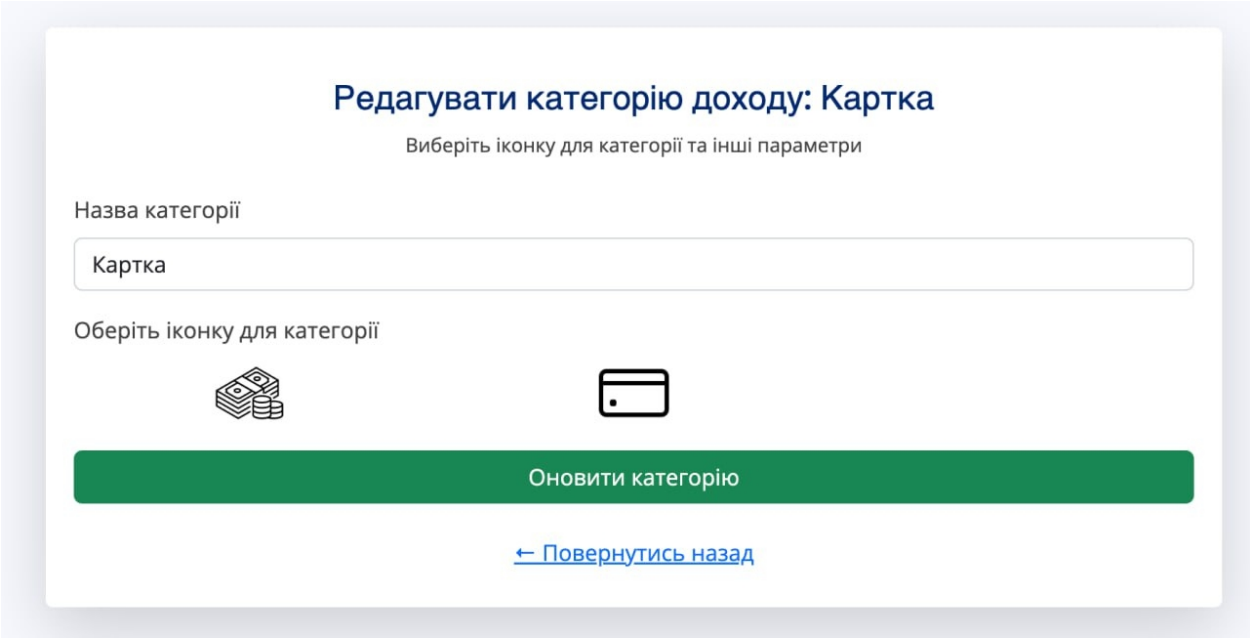

Рис. 20 Форма редагування категорії доходу

Якщо натиснути на категорію витрати, у нас відкриється її сторінка де ми зможемо вказати суму витрати яку ми виробили, можемо також встановити ліміт для даної категорії вказавши період і саму суму ліміту. І так само є налаштування іконок для категорій, щоб візуально швидше знаходити потрібні нам категорії (Рис. 21)

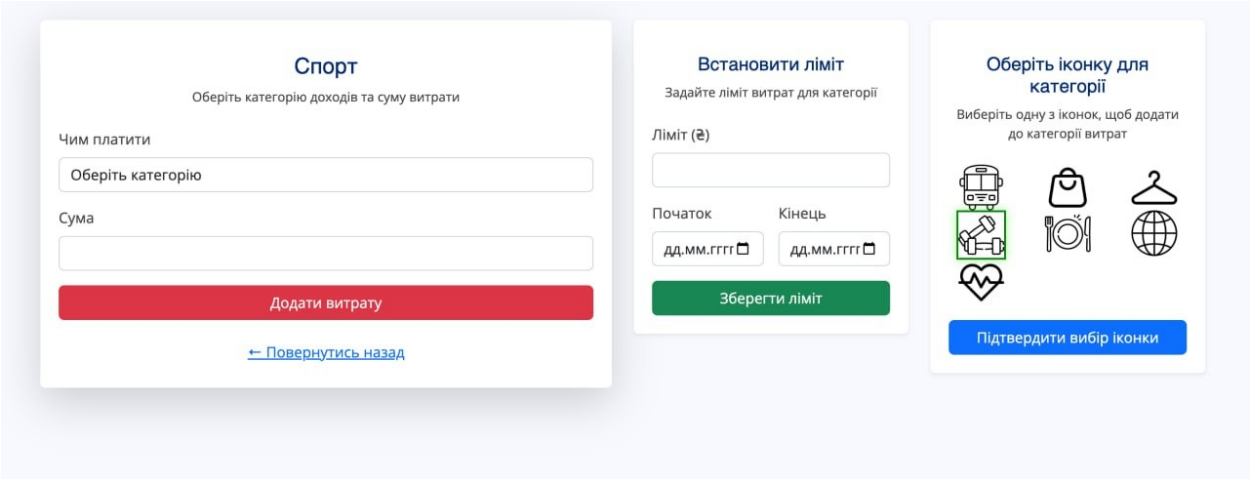

Рис. 21 Форма категорії витрат

Спробуємо заздалегідь встановити ліміт для категорії, наприклад 1000 гривень, і спробувати ввести суму витрати >1000, то ми отримаємо повідомлення про те, що дана витрата перевищує ліміт і в результаті транзакція не буде записана (Рис. 22).

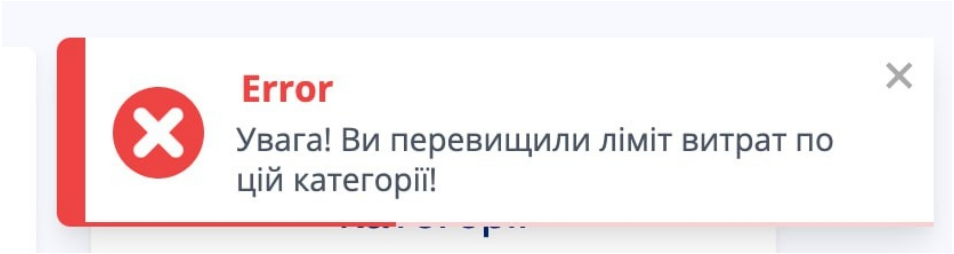

Рис. 22 Повідомлення про перевищення ліміту

І насамкінець, у цій програмі представлений функціонал звітності, нам потрібно перейти на сторінку звітів і тут вибрати діапазон дат, що цікавить нас, за яким буде проводиться підрахунок статистики. Тут у нас показується статистика з доходів і витрат для конкретного користувача (Рис. 23).

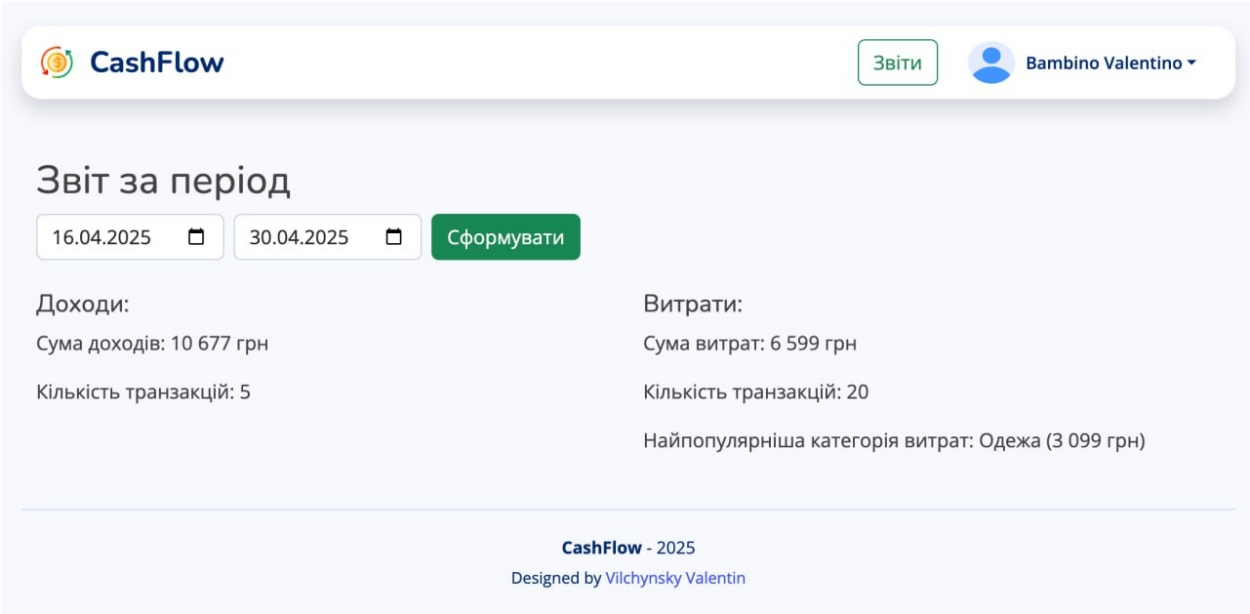

Рис. 23 Сторінка “Звіти”

ВИСНОВКИ

Під час розробки системи управління особистими фінансами було впроваджено повноцінний веб-додаток для задоволення суттєвої потреби в точному та доступному відстеженні доходів і витрат. Ця система дозволяє користувачам повністю контролювати свої особисті фінанси, пропонуючи можливість створювати власні категорії як для доходів, так і для витрат, встановлювати фінансові ліміти та контролювати поведінку витрат за допомогою інтуїтивно зрозумілих візуалізацій та детальної статистики.

Робота розпочалася з вивчення предметної області та формулювання проблеми дослідження. Існуючі рішення були переглянуті, щоб виявити прогалини та можливості для вдосконалення програмного забезпечення для управління особистими фінансами. Це послужило основою для наступних етапів роботи.

Було створено логічну модель даних, що представляє вимоги до інформації та зв’язки в програмному забезпеченні управління особистими фінансами. Ця модель лягла в основу проєктування програмної системи. Розроблено фізичну модель даних, що забезпечує ефективне зберігання та пошук даних за допомогою вибраної системи керування базами даних.

Упродовж усього проєкту проєктування та розробка дотримувалися структурованого підходу. На аналітичному етапі було досліджено предметну область, переглянуто існуючі рішення та визначено основні вимоги – як функціональні, так і нефункціональні. Архітектура проєкту базувалася на сучасних та надійних технологіях: PHP з фреймворком Symfony для бекенд-логіки, MySQL як система управління базами даних, Doctrine ORM для взаємодії з базою даних та HTML/CSS/JavaScript для інтерактивного та адаптивного інтерфейсу користувача.

Було надано візуальне представлення та детальний опис функціональності додатку, що ілюструють, як система підтримує такі операції,

як управління категоріями, встановлення лімітів витрат, облік доходів і витрат та створення звітів. Ці функції інтегровані таким чином, що забезпечує простоту використання, водночас надаючи користувачам надійну фінансову аналітику. Реалізація підтримує інтуїтивно зрозумілу навігацію та ефективний доступ до всіх ключових функцій, що сприяє дизайну, орієнтованому на користувача.

На завершення, розроблена система управління особистими фінансами відповідає початковим цілям роботи. Вона дозволяє користувачам ефективно організовувати свої фінансові дані, приймати обґрунтовані рішення та розвивати кращі звички управління грошима. Гнучкість системи та її модульна архітектура також надають можливості для подальшого розширення, такого як додавання багатокористувацької підтримки, інтеграція із зовнішніми фінансовими службами або розробка мобільних додатків. Ця робота не лише демонструє практичне застосування набутих знань та навичок, але й надає цінний інструмент із реальним застосуванням.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Kapoor, J.R., Dlabay, L.R., Hughes, R.J. Особисті фінанси. — McGraw-Hill Education, 2020.
2. Garman, E.T., Forgue, R.E. Особисті фінанси. — Cengage Learning, 2018.
3. Lusardi, A., Mitchell, O.S. Економічне значення фінансової грамотності: теорія та докази. — Journal of Economic Literature, 2014.
4. OECD. Рекомендація щодо фінансової грамотності. —— Режим доступу: https://www.oecd.org/finance/financial-education/
5. National Endowment for Financial Education. Основи особистих фінансів. —— Режим доступу: https://www.nefe.org/
6. Hilgert, M.A., Hogarth, J.M., Beverly, S.G. Управління фінансами домогосподарств: зв'язок між знаннями та поведінкою. — Бюлетень Федеральної резервної системи, 2003.
7. Комісія з фінансової грамотності та освіти США. Звіт Комісії з фінансової грамотності та освіти.– Режим доступу: https://home.treasury.gov/
8. Гітман, Л.Дж., Джоенк, М.Д. Особисте фінансове планування. — Cengage Learning, 2013.
9. Агентство Канади з питань фінансового захисту прав споживачів. Управління вашими грошима.– Режим доступу: https://www.canada.ca/en/financial-consumer-agency.html
10. Х'юстон, С.Дж. Вимірювання фінансової грамотності. — Журнал у справах споживачів, 2010.
11. CoinKeeper — Режим доступу: https://about.coinkeeper.me
12. 1Money – Режим доступу: https://1money.uptodown.com/android
13. Посібник з PHP — Режим доступу: https://www.php.net/manual/en/
14. Symfony Framework — Режим доступу: https://symfony.com/
15. Документація MySQL — Режим доступу: https://dev.mysql.com/doc/

16. Doctrine ORM — Режим доступу: https://www.doctrine-project.org/projects/orm.html
17. HTML Living Standard — Режим доступу: https://html.spec.whatwg.org/
18. CSS: Каскадні таблиці стилів — Режим доступу: https://developer.mozilla.org/en-US/docs/Web/CSS
19. Посібник з JavaScript– Режим доступу: https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide
20. PlantUML — Режим доступу: https://plantuml.com/
21. Інструмент для створення діаграм yUML — Режим доступу: https://yuml.me/
22. Інструмент для створення діаграм Draw.io — Режим доступу: https://app.diagrams.net/
23. ПРОЄКТУВАННЯ ER-ДІАГРАМИ — Режим доступу: https://nationalteam.worldskills.ua/skills/proektirovanie-er-diagrammy/
24. Діаграма варіантів використання (UseCase diagram) — Режим доступу: https://flexberry.github.io/ua/fd_use-case-diagram.html
25. ПРОЄКТУВАННЯ USE CASE ДІАГРАМИ. ВИЗНАЧЕННЯ ФУНКЦІОНАЛЬНИХ МОЖЛИВОСТЕЙ СИСТЕМИ — Режим доступу: https://nationalteam.worldskills.ua/skills/proektirovanie-use-case-diagrammy-opredelenie-funktsionalnykh-vozmozhnostey-sistemy/
26. What is Sequence Diagram? — Режим доступу: https://www.visual-paradigm.com/guide/uml-unified-modeling-language/what-is-sequence-diagram/
27. UML - Activity Diagrams – Tutorialspoint — Режим доступу: https://www.tutorialspoint.com/uml/uml_activity_diagram.htm
28. Побудова діаграми класів — Режим доступу: https://flexberry.github.io/ua/gpg_class-diagram.html
29. UML-діаграми класів — Режим доступу: https://prog-cpp.ua/uml-classes/

30. Entity Relationship Diagram - Data Modeling — Режим доступу: https://www.visualparadigm.com/VPGallery/datamodeling/EntityRelationshipDiagram.html
31. UML 2 Tutorial - Package Diagram - Sparx Systems — Режим доступу: https://sparxsystems.com/resources/tutorials/uml2/package-diagram.html
32. Документація Bootstrap (офіційний сайт) — Режим доступу: www.getbootstrap.com/documentation.

ДОДАТОК А

Фрагменти програмного коду. Функція створення категорії

```
#[Route('/income/category/add', name: 'add_income_category')]
    public function addIncomeCategory(Request $request, EntityManagerInterface
$entityManager): Response
    {
        if ($request->isMethod('POST')) {
            $categoryName = $request->request->get('name');
            $user = $this->getUser();

            $category = new IncomeCategory();
            $category->setCategoryName($categoryName);
            $category->setUser($user);

            $entityManager->persist($category);
            $entityManager->flush();

            flash()->success('Ви успішно створили категорію доходів');
            return $this->redirectToRoute('app_home');
```

```
        }

        return $this->render('income/add_income_category.html.twig');
    }

    #[Route('/expense/category/add', name: 'add_expense_category')]
    public function addExpenseCategory(Request $request, EntityManagerInterface
$entityManager): Response
2 days ago

created authorization
    {
yesterday

created money functions
        if ($request->isMethod('POST')) {
            $categoryName = $request->request->get('name');
            $user = $this->getUser();

            $category = new ExpenseCategory();
            $category->setCategoryName($categoryName);
            $category->setUser($user);

            $entityManager->persist($category);
            $entityManager->flush();

            flash()->success('Ви успішно створили категорію витрат');
            return $this->redirectToRoute('app_home');
        }

        return $this->render('expense/add_expense_category.html.twig');
    }

#[Route('/income/add', name: 'add_income')]
    public function addIncome(
        Request $request,
        EntityManagerInterface $entityManager,
        IncomeCategoryRepository $incomeCategoryRepository,
        Security $security
    ): Response {
        $user = $security->getUser();
        $categories = $incomeCategoryRepository->findBy(['user' => $user]);

        if ($request->isMethod('POST')) {
            $amount = (float)$request->request->get('amount');
            $categoryId = (int)$request->request->get('category');
            $category = $incomeCategoryRepository->find($categoryId);

            if ($category && $category->getUser() === $user) {
                $income = new Income();
                $income->setAmount($amount);
                $income->setIncomeCategory($category);
```

```
            $income->setUser($user);
            $income->setCreatedAt(new \DateTimeImmutable());

            $entityManager->persist($income);
            $entityManager->flush();

            flash()->success('Новий дохід успішно додано');
            return $this->redirectToRoute('app_home');
        }
    }

    return $this->render('income/add_income.html.twig', [
        'categories' => $categories
    ]);
}
```

ДОДАТОК Б

Фрагменти програмного коду. Підрахунки власних фінансів

```
<?php

namespace App\Repository;

use App\Entity\Income;
yesterday

created money functions
use App\Entity\IncomeCategory;
use App\Entity\User;
use Doctrine\Bundle\DoctrineBundle\Repository\ServiceEntityRepository;
use Doctrine\Persistence\ManagerRegistry;

/**
 * @extends ServiceEntityRepository<Income>
 */
class IncomeRepository extends ServiceEntityRepository
```

```
{
    public function __construct(ManagerRegistry $registry)
    {
        parent::__construct($registry, Income::class);
    }

    public function getTotalIncomeByUser(User $user): float
    {
        return $this->createQueryBuilder('i')
            ->select('SUM(i.amount)')
            ->andWhere('i.user = :user')
            ->setParameter('user', $user)
            ->getQuery()
            ->getSingleScalarResult() ?: 0;
    }

    public function getIncomeSumByCategory(User $user): array
    {
        $result = $this->createQueryBuilder('i')
            ->select('IDENTITY(i.incomeCategory) AS categoryId, SUM(i.amount) AS total')
            ->andWhere('i.user = :user')
            ->setParameter('user', $user)
            ->groupBy('i.incomeCategory')
            ->getQuery()
            ->getResult();

        return array_column($result, 'total', 'categoryId');
    }

    public function getIncomeSum(User $user, IncomeCategory $incomeCategory): float
    {
        return $this->createQueryBuilder('i')
            ->select('SUM(i.amount)')
            ->where('i.incomeCategory = :category')
            ->andWhere('i.user = :user')
            ->setParameter('category', $incomeCategory)
            ->setParameter('user', $user)
            ->getQuery()
            ->getSingleScalarResult() ?: 0;
    }

    public function getIncomeSumByDateRange(User $user, \DateTimeImmutable $startDate,
\DateTimeImmutable $endDate): float
    {
        $result = $this->createQueryBuilder('i')
            ->select('SUM(i.amount) AS total')
            ->andWhere('i.user = :user')
            ->andWhere('i.createdAt BETWEEN :start AND :end')
            ->setParameter('user', $user)
            ->setParameter('start', $startDate)
            ->setParameter('end', $endDate)
            ->getQuery()
```

```
            ->getSingleScalarResult();

        return (float) $result;
    }

    public function getIncomeTransactionCountByDateRange(User $user, \DateTimeImmutable
$startDate, \DateTimeImmutable $endDate): int
    {
        $result = $this->createQueryBuilder('i')
            ->select('COUNT(i.id) AS total')
            ->andWhere('i.user = :user')
            ->andWhere('i.createdAt BETWEEN :start AND :end')
            ->setParameter('user', $user)
            ->setParameter('start', $startDate)
            ->setParameter('end', $endDate)
            ->getQuery()
            ->getSingleScalarResult();

        return (int) $result;
    }
}
```