ХМЕЛЬНИЦЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

Факультет інформаційних технологій

Кафедра інженерії програмного забезпечення

КВАЛІФІКАЦІЙНА РОБОТА

Вебзастосунок для персонального обліку фінансів користувача

Назва теми

Рівень вищої освіти Перший (бакалаврський)

Галузь знань 12 «Інформаційні технології»

Спеціальність 121 «Інженерія програмного забезпечення»

Освітня програма «Інженерія програмного забезпечення»

Шифр КвРІПЗ.2201102.01.10.ПЗ

Виконав здобувач IV курсу група ІПЗ-22-1 _____ Станіслав ПАПКА
Підпис Ім'я, ПРІЗВИЩЕ

Керівник канд. пед. наук, доцент _____ Наталія ПРАВОРСЬКА
Науковий ступінь, учене звання Підпис Ім'я, ПРІЗВИЩЕ

Нормоконтролер канд. тех. наук, доцент _____ Оксана ЯШИНА
Науковий ступінь, учене звання Підпис Ім'я, ПРІЗВИЩЕ

До захисту допускаю:
Завідувач кафедри інженерії
програмного забезпечення _____ Леонід БЕДРАТЮК
Підпис Ім'я, ПРІЗВИЩЕ

1 червня 2026р.
Дата

Хмельницький, 2026

ХМЕЛЬНИЦЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

Факультет Інформаційних технологій

Кафедра Інженерії програмного забезпечення

Рівень вищої освіти Перший (бакалаврський)

Галузь знань 12 «Інформаційні технології»

Спеціальність 121 «Інженерія програмного забезпечення»

Освітня програма «Інженерія програмного забезпечення»

ЗАТВЕРДЖУЮ
Завідувач кафедри ІПЗ
Л. П. Бедратюк
02.01 2026 року

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ

Папка Станіслав Федорович
Прізвище, ім'я, по батькові студента

1. Тема роботи Вебзастосунок для персонального обліку фінансів користувача

Керівник проекту Праворська Наталія Іванівна канд. пед. наук, доцент
Прізвище, ім'я, по батькові науковий ступінь, вчене звання

Затверджена наказом ректора університету від 20.01.2026 р. №7

2. Строк подання студентом проєкту на кафедру 01.06.2026 р.

3. Вихідні дані до роботи: Матеріали переддипломної практики

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

Аналіз предметної області та постановка задачі

Проєктування програмного забезпечення

Програмна реалізація та тестування програмного забезпечення

5. Перелік графічного матеріалу (із зазначенням обов'язкових креслень)

1. Діаграма класів
2. DFD-діаграма
3. Діаграма процесів

6. Консультанти розділів кваліфікаційної роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
Нормоконтроль	Яшина М. О., доцент	04.05.26	04.05.26
Антиплагіат	Форкун Ю. В., доцент	04.05.26	05.05.26

7. Дата видачі завдання « 02 » січня 2026 р.

Календарний графік

виконання кваліфікаційної роботи студентом гр. ІПЗ-22-1 за спеціальністю 121 «Інженерія програмного забезпечення»

№ з/п	Назва етапу виконання	Дата початку-завершення етапу	Примітка
1	Ознайомлення з тематикою дипломного проєктування, визначення та узгодження індивідуальних тем кваліфікаційних робіт (КвР)	01.12 - 31.12.2025	
2	Збір матеріалу за темою КвР; дослідження предметної області, в якій планується використання програмного забезпечення (ПЗ), визначення задач та вимог, розробка технічного завдання	01.01 - 20.02.2026	
3	Проєктування програмного забезпечення	21.02 - 20.03.2026	
4	Програмна реалізація з використанням відповідних засобів розробки. Тестування ПЗ	21.03 - 30.04.2026	
5	Написання вступу, загальних висновків, оформлення переліку джерел посилання та додатків. Оформлення пояснювальної записки КвР згідно вимог	01.05 - 25.05.2026	
6	Попередній захист КвР	15.05.2026	Згідно графіка
7	перевірка КвР на плагіат, нормоконтроль, отримання відгуків, рецензій та інших супровідних документів. Брошурування (зшиття) пояснювальної записки.	26.05 - 30.06.2026	
8	Здача КвР на кафедру; підготовка КвР для розміщення у репозиторій ХНУ; підготовка до захисту та захист КвР	з 01.06.2026	

Студент _____ Станіслав ПАПКА
Підпис Ім'я, ПРІЗВИЩЕ

Керівник проєкту (роботи) _____ Наталія ПРАВОРСЬКА
Підпис Ім'я, ПРІЗВИЩЕ

АНОТАЦІЯ

Тема кваліфікаційної роботи: «Вебзастосунок для персонального обліку фінансів користувача»

Автор проєкту: Папка Станіслав Федорович.

Керівник проєкту: Праворська Наталія Іванівна

Пояснювальна записка: 59 ст., 6 малюнок., 6 таблиці, 4 діаграм, 3 додатка, 30 джерел.

Графічна частина: 3 креслення ф. А3

Ключові слова: вебсервіс; облік фінансів; .NET; ASP.NET Core MVC; C#; MS SQL local; Entity Framework Core; архітектура MVC; автентифікація; база даних; візуалізація даних.

Метою проєкту роботи є розробка інтуїтивно зрозумілого, людино-орієнтованого вебзастосунку «Finance-Tracker», який забезпечує ефективний облік доходів і витрат, високу швидкість обробки даних та надійний захист фінансової інформації на базі платформи ASP.NET Core MVC.

У кваліфікаційній роботі виконано аналіз методів автоматизації фінансового обліку та моніторингу доходів і витрат для суб'єктів малого бізнесу. Спроєктовано та реалізовано вебсервіс Finance-Tracker, що поєднує використання серверної логіки на мові C# та реляційної бази даних MS SQL Server під управлінням Entity Framework Core. Розроблено інтуїтивно зрозумілий графічний інтерфейс користувача з підтримкою візуалізації фінансової аналітики та збереженням історії транзакцій у базу даних. Програмний продукт забезпечує високий рівень безпеки персональних даних завдяки впровадженню алгоритмів хешування SHA256 для захисту облікових записів.

25 травня 2026р.
Дата

Підпис

ВІДОМІСТЬ ДОКУМЕТІВ

№ рядка	Формат	Позначення документа	Найменювання документа	К-ть аркушів	№ екз.	Приміт ка
			Текстові документи			
1	А4	КвРІПЗ.2201105.01.10.ПЗ	Пояснювальна записка	59		
2	А4		Завдання на кваліфіка-	1		
			ційну роботу			
3	А4		Анотація	1		
			Графічні документи	1		
4	А3	КвРІПЗ.2201105.01.10.Е8	Діаграма класів	1		
5	А3	КвРІПЗ.2201105.01.10.Е8	DFD-діаграма	1		
6	А3	КвРІПЗ.2201105.01.10.Е8	Діаграма процесів	1		

Змн.	Арк.	№ докум.	Підпис	Дата	КвРІПЗ.2201105.01.10.ПЗ		
Розроб.		Папка С.Ф.		25.05	Вебзастосунок для персонального обліку фінансів користувача	Літ.	Арк. 5 / Акрушів 59
Перевір.		Праворська Н.І.		25.05			
Реценз.						ХНУ, ІПЗ-22-1	
Н. Контр.		Яшина О. М.		25.05	Пояснювальна записка		
Затверд.		Бедратюк Л. П.		25.05			

ЗМІСТ

ВСТУП

Сьогодення неможливо уявити без тотальної цифровізації всіх сфер людської діяльності, де інформаційні технології перетворилися з допоміжних інструментів на фундаментальну основу функціонування суспільства. Обчислювальні системи пройшли стрімкий шлях еволюції від громіздких машин для вузькоспеціалізованих розрахунків до персоналізованих екосистем, що супроводжують користувача у кожній справі. Якщо на ранніх етапах розвитку техніки взаємодія з комп'ютером вимагала глибоких технічних знань, то сучасна парадигма передбачає повну адаптацію програмного забезпечення під індивідуальні потреби людини чи конкретного бізнесу, та інтерфейс більш дружнім та орієнтованим на кінцевий результат.

Особливої актуальності питання автоматизації набуває у сфері управління ресурсами, зокрема у фінансовому секторі малого бізнесу. Сучасний темп економічних процесів вимагає від підприємців високої швидкості прийняття рішень та абсолютного контролю над грошовими потоками. Хоча ринок пропонує широкий спектр готових хмарних рішень, вони часто мають надлишковий функціонал, вимагають регулярних витрат на підписку або створюють потенційні ризики для конфіденційності через зберігання даних на сторонніх серверах. Це породжує стійкий попит на створення автономних, надійних та легких у використанні вебзастосунків, що базуються на перевірених технологіях, таких як архітектура MVC та платформа .NET.

Цифрова трансформація малого бізнесу стимулює розробку спеціалізованих інструментів, які дозволяють автоматизувати рутинні операції обліку без залучення професійних бухгалтерів. Впровадження таких систем вимагає вирішення комплексу науково-практичних завдань, що охоплюють проєктування масштабованих баз даних на базі MS SQL Server та забезпечення високої швидкодії за допомогою Entity Framework Core. Розробка вебсервісу для

персонального моніторингу фінансів є своєчасною відповіддю на сучасні вимоги до прозорості, безпеки та мобільності процесів управління капіталом.

Важливою науково-технічною проблемою під час проєктування таких локальних систем є усунення затримок під час обробки та запису інформаційних потоків на дискові носії. Традиційне використання реляційних баз даних у локальному середовищі часто супроводжується ризиками тимчасового блокування графічного інтерфейсу користувача під час виконання важких дискових операцій введення-виведення (I/O). Вирішення цього завдання потребує впровадження сучасних парадигм асинхронного програмування на базі маркерів async/await у межах екосистеми .NET. Це дозволяє здійснювати неблокуючу взаємодію між презентаційним шаром та фізичним сховищем даних, забезпечуючи стабільно низький час відгуку інтерфейсу та високу відмовостійкість застосунку навіть під час генерації масштабних аналітичних зрізів за тривалі звітні періоди.

Окрім обчислювальної швидкодії, на перший план виходить концепція цифрового суверенітету користувача над власною комерційною інформацією. Стрімке зростання кількості кібератак та регулярні витоки конфіденційних даних із хмарних сховищ актуалізують розробку архітектурних моделей класу «Local-first». Використання легковагового примірника системи управління базами даних MS SQL Server LocalDB у поєднанні з жорсткою серверною валідацією вхідних моделей даних дозволяє ізолювати фінансову історію в межах локального пристрою користувача. Такий інженерний підхід повністю нівелює потребу в постійному мережевому з’єднанні з віддаленими серверами, мінімізує вектор потенційних зовнішніх атак та гарантує безкомпромісну конфіденційність облікових процесів, що є критично важливим для представників малого бізнесу та самозайнятих осіб.

Метою кваліфікаційної роботи є розробка інтуїтивно зрозумілого, людино-орієнтованого вебзастосунку «Finance-Tracker», який забезпечує ефективний облік доходів і витрат, високу швидкість обробки даних та надійний захист фінансової інформації на базі платформи ASP.NET Core MVC. Реалізація

поставленої мети передбачає послідовне вирішення низки науково-практичних завдань, що розпочинаються з комплексного аналізу предметної області та сучасних тенденцій у сфері автоматизації фінансового обліку для виявлення недоліків існуючих комерційних рішень. Наступним етапом є порівняльна характеристика сучасних технологій веброзробки та обґрунтування вибору оптимального стеку технологій, що включає ASP.NET Core, Entity Framework та MS SQL Server. Проєктна частина роботи передбачає проєктування архітектури системи, розробку діаграм діяльності та моделей даних, які детально описують процеси взаємодії користувача з фінансовими модулями. Важливою складовою є обґрунтування методів автентифікації та захисту даних, зокрема реалізація сервісу UserService та використання алгоритмів хешування SHA256 для гарантування безпеки облікових записів. Програмна реалізація охоплює створення серверної логіки на мові C# для обробки транзакцій та розробку адаптивного графічного інтерфейсу на базі Razor Views та Bootstrap. Завершальним етапом є проведення комплексного тестування для оцінки коректності фінансових розрахунків, надійності збереження даних та зручності навігації.

Об’єктом дослідження у даній роботі є процес автоматизованого обліку та моніторингу фінансових ресурсів у цифровому середовищі за допомогою сучасних вебтехнологій. Предметом дослідження виступають методи, засоби та алгоритми розробки безпечних вебзастосунків на базі архітектури MVC, що забезпечують оперативний контроль за персональними та корпоративними фінансами.

Розроблений програмний продукт є готовим інструментом для прозорого ведення обліку, що не потребує впровадження складних корпоративних систем. Запропоноване рішення може бути використане як базова платформа для цифровізації звітності малих підприємств, де критично важливою є простота використання та точність отриманих даних. Отримані результати можуть бути інтегровані у навчальний процес при вивченні дисциплін, пов’язаних із проєктуванням інформаційних систем та веброзробкою.

1. ДОСЛІДЖЕННЯ ПРЕДМЕТНОЇ ОБЛАСТІ ТА ПОСТАНОВКА ЗАДАЧІ

1.1 Аналіз предметної області, її структурних та функціональних особливостей

Сучасний світ перебуває у стані безперервної, інтенсивної та всеохоплюючої цифрової трансформації, яка докорінно змінює архітектуру суспільно-економічних відносин та практично всі аспекти життєдіяльності людства. Обчислювальні комплекси та автоматизовані системи, які ще кілька десятиліть тому сприймалися виключно як вузькоспеціалізоване професійне обладнання для виконання громіздких науково-технічних розрахунків, сьогодні остаточно інтегрувалися у повсякденний побут, глобальні освітні процеси та корпоративний сектор. Ця стрімка еволюція апаратно-програмних засобів спровокувала системні зрушення не лише у кількісних характеристиках швидкодії обчислювальних пристроїв, а й у самій філософії та методології взаємодії людини з розподіленими інформаційними системами.

У ретроспективному розрізі еволюція засобів автоматизації та моніторингу персонального фінансового обліку пройшла кілька визначальних та концептуальних етапів:

- на початкових стадіях цифровізації та зародження систем класу PFM (Personal Finance Management) користувачі були змушені адаптуватися під жорстку, низькорівневу машинну логіку, взаємодіючи з суворими текстовими або консольними інтерфейсами, а також самостійно конструюючи складні математичні формули в ізольованих операційних середовищах;
- згодом, етап масового поширення графічних оболонок (GUI) та перших настільних табличних процесорів істотно знизив поріг входження, проте тривалий час архітектурна функціональність таких систем залишалася

надмірно перевантаженою, неструктурованою та вимагала від оператора наявності спеціалізованої бухгалтерської або економічної підготовки;

– на сучасному етапі розвитку індустрії програмного забезпечення спостерігається глобальний перехід до парадигми людино-орієнтованого дизайну (Human-Centered Design) та реактивних інтерфейсів, де ключовим критерієм стає когнітивний комфорт користувача.

Мета сучасної веброзробки у фінансовому секторі полягає у проєктуванні таких ергономічних інтерфейсів, які б повністю нівелювали бар'єр між складними реляційними алгоритмами збереження, агрегації чи аналізу великих масивів даних та повсякденними утилітарними потребами кінцевого користувача, створюючи умови для безперешкодного, легкого та оперативного керування активами.

Ключовим каталізатором та технологічним фундаментом таких трансформацій став стрімкий прогрес у галузі побудови вебресурсів та високопродуктивних систем обробки структурованих даних, зокрема повсюдне впровадження архітектурного шаблону MVC (Model-View-Controller) та системна еволюція кросплатформеної екосистеми .NET. Використання сучасних компонентних фреймворків дозволило прикладним технологіям вийти на якісно новий рівень: від лінійного виконання жорстко запрограмованих скриптів до гнучкого, контекстно-залежного моделювання господарських операцій, розпізнавання намірів користувача та миттєвої генерації точних аналітичних зрізів у режимі реального часу.

Проте, попри очевидні успіхи світових індустріальних рішень, поточний стан екосистеми фінансового програмного забезпечення характеризується наявністю низки критичних структурних недоліків:

– домінування хмарної моделі розгортання: більшість найпотужніших та функціональних комерційних сервісів функціонують виключно у Cloud-only середовищі, що створює перманентні загрози для конфіденційності комерційної таємниці та недоторканності персональних даних;

- загрози несанкціонованого доступу: систематична передача чутливої фінансової інформації, логів транзакцій та балансів рахунків на сторонні віддалені сервери несе в собі непереборні ризики компрометації даних внаслідок цілеспрямованих кібератак або порушення угод про нерозголошення (NDA) з боку провайдерів;
- жорстка інтернет-залежність: централізована архітектура сторонніх хмарних платформ зумовлює повну залежність працездатності застосунку від наявності стабільного та високошвидкісного мережевого зв’язку, що є абсолютно неприпустимим для автономних комп’ютерних систем, які функціонують в умовах ізольованих або обмежених Air-gapped середовищ.

Окрім того, значні мережеві затримки (latency), ризики деградації каналів зв'язку та висока вартість регулярних ліцензійних підписок роблять використання комерційних хмарних API економічно неефективним та обтяжливим для суб'єктів малого бізнесу чи індивідуальних користувачів.

У відповідь на ці технологічні та економічні виклики в сучасній IT-індустрії сформувався та активно розвивається альтернативний концептуальний підхід до розробки програмного забезпечення, відомий як концепція Local-first software. Ця парадигма передбачає умисне перенесення основних обчислювальних потужностей, бізнес-логіки та аналітичних алгоритмів безпосередньо на кінцеві клієнтські пристрої або виділені приватні локальні сервери користувачів. Така локалізація обробки гарантує абсолютну незалежність від зовнішньої мережевої інфраструктури та забезпечує безпрецедентний рівень захисту інформації, оскільки будь-яка фінансова активність фізично ніколи не залишає внутрішній захищений периметр пристрою користувача.

Однак розгортання комплексних систем обліку та аналізу капіталу в межах автономних пристроїв неминуче наштовхується на складну інженерну проблему оптимізації наявних обчислювальних ресурсів. Сучасні реляційні бази даних для швидкої обробки складних аналітичних вибірок потребують значних обсягів

оперативної пам’яті (RAM) та низькорівневої оптимізації архітектури доступу, якими далеко не завжди володіють стандартні робочі станції середнього класу.

Аналіз передових тенденцій у галузі інженерії програмного забезпечення виділяє ефективні напрямки вирішення цієї проблеми через синтез таких рішень:

- застосування математичних методів оптимізації та нормалізації структур даних на рівні проєктування реляційних схем;
- впровадження сучасного високотехнологічного об’єктно-реляційного відображення за допомогою ORM-системи Entity Framework Core;
- перехід на легковагові, але повнофункціональні транзакційні СУБД, зокрема MS SQL Server LocalDB, що дозволяє радикально підвищити загальну продуктивність і мінімізувати апаратний слід системи.

Спільне використання оптимізованих форматів внутрішньої передачі даних, механізмів кешування та асинхронного виконання потоків за рахунок вбудованих можливостей платформи .NET дозволяє ефективно розподіляти фонове навантаження на центральний процесор (CPU).

Окрім подолання суто апаратних та системних обмежень, важливою функціональною особливістю цієї предметної області є необхідність глибокої інтеграції ядра застосунку з персональними даними для формування фактологічно точних, релевантних та оперативних відповідей. Це досягається завдяки застосуванню передових технологій пошукової генерації звітів на основі структурованої локальної бази даних, без потреби у використанні сторонніх хмарних аналітичних сервісів чи складних ручних маніпуляцій з вихідним кодом з боку користувача.

Таким чином, проєктування та програмна реалізація вебзастосунку «Finance-Tracker» постає як надзвичайно актуальне науково-практичне завдання сучасної інженерії програмного забезпечення. Його розробка дозволяє створити захищений, відмовостійкий та оптимізований інструмент, який органічно поєднує високі аналітичні можливості сучасних вебсистем із гарантуванням повної конфіденційності інформації користувача. Отримані в ході дослідження та

розробки результати можуть бути успішно використані як базовий архітектурний фреймворк для створення спеціалізованих фінансових асистентів, а також інтегровані у навчальний процес університетів при вивченні дисциплін, пов'язаних із проєктуванням, розробкою та тестуванням складного, безпечного програмного забезпечення.

1.2 Аналіз останніх публікацій, досліджень та існуючих рішень предметної області

Сучасні комплексні системи фінансового управління корпоративного рівня, такі як SAP, Oracle Financials або 1С, демонструють високу точність, гнучкість та широку функціональність у задачах обробки великих масивів комерційної звітності й аудиту. Однак їхня монолітна архітектура вимагає значних обчислювальних ресурсів, складного інфраструктурного супроводу та високовартісного апаратного забезпечення. Наприклад, стандартна ERP-система для підтримки великої кількості одночасних користувацьких запитів, розподілених транзакцій та складних аналітичних SQL-процедур потребує розгортання потужних серверних станцій із великим обсягом оперативної пам’яті (RAM) та високошвидкісних дискових масивів лише для підтримання базової стабільної роботи у фоновому режимі. Це робить їхнє пряме використання у форматі легких автономних рішень для персонального обліку, самозайнятих осіб або малого бізнесу практично неможливим без проведення спеціальної глибокої програмної оптимізації та рефакторингу логіки обробки даних.

Аналіз сучасних інженерних підходів у галузі веброзробки показує, що для вирішення проблеми ресурсомісткості та громіздкості фінансового програмного забезпечення застосовують передові методи оптимізації програмної архітектури та стиснення логіки обробки інформаційних потоків. До основних методів, що дозволяють успішно адаптувати важкі фінансові алгоритми до умов обмежених ресурсів локальних вебресурсів, належать підходи, орієнтовані на оптимізацію шару доступу до даних.

Першим та ключовим інженерним методом є оптимізація об’єктно-реляційного відображення (ORM), що дозволяє здійснити значний прорив у швидкодії локальних вебсервісів. Суть цього підходу полягає у суттєвому зменшенні навантаження на реляційну базу даних через використання невідстежуваних запитів (AsNoTracking), явне визначення об'єктних проєкцій для вилучення лише необхідних полів замість завантаження всієї сутності, а також впровадження асинхронних маркерів виконання операцій. Використання таких механізмів забезпечує декількакратне зниження часу відгуку інтерфейсу системи, мінімізує ризики взаємного блокування потоків та дозволяє стабільно працювати з великими масивами транзакцій навіть на клієнтському апаратному забезпеченні початкового або середнього рівня.

Для обґрунтування доцільності розробки власного локального програмного забезпечення було проведено порівняльний аналіз існуючих аналогів, представлених на ринку. Сьогодні в індустрії домінують два альтернативні підходи, кожний з яких має суттєві архітектурні обмеження: використання закритих комерційних хмарних платформ (зокрема Monefy або PocketGuard) та універсальні десктопні системи бухгалтерського обліку. Хоча мобільні та хмарні сервіси успішно вирішують проблему простоти доступу та кросплатформеної мобільності, вони мають критично низький рівень конфіденційності, оскільки абсолютно всі транзакції користувачів обробляються, аналізуються та зберігаються на сторонніх серверах, що створює загрозу витоку приватної фінансової інформації. З іншого боку, класичні десктопні програми часто є перевантаженими професійними налаштуваннями, складними бухгалтерськими проводками та громіздким інтерфейсом, що вимагає тривалого навчання. Виявлені недоліки формують чітку потребу в розробці легковагового, автономного вебзастосунку «Finance-Tracker» із інтегрованою локальною системою безпеки, персоналізованими довідниками та реактивною підсистемою аналітики.

Порівняльна характеристика підходів до автоматизації фінансового обліку наведена в таблиці 1.1.

Таблиця 1.1 – Порівняльна характеристика підходів до фінансового обліку

Характеристика	Хмарні сервіси (Monefy Cloud)	Десктопні платформи (1С / Excel)	Розроблюваний вебзастосунок
Приватність даних	Низька (дані на зовнішніх серверах)	Висока (дані на одному ПК)	Максимальна (сучасні методи шифрування та локальна БД)
Залежність від мережі	Повна	Відсутня	Відсутня
Вимоги до ресурсів	Мінімальні	Високі (завантажують RAM)	Оптимізовані (.NET 6.0 / EF Core)
Аналітичні можливості	Стандартні	Обмежені без налаштувань	Вбудований аналітичний дашборд
Складність інтерфейсу	Низька	Висока (професійна)	Низька (мінімалістичний дизайн)

Аналіз існуючих рішень та методів оптимізації показав, що найбільш доцільним для досягнення мети кваліфікаційної роботи є розробка власного вебзастосунку, який поєднує методи архітектурної оптимізації та індивідуально налаштовану систему безпеки. Це дозволить створити швидку, автономну систему з інтуїтивно зрозумілим та знучиним у використанні інтерфейсом, яка перевершує аналоги за критеріями захисту фінансової інформації та зручності оперативного контролю за капіталом в режимі реального часу.

1.3 Аналіз вимог до програмного забезпечення

На основі проведеного комплексного аналізу предметної області, а також критичного виявлення недоліків існуючих рішень для моніторингу капіталу (зокрема ризиків витоку конфіденційної інформації у хмарних сервісах та надмірної складності інтерфейсів для пересічного користувача), було сформовано інженерно обґрунтований перелік вимог до проектованого вебзастосунку «Finance-Tracker». Метою цього етапу є чітке, несуперечливе та верифіковане визначення функціональних можливостей системи, а також граничних технічних умов, у яких вона має функціонувати для забезпечення абсолютної надійності, точності та приватності фінансового обліку.

Усі вимоги до розроблюваного програмного продукту було класифіковано на дві базові категорії: функціональні, які деталізують безпосередні експлуатаційні можливості системи та реакцію компонентів на дії користувача, й нефункциональні, що регламентують загальні атрибути якості, критерії безпеки, метрики швидкодії та архітектурні обмеження.

До функціональних вимог, що описують набір операцій для повного задоволення потреб кінцевого користувача, віднесено такі підсистеми:

а) підсистема автентифікації та управління профілем користувача, яка повинна забезпечувати:
   1) можливість реєстрації нових облікових записів із обов'язковою первинною валідацією полів;
   2) безпечний вхід до системи з розгортанням захищеної локальної сесії;
   3) надійну ізоляцію даних персональних профілів за допомогою спеціалізованого сервісу UserService;

б) підсистема управління фінансовими операціями, що відповідає за автоматизацію обліку та реалізує:

4) повний життєвий цикл транзакцій, включаючи створення, перегляд, інтерактивне редагування та видалення записів (CRUD-операції);
5) обов'язкову сувору прив'язку кожної фінансової операції до конкретної категорії доходів або витрат;
6) фіксацію часових міток, числових значень суми з високою точністю та текстових коментарів користувача;

в) підсистема адміністрування категорій, яка надає користувачеві інструменти для самостійного формування багаторівневої структури джерел надходжень та напрямків витрат з метою гнучкої персоналізації обліку;

г) підсистема аналітичної звітності та візуалізації даних, призначена для:
1) динамічного обчислення й відображення поточного загального балансу в реальному часі;
2) статистичного групування фінансових показників за обрані часові періоди (місяць, квартал, рік);
3) рендерингу інтерактивних графіків і кругових діаграм розподілу витрат на головній панелі (Dashboard); д) підсистема взаємодії з базою даних, що функціонує в межах Persistence-шару через об'єктно-реляційне відображення Entity Framework Core та гарантує повну транзакційну цілісність інформації при виконанні складних агрегаційних запитів до SQL-сервера.

Нефункціональні вимоги визначають системні характеристики застосунку, обмеження цільового середовища виконання, а також критерії якості програмного коду:

а) вимоги до продуктивності та швидкодії:
1) мінімізація часу відгуку системи при формуванні аналітичних звітів (швидкість обробки запиту не повинна перевищувати 0.2 секунди);

2) висока ефективність виконання SQL-операцій завдяки оптимізації контексту даних ApplicationDbContext та використанню LINQ-виразів;

б) вимоги до безпеки та конфіденційності:

1) обов'язкове незворотне хешування паролів на стороні сервера за допомогою криптографічного алгоритму SHA256;
2) впровадження механізмів ValidateAntiForgeryToken для захисту від міжсайтової підробки запитів (CSRF-атак);
3) фізичне обмеження доступу до файлів бази даних у межах локального користувацького середовища;

в) вимоги до архітектури програмного забезпечення, які базуються на суворому дотриманні трирівневого патерну MVC (Model-View-Controller) для забезпечення слабкої зв'язності компонентів, високої масштабованості системи та повної ізоляції бізнес-правил від презентаційного шару;

г) вимоги до інтерфейсу користувача (UI/UX):

1) застосування адаптивної компоновки елементів на базі фреймворку Bootstrap 5 для коректного відображення на пристроях із різною роздільною здатністю екрана;
2) підтримка темної теми (Dark Mode) для зниження когнітивного навантаження під час тривалої роботи; д) вимоги до надійності та відмовостійкості, що полягають у здатності системи підтримувати ACID-властивості фінансових транзакцій і автоматично зберігати структурну цілісність даних у базі MS SQL Server LocalDB навіть у разі раптового або некоректного завершення сесії користувача чи критичного збою апаратного забезпечення.

Сформований комплексний перелік вимог є архітектурним фундаментом для подальшого проєктування логічної структури вебзастосунку, розробки ER-діаграми бази даних та вибору інструментального стеку. Неухильне дотримання цих критеріїв під час програмної реалізації гарантує створення автономного,

стійкого до навантажень та оптимізованого програмного продукту, що повністю відповідає меті та завданням цієї кваліфікаційної роботи.

1.4 Висновок до першого розділу. Постановка задачі

У першому розділі проведено комплексний аналіз предметної області автоматизації обліку фінансових ресурсів за допомогою вебтехнологій. Встановлено, що попри розвиток хмарних фінтех-сервісів, домінуюча архітектура їх розгортання створює критичні ризики для конфіденційності даних та зумовлює повну залежність від інтернет-з’єднання. Аналіз чинних рішень показав потребу ринку в переході до приватних фінансових помічників — легковагових вебзастосунків, здатних функціонувати в ізольованих локальних мережах. Дослідження підтвердило, що використання платформи .NET 6.0, патерну MVC та технології Entity Framework Core дозволяє створювати високопродуктивні системи з мінімальним часом відгуку на апаратному забезпеченні середнього рівня. Крім того, виявлено потребу в інтеграції підсистем динамічної візуалізації даних для оперативного контролю за капіталом у реальному часі.

На основі проведеного аналізу сформовано перелік функціональних та нефункціональних вимог до програмного забезпечення. Визначено, що система повинна мати модульну архітектуру, адаптивний інтерфейс та забезпечувати високий рівень захисту даних за допомогою алгоритмів хешування. Критично важливим критерієм під час декомпозиції вимог визначено забезпечення суворої реляційної цілісності та каскадної узгодженості даних на рівні сховища. Проєктована логіка взаємодії через Entity Framework Core має гарантувати перевірку унікальності об'єктів для виключення дублювання записів. Додатковою вимогою виступило проєктування ергономічного графічного модуля динамічного управління локальними довідниками категорій без прямого маніпулювання таблицями бази даних MS SQL Server LocalDB, що дозволить приховати

складність SQL-процедур від оператора, мінімізувати кількість кроків для реєстрації операцій та забезпечити наочність метрик.

Метою кваліфікаційної роботи є розробка інтуїтивно зрозумілого, людино-орієнтованого вебзастосунку «Finance-Tracker», який забезпечує ефективний облік доходів і витрат, високу швидкість обробки даних та надійну конфіденційність фінансової інформації на базі платформи ASP.NET Core MVC.

Об’єктом дослідження є процес автоматизованого обліку та моніторингу фінансових ресурсів у цифровому середовищі за допомогою вебтехнологій.

Предметом дослідження є методи, засоби та архітектурні патерни розробки безпечних вебзастосунків на базі ASP.NET Core MVC, а також алгоритми обробки транзакцій та візуалізації фінансової звітності.Для досягнення поставленої мети необхідно виконати такі завдання:

а) проаналізувати предметну область, існуючі програмні рішення та методи архітектурної оптимізації вебзастосунків;
б) сформувати вимоги до програмного забезпечення та обґрунтувати вибір технологічного стеку розробки;
в) спроєктувати модульну архітектуру вебсервісу та реляційну базу даних для збереження фінансових операцій;
г) розробити програмний комплекс, що включає:
    1) презентаційний рівень на базі Razor Views та Bootstrap;
    2) сервісну логіку управління користувачами та автентифікації;
    3) підсистему управління транзакціями та категоріями;
    4) аналітичний модуль для графічного відображення фінансових показників;
д) провести тестування розробленого вебзастосунку на предмет коректності виконання функціональних вимог, стійкості до помилок введення та ефективності роботи з базою даних.

2. ПРОЕКТУВАННЯ СТРУКТУРИ ТА КОМПОНЕНТІВ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

2.1 Проектування архітектури програмного забезпечення

Архітектура розробленого програмного комплексу «Finance-Tracker» побудована за модульним принципом із чітким логічним розділенням рівнів відповідальності на основі патерну MVC. Оскільки головною вимогою до системи є надійність збереження та конфіденційність фінансових даних в умовах можливого локального використання, архітектура не передбачає передачі чутливої інформації на зовнішні хмарні сервери. Усі ключові компоненти системи, включаючи обчислювальне ядро обробки транзакцій та реляційну базу даних, розгортаються у захищеному периметрі.

Загальна структура розробленого вебзастосунку складається з п'яти ключових взаємопов'язаних модулів, кожен з яких інкапсулює специфічний набір функцій:

а) презентаційний рівень (контролери та Razor-представлення), який виконує роль точки входу в систему та оркестратора запитів. Цей компонент відповідає за відображення графічного інтерфейсу користувача та включає:

1) рендеринг вебсторінок та інтерактивних елементів панелі керування;
2) управління станом сесії користувача для збереження авторизаційного контексту в реальному часі;
3) обробку користувацького введення: фіксація сум, вибір категорій та описів операцій;
4) забезпечення валідації даних на стороні клієнта для швидкого відгуку системи;
5) динамічне відображення оновлених фінансових підсумків;

б) рівень конфігурації та інфраструктури (модулі Program.cs та appsettings.json), що є єдиним центром управління глобальними параметрами вебзастосунку. Він містить:
    1) шляхи до системних ресурсів: рядки підключення до бази даних, шляхи до статичних файлів та логів;
    2) реєстрацію залежностей (Dependency Injection) для сервісів та контексту даних;
    3) інтеграцію стилів Bootstrap та кастомних CSS-правил для адаптивності інтерфейсу;
в) підсистема управління даними та доступом (модуль UserService та Identity), яка відповідає за персистентність облікових записів та реалізує:
    1) повний цикл CRUD-операцій для роботи з профілями користувачів;
    2) забезпечення безпеки через хешування паролів та управління правами доступу;
г) обчислювальне ядро обробки транзакцій (модуль TransactionEngine), що є ключовим логічним компонентом системи і керує фінансовими потоками. До його функцій належать:
    1) ініціалізація контексту даних в оперативній пам'яті сервера з максимальною оптимізацією запитів;
    2) формування структурованих команд для бази даних на основі дій користувача;
    3) генерація балансових звітів з урахуванням фільтрів за часовими періодами та типами витрат;
    4) автоматична перевірка лімітів та кореляція доходів і витрат;
д) підсистема аналітики та візуалізації (модуль ReportingEngine), яка забезпечує здатність вебзастосунку працювати з накопиченими масивами інформації. Процес включає:
    1) автоматичний збір даних із таблиць транзакцій та категорій;

2) агрегацію фінансових показників (обчислення сум, середніх значень, відсоткових часток);
3) перетворення числових даних у формат, придатний для графічного відображення;
4) збереження аналітичних зрізів у локальній системі звітності;
5) здійснення фільтрації найрелевантніших записів під час перегляду історії.

Запропонована архітектурна модель формує замкнений цикл обробки фінансової інформації: від отримання даних користувача через презентаційний рівень до їхнього збереження у базі знань та видачі аналітичної відповіді виключно на локальних обчислювальних потужностях.

Для наочного відображення взаємодії описаних компонентів розроблено загальну структурну схему архітектури вебзастосунку, яка демонструє потік управління та передачу даних між підсистемами (Рисунок 2.1).

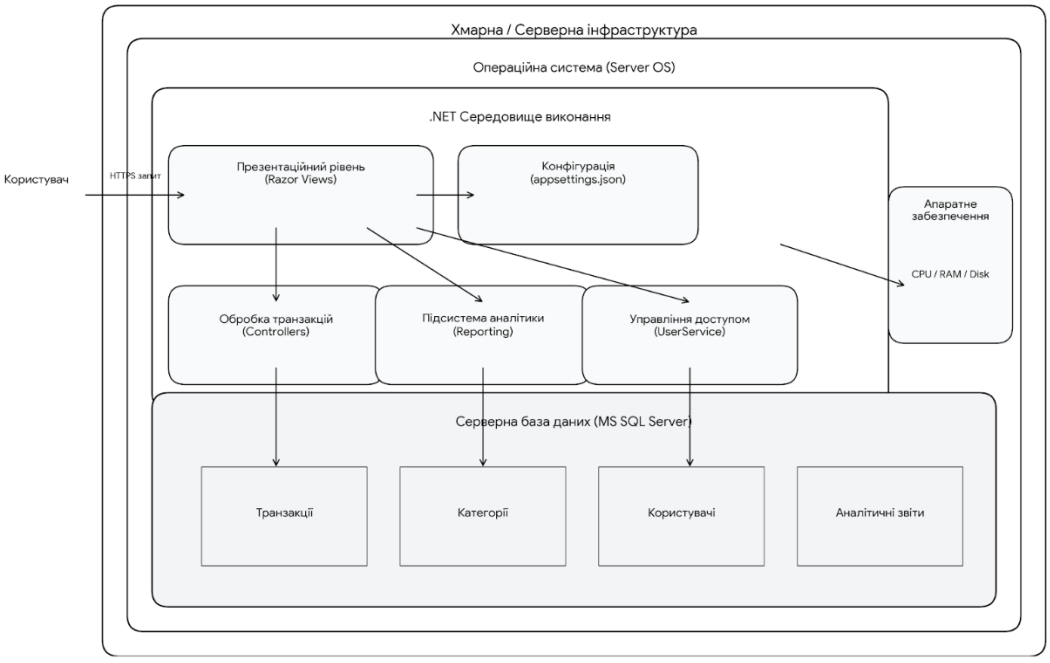


Рисунок 2.1 – Структурна схема архітектури розробленого вебзастосунку

Життєвий цикл обробки типового запиту в межах спроєктованої архітектури відбувається за таким сценарієм:

- ініціалізація та введення – користувач через форми презентаційного рівня вводить дані про фінансову операцію;
- перевірка та валідація – запит маршрутизується до контролерів, які перевіряють логічну цілісність даних (наявність суми, обраної категорії). Якщо дані коректні, вони вилучаються для подальшої обробки;
- обробка бізнес-логіки – оригінальні дані, ідентифікатор користувача та поточний стан балансу передаються до обчислювального ядра (TransactionEngine) для формування запису;
- персистентність та запис – обчислювальне ядро застосовує правила Entity Framework і виконує транзакційний запис у MS SQL Server. Дані асинхронно повертаються на презентаційний рівень для миттєвого оновлення балансу в інтерфейсі;
- аналітика – після завершення операції оновлений масив даних доступний для підсистеми аналітики (ReportingEngine) для перерахунку графічних діаграм.

Для деталізації фізичного розміщення компонентів системи розроблено діаграму розгортання. Оскільки вебзастосунок спроєктовано для автономної роботи, усі вузли обробки сконцентровані на одному сервері або робочій станції користувача. Опис архітектури розгортання включає такі рівні:

а) рівень апаратного забезпечення:
   1) центральний процесор (CPU) для виконання серверної логіки застосунку;
   2) оперативна пам'ять (RAM), де розміщуються активний контекст бази даних та кеш сесії;
   3) дискова підсистема для зберігання файлів бази даних та статичного контенту;

б) рівень операційної системи та середовища виконання:

1) серверна операційна система, що забезпечує функціонування вебсервера;
2) середовище виконання .NET 6.0 із завантаженими бібліотеками та залежностями;

в) рівень програмних артефактів:

1) зібрані модулі вебзастосунку (контролери, моделі, представлення);
2) реляційна база даних MS SQL Server.

Окрему увагу на етапі проєктування приділено структурі зберігання даних. Для забезпечення персистентності фінансової історії та можливості її подальшого аналізу було розроблено реляційну схему. Кожна операція користувача представляє собою структурований запис у таблиці. Опис ключів, що використовуються для формування структури збереження фінансових операцій, наведено в таблиці 2.1.

Таблиця 2.1 – Структура об'єктів таблиці збереження транзакцій

Ключове поле	Тип даних	Призначення та опис
TransactionId	Integer	Унікальний ідентифікатор операції в системі
Amount	Decimal	Містить безпосередній грошовий вираз операції
Timestamp	DateTime	Часова мітка збереження (використовується для сортування історії)

Приклад об'єкта, що зберігається у базі даних вебзастосунку, має такий вигляд:

{
"TransactionId": 1024,

"Amount": 1500.50,
"Timestamp": "2026-03-20T10:15:30",
"CategoryId": 5,
"Note": "Оплата комунальних послуг"
}

Такий підхід дозволяє вебзастосунку швидко зчитувати історію попередніх операцій та проводити їх групування під час генерації аналітичних звітів. Важливим компонентом архітектури є організація взаємозв'язків у MS SQL Server. На відміну від файлових систем, реляційна СУБД спроєктована для забезпечення цілісності даних. Процес організації та пошуку включає:

- нормалізацію даних – кожен атрибут фінансової операції перетворюється у чітко визначений числовий або текстовий тип;
- індексацію – збережені транзакції групуються за датою та користувачем для прискорення вибірки великих масивів даних;
- реляційне зіставлення – для вилучення релевантної інформації система алгоритмічно об'єднує таблиці транзакцій та категорій, обираючи записи, які відповідають фільтрам користувача.

Така багаторівнева архітектура забезпечує високу швидкість обробки інформації та гарантує, що вебзастосунок залишається повністю функціональним при зростанні обсягів даних, не перевантажуючи інші рівні системи.

2.2 Проектування модулів

На етапі проєктування внутрішньої структури вебзастосунку було проведено декомпозицію загальної системи на незалежні функціональні одиниці (модулі). Кожен модуль спроєктовано таким чином, щоб мінімізувати зв'язність коду та забезпечити високий рівень інкапсуляції логіки. Взаємодія між модулями відбувається через чітко визначені програмні інтерфейси та виклики функцій у

межах .NET-середовища. Детальне проєктування модулів системи включає такі структурні елементи:

а) модуль управління базовими налаштуваннями (на базі конфігураційних файлів appsettings.json та Program.cs):
   1) спроєктовано як централізований вузол конфігурації, що ініціалізується під час запуску системи;
   2) містить логіку автоматичної перевірки доступності локального екземпляра MS SQL Server та цілісності бази даних;
   3) інкапсулює візуальні та технічні налаштування, які передаються на презентаційний рівень для коректного рендерингу інтерфейсу;

б) підсистема управління станом та персистентністю (модуль ApplicationDbContext):
   1) спроєктовано набір функцій для роботи з локальною базою даних:
      – функція читання історії транзакцій, що виконує десеріалізацію даних із SQL-таблиць у оперативну пам'ять;
      – функція збереження поточного стану, яка забезпечує атомарний запис фінансових операцій на локальний накопичувач;
      – функція безпечного видалення записів та очищення історії користувача;
   2) реалізовано алгоритми обробки виключень для уникнення конфліктів у базі даних під час одночасного виконання кількох операцій;

в) модуль обчислювального ядра фінансової логіки (модуль TransactionEngine):
   1) спроєктовано механізм кешування активних фінансових даних в оперативну пам'ять пристрою, що запобігає надлишковому зверненню до дискової підсистеми під час кожного розрахунку;

2) розроблено алгоритм форматування вхідних даних: – отримання числових значень та текстових описів від користувача; – додавання системних метаданих (ідентифікатор користувача, часова мітка); – конкатенація та групування останніх операцій для збереження контексту поточної аналітики;
3) реалізовано функцію асинхронної обробки транзакцій, яка забезпечує миттєвий відгук інтерфейсу в міру виконання обчислень;

г) модуль інтелектуального аналізу та звітності (аналог RAG-системи, модуль ReportingEngine):
1) спроєктовано конвеєр обробки фінансових даних, що складається з таких етапів:
– ідентифікація формату та типу операції;
– виклик алгоритму агрегації сумарних показників;
– сегментація загального масиву даних на фрагменти за категоріями та часовими проміжками для збереження точності звітності;
2) спроєктовано підсистему семантичної вибірки:
– функція генерації аналітичних зрізів для кожного сегмента даних;
– алгоритм обчислення відсоткових часток витрат у межах загального бюджету;
– вилучення найрелевантніших показників та їх передача до презентаційного рівня;

д) презентаційний модуль, що відповідає за інтерфейс вебзастосунку (модуль HomeController та Razor Views):
1) спроєктовано життєвий цикл сторінки вебзастосунку, який оновлюється на базі актуальних даних із сервера;

2) розроблено структуру поділу екрана: – бічна панель для навігації між розділами, управління категоріями та налаштуваннями профілю; – основна робоча область для відображення динамічних графіків та таблиць із історією операцій;
3) імплементовано алгоритм перехоплення помилок обробки даних з їх подальшим безпечним виведенням у графічному інтерфейсі користувача.

Спроєктована структура модулів забезпечує надійне функціонування локальної системи та дозволяє легко розширювати функціонал без необхідності рефакторингу всього коду. Детальне проєктування модуля інтелектуального аналізу фінансових даних вимагає формалізації алгоритму обробки вхідних показників. Процес функціонування підсистеми ReportingEngine концептуально поділяється на два незалежні процеси: процес індексування фінансової історії та процес аналітичної вибірки. Логіка виконання процесу індексування складається з таких кроків:

а) етап завантаження та парсингу:
   1) отримання даних про операцію (сума, категорія, опис) через презентаційний рівень;
   2) вилучення неструктурованого тексту приміток із очищенням від службових символів;

б) етап фрагментації та групування даних:
   1) поділ загального масиву транзакцій на смислові фрагменти за датою або типом операції;
   2) застосування механізмів перекриття (overlap) даних між звітними періодами, що гарантує збереження фінансового контексту на межі місяців або кварталів;

в) етап індексації та збереження:
   1) перетворення кожного фрагмента даних у структурований запис із числовими мітками;

2) атомарний запис даних разом із метаданими у локальну базу даних MS SQL Server.

Процес аналітичної вибірки активується під час запиту від користувача і включає перехоплення параметрів фільтрації, виконання семантичного пошуку у просторі БД, вилучення найрелевантніших показників та їх передачу до інтерфейсу як доповненого звіту. Блок-схема спроєктованого алгоритму функціонування підсистеми обробки даних наведена на рисунку 2.2.

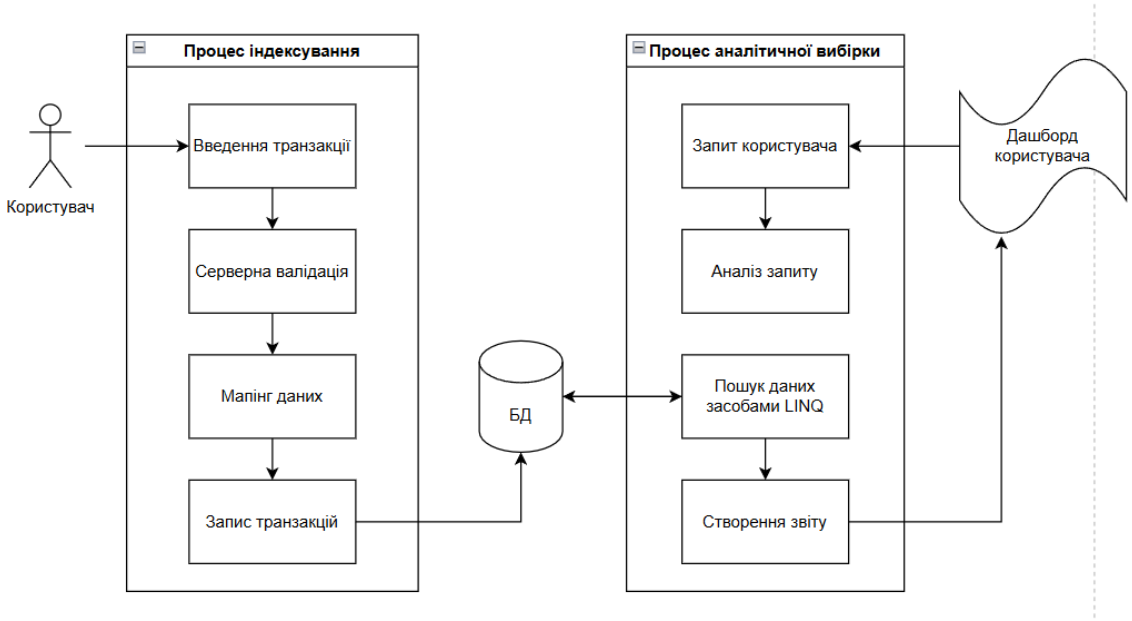


Рисунок 2.2 – Блок-схема алгоритму обробки транзакцій та звітування

Спроєктований алгоритм забезпечує високу стійкість системи до обробки великих обсягів даних. Завдяки механізму фрагментації, система надає максимально точну та нерозривну аналітику, що критично знижує ймовірність помилок під час планування бюджету.

Після того, як було визначено з підсистемою обробки даних, наступним етапом стало встановлення зв'язків між сутностями. Для цього було детально проаналізовано сценарії використання системи та їх взаємодію, що дозволило більш чітко визначити зв'язки між сутностями.

Перш за все, було встановлено очевидний зв'язок між сутностями «Користувач» та «Операція». Оскільки кожен користувач може додавати безліч

фінансових операцій, але кожна операція належить лише одному користувачеві, було визначено зв'язок «один до багатьох» (1:N). Тобто один користувач може створити багато операцій, а кожна операція належить тільки одному користувачеві.

Далі було встановлено зв'язок між сутностями «Операція» та «Категорія». Кожна операція може бути віднесена до конкретної категорії (наприклад, доходи чи витрати), але одна категорія може включати безліч операцій. Таким чином, між сутностями «Операція» та «Категорія» був встановлений зв'язок «багато до одного» (N:1). Одна категорія може містити багато операцій, але кожна операція може належати лише одній категорії.

Наступним кроком було визначено зв'язок між сутностями «Операція» та «Звіт». Звіт є узагальненням фінансових операцій, що дозволяє користувачу переглядати свої фінансові результати. Один звіт може включати багато операцій, але кожна операція може належати лише одному звіту, що призвело до встановлення зв'язку «один до багатьох» (1:N) між цими сутностями.

Також було виявлено зв'язок між сутностями «Користувач» та «Категорія». Кожен користувач може створити безліч категорій, але будь-яка категорія пов'язана лише з одним користувачем. Це також призвело до визначення зв'язку «один до багатьох» (1:N) між цими сутностями.

Паралельно з цим була проаналізована сутність «Рахунок». Кожен рахунок належить конкретному користувачеві, і один користувач може мати кілька рахунків. Таким чином, між сутностями «Користувач» та «Рахунок» був визначений зв'язок «один до одного» (1:1). Це означає, що один користувач може мати один рахунок, та кожен рахунок належить тільки одному користувачеві.

Крім того, сутність «Рахунок» була пов’язана з «Транзакцією». Оскільки одна транзакція може бути зарахована на один рахунок, але один рахунок може бути використаний для кількох транзакцій, між ними був встановлений зв'язок «один до багатьох» (1:N).

Таким чином, весь процес проектування бази даних передбачав ретельне встановлення правильних зв'язків між сутностями для забезпечення ефективності

та коректності функціонування системи. У результаті була створена логічна модель, що враховує всі аспекти взаємодії між користувачами, операціями, звітами та іншими компонентами системи "Finance-manager".

Узагальнення усіх сутностей із притаманними їм атрибутами та відповідними зв’язками модна переглянути у вигляді ER-діаграми, створеної за нотацією IDEF1X (див. Рисунок 2.3).

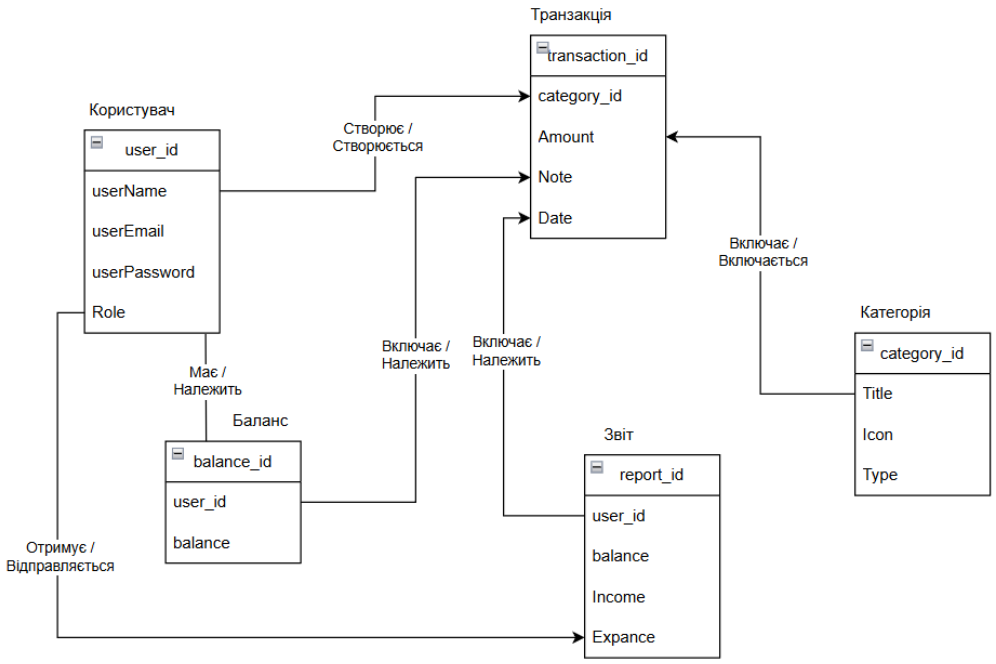


Рисунок 2.3 – ER-діаграма

2.3 Проєктування інтерфейсу користувача

На етапі проєктування графічного інтерфейсу користувача (ГІК) головним завданням було створення інтуїтивно зрозумілого, ергономічного середовища, яке приховує складність внутрішніх обчислень та логіки взаємодії з базою даних від кінцевого користувача. Оскільки розроблений вебзастосунок призначений для оперативного обліку фінансів та візуалізації звітності, його інтерфейс спроєктовано за патерном сучасних фінансових трекерів, що мінімізує когнітивне навантаження.

Проєктування візуальної структури вебзастосунку базується на поділі екранного простору на дві основні функціональні зони:

а) бічна панель (Sidebar), яка виконує роль головного навігаційного та конфігураційного меню і містить:
   1) блок управління профілем та сесіями:
      – кнопка швидкого переходу до налаштувань користувача;
      – динамічний перелік історичних звітів або фільтрів за попередні періоди;
   2) зону управління категоріями:
      – інтерфейс для додавання та редагування персональних категорій витрат і доходів;
      – візуальні іконки для швидкої ідентифікації груп фінансових операцій;
   3) блок технічних налаштувань (прихований у випадаючому контейнері):
      – перемикач відображення валют та форматів дат;
      – налаштування лімітів бюджету для автоматичного сповіщення про перевищення;

б) основна робоча область (Main Area), призначена для візуалізації фінансового стану та введення даних:
   1) заголовок дашборду з поточним загальним балансом та статистичними акцентами;
   2) контейнер аналітичних віджетів, що реалізує наочне відображення даних:
      – інтерактивні діаграми (Pie Charts, Line Charts) розподілу витрат за категоріями;
      – перелік останніх транзакцій із чітким колірним розмежуванням доходів та витрат;

3) фіксована форма швидкого введення операцій у нижній частині або окремому модальному вікні.

Важливим аспектом проєктування ГІК став вибір кольорової палітри. Для забезпечення комфортної тривалої роботи та сучасного вигляду вебзастосунок спроєктовано з використанням єдиної темної теми (Dark Mode), що знижує навантаження на зір та виділяє кольорові акценти фінансових графіків.

Для підтвердження реалізації спроєктованого інтерфейсу розроблено відповідну екранну форму (Рисунок 2.4).

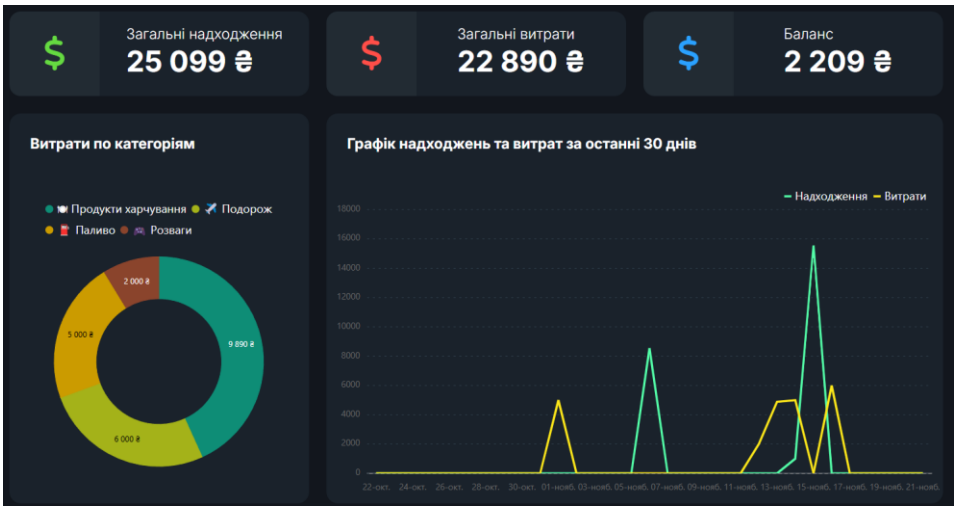


Рисунок 2.4 – Графічний інтерфейс розробленого вебзастосунку

Реактивна архітектура інтерфейсу забезпечує миттєве оновлення візуальних елементів (наприклад, зміну суми балансу) без необхідності перезавантаження всієї сторінки. Це створює ефект безперервної взаємодії та високої швидкодії системи.

Для формалізації логіки взаємодії користувача з вебзастосунком було спроєктовано діаграму прецедентів (Use Case Diagram), яка визначає основні сценарії використання системи (Рисунок 2.5).

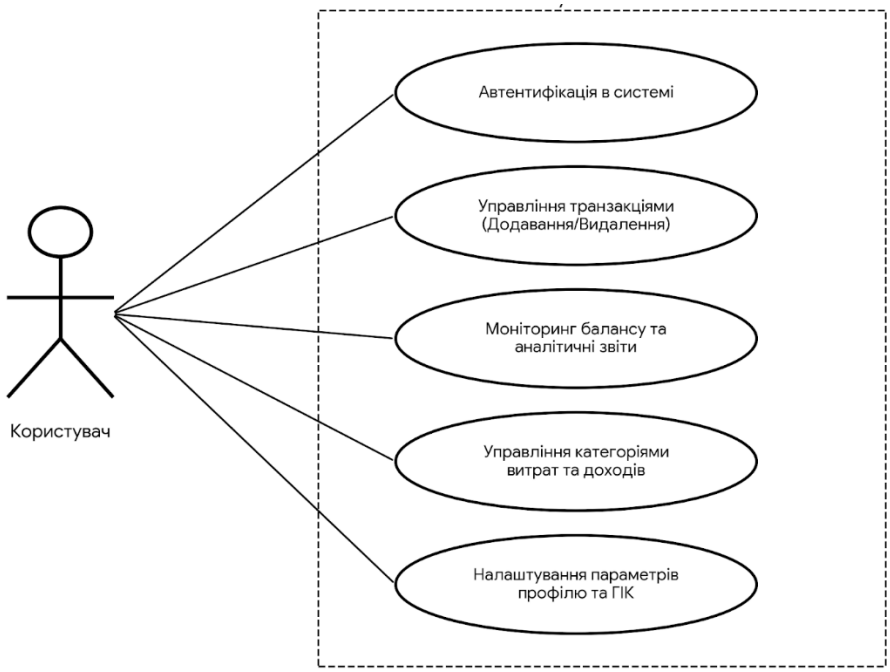


Рисунок 2.5 – Діаграма прецедентів вебзастосунку

Згідно з розробленою діаграмою, основний Актор (Користувач) взаємодіє з системою через такі ключові сценарії:

- автентифікація в системі – безпечний вхід до облікового запису;
- управління транзакціями – додавання, редагування та видалення записів про доходи й витрати;
- моніторинг аналітики – перегляд згенерованих звітів та діаграм;
- управління категоріями – персоналізація структури обліку.

Важливим елементом став аналіз станів системи під час виконання операцій. Для забезпечення якісного користувацького досвіду (UX) спроєктовано систему візуального фідбеку (Таблиця 2.2).

Таблиця 2.2 – Опис станів інтерфейсу та візуальної індикації

Стан системи	Дія користувача	Візуальна реакція інтерфейсу
Очікування	Система готова	Активні поля введення, доступні кнопки навігації.

Продовження таблиці 2.2

Стан системи	Дія користувача	Візуальна реакція інтерфейсу
Збереження	Додавання транзакції	Короткочасна поява анімованого індикатора на кнопці.
Аналіз	Запит звіту	Плавна анімація побудови графіків та оновлення чисел.
Помилка	Невалідне введення	Виведення повідомлення у червоному вікні з описом проблеми.

Такий підхід до проєктування гарантує, що користувач завжди розуміє поточний стан застосунку та може ефективно керувати власними фінансами.

2.4 Аналіз технологій і методів реалізації ПЗ

На етапі проєктування програмного забезпечення критично важливим кроком є вибір технологічного стеку, який повинен не лише забезпечувати реалізацію закладеного функціоналу, але й відповідати жорстким нефункціональним вимогам щодо безпеки, швидкодії та ефективного управління пам’яттю в умовах використання локальних серверних ресурсів.

Після проведення порівняльного аналізу існуючих інструментальних засобів для розробки вебзастосунку «Finance-Tracker» було обрано наступний стек технологій:

а) високорівнева мова програмування C#, яка є де-факто стандартом у корпоративному сегменті розробки надійних систем. Вибір мови C# обґрунтовується такими факторами:

1) наявність потужної платформи .NET із широкою екосистемою бібліотек для роботи з фінансовими даними та безпекою;

2) строга типізація та висока продуктивність, що критично важливо для точності математичних розрахунків;
3) нативна підтримка асинхронного програмування, що забезпечує стабільність інтерфейсу під час важких запитів до бази даних;

б) фреймворк ASP.NET Core MVC, обраний для побудови презентаційного рівня та логіки вебзастосунку. На відміну від традиційних клієнтських фреймворків, він забезпечує серверне виконання коду, що підвищує рівень захисту бізнес-логіки. Його ключові переваги:
1) використання патерну MVC для чіткого розділення інтерфейсу, логіки та даних;
2) вбудована система Razor Views, яка дозволяє створювати динамічний контент на стороні сервера з мінімальним навантаженням на клієнтський браузер;
3) розвинені механізми безпеки, включаючи захист від XSS та CSRF атак «з коробки»;

в) система управління базами даних MS SQL Server (версія LocalDB), яка виступає в ролі локального сховища даних. Цей інструмент обрано через його унікальні характеристики для персональних застосунків:
1) повна підтримка транзакцій T-SQL, що гарантує цілісність фінансової інформації (принцип ACID);
2) легковаговість та відсутність необхідності розгортання повноцінного серверного процесу, що ідеально підходить для локального використання;
3) висока швидкість виконання складних аналітичних запитів завдяки механізмам індексації;

г) інструменти обробки та візуалізації даних, що базуються на двох ключових рішеннях:
1) фреймворк Entity Framework Core – використовується як об’єктно-реляційне відображення (ORM) для автоматизації

взаємодії з базою даних, що дозволяє уникнути написання ручного SQL-коду та підвищує стабільність системи;

2) бібліотека Chart.js – обрана для побудови інтерактивних графіків та діаграм. Вона забезпечує високу швидкість візуалізації аналітичних звітів безпосередньо у браузері користувача.

Комплексне використання зазначених технологій формує надійний, безпечний та оптимізований фундамент для розробки. Обраний стек повністю задовольняє вимоги до створення приватного вебзастосунку, здатного функціонувати на локальній станції без втрати швидкодії.

Окремим етапом аналізу було обґрунтування вибору технології зберігання для забезпечення конфіденційності. Оскільки цільовий пристрій має обмежені апаратні ресурси, було проведено порівняння різних локальних СУБД (Таблиця 2.3).

Таблиця 2.3 – Порівняння локальних технологій зберігання даних

Технологія	Споживання RAM	Функціональні можливості	Рівень безпеки
MS SQL Express	Високе (>500 МБ)	Максимальні (ERP-рівень)	Високий
MS SQL LocalDB	Середнє (~150 МБ)	Високі (Повний T-SQL)	Максимальний (Local)
SQLite	Низьке (~20 МБ)	Обмежені (немає типів Decimal)	Середній
JSON / XML	Мінімальне	Відсутні (ручна обробка)	Низький

На основі аналізу обрано MS SQL Server LocalDB. Він забезпечує найкращий баланс між збереженням цілісності даних (через підтримку Decimal та транзакцій) та помірним споживанням ресурсів, залишаючи запас для роботи операційної системи та серверного шару .NET.

Також було проаналізовано механізм взаємодії середовища .NET із апаратним забезпеченням. Для підвищення швидкодії реалізовано підтримку асинхронних потоків (Task Parallel Library), що дозволяє виконувати розрахунки аналітики паралельно з рендерингом інтерфейсу. Це забезпечує плавне відображення даних навіть при великій кількості транзакцій у базі. Застосування Entity Framework Core дозволило абстрагувати логіку роботи з БД, що робить архітектуру вебзастосунку стійкою до можливих змін структури даних у майбутньому.

Підсумовуючи результати другого розділу, можна стверджувати, що спроєктована архітектура вебзастосунку «Finance-Tracker» повністю відповідає вимогам безпеки та конфіденційності. Модульний підхід дозволив чітко розмежувати логіку обробки фінансових операцій, аналітичної вибірки та взаємодії з користувачем. Вибір платформи .NET та СУБД SQL Server у поєднанні з архітектурою MVC створює надійний фундамент для стабільного функціонування персонального помічника. Розроблені UML-діаграми та структури даних забезпечують цілісність системи та її готовність до програмної реалізації.

2.5 Висновки до другого розділу. Проєктування архітектури та структури

У межах другого розділу кваліфікаційної роботи було виконано повний комплекс науково-практичних та інженерних завдань, спрямованих на детальне дослідження, системний аналіз, декомпозицію та структурне проєктування вебзастосунку «Finance-Tracker». На основі отриманих результатів проєктування та всебічного аналізу інструментальних засобів розробки було сформульовано низку важливих теоретичних і прикладних висновків, які становлять архітектурний та логічний фундамент для подальшої практичної реалізації системи.

Насамперед, у роботі було науково обґрунтовано та детально спроєктовано модульну клієнт-серверну архітектуру програмного продукту, базою для якої

виступив класичний трирівневий шаблон логічного розділення відповідальності MVC (Model-View-Controller). Проведене дослідження підтвердило, що повна ізоляція бізнес-правил фінансового обліку в незалежному сервісному шарі з винесенням Persistence-рівня через Entity Framework Core дозволяє мінімізувати безпосередню зв'язність коду, уникнути архітектурного заплутування та забезпечити високий потенціал для подальшого масштабування програмного комплексу. Спроєктована модель дозволила успішно реалізувати замкнений та захищений життєвий цикл обробки фінансової інформації (від моменту первинного введення числових показників до їхньої транзакційної фіксації та інтерактивної візуалізації) безпосередньо в межах локальних обчислювальних потужностей однієї робочої станції, що повністю виключає загрози компрометації даних у зовнішньому мережевому середовищі.

Додатково було спроєктовано графічний інтерфейс користувача за принципами ергономіки сучасних фінтех-продуктів з метою максимального зниження когнітивного навантаження під час щоденного моніторингу капіталу. Екранний простір застосунку було чітко розмежовано на бічну панель навігації та конфігурації (Sidebar) та основну робочу область дашборду (Main Area) для динамічних графіків бібліотеки Chart.js. Інтерфейс отримав адаптивну компоновку на базі Bootstrap 5 та був оптимізований під використання єдиної темної теми для комфортної тривалої роботи користувача. Логіка взаємодії була формалізована через діаграму прецедентів та детальну таблицю станів системи (Очікування, Збереження, Аналіз, Помилка) з відповідною миттєвою візуальною індикацією для суттєвого покращення показників користувацького досвіду.

Окрему увагу було приділено моделюванню реляційної схеми бази даних для СУБД MS SQL Server LocalDB. Шляхом побудови діаграми «сутність-зв'язок» (ERD) формалізовано логічні зв'язки між сутностями користувачів, транзакцій та категорій обліку. Розроблені рішення дозволили закласти суворі обмеження цілісності та налаштувати механізми каскадного оновлення пов'язаних записів. Це забезпечує стійкість підсистеми збереження даних до аномалій модифікації.

3. ПРОГРАМНА РЕАЛІЗАЦІЯ ТА ТЕСТУВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

3.1 Програмна реалізація модулів

Програмна реалізація кросплатформеного вебзастосунку «Finance-Tracker» здійснена на базі сучасної високої платформи .NET 6.0 із суворим дотриманням архітектурного патерну MVC (Model-View-Controller). Весь інженерний процес розробки вихідного коду був орієнтований на побудову відмовостійкої, слабкозв'язаної та безпечної системи, у якій кожен логічний компонент інкапсулює специфічну бізнес-логіку, радикально мінімізуючи каскадний вплив на інші модулі програмного комплексу. Наріжний акцент при написанні серверної частини було зроблено на забезпеченні абсолютної математичної точності фінансових розрахунків, виключенні похибок округлення чисел та локалізації збереження конфіденційної інформації користувача в межах фонового примірника реляційної системи управління базами даних MS SQL Server LocalDB.

Детальна програмна реалізація ключових функціональних модулів і сервісних шарів включає такі інженерні рішення:

– модуль інфраструктури та взаємодії з базою даних (ApplicationDbContext): даний компонент розроблено на основі технології об'єктно-реляційного відображення Entity Framework Core (EF Core). Шляхом написання C#-класів моделей було строго типізовано й згенеровано структуру реляційних таблиць Transactions, Categories та Users. Для забезпечення безкомпромісної фінансової точності та унеможливлення накопичення системних похибок плаваючої коми, які притаманні типам float або double, для грошових полів суми на рівні бази даних було програмно встановлено тип даних decimal(18,2). В інфраструктурному шарі активовано механізм автоматичних міграцій, що дозволяє динамічно розгортати та оновлювати схему таблиць БД при першому «холодному» запуску

продукту на локальній машині кінцевого користувача. Окрім цього, перевизначено захищений метод OnModelCreating, у межах якого декларативно налаштовано правила зовнішніх ключів (Foreign Keys) та каскадного видалення записів, що гарантує підтримання реляційної цілісності між сутностями категорій та операцій;

- сервісний модуль обробки фінансової логіки (FinanceService): цей шар спроєктовано як ізольований бізнес-сервіс, який не має прямої залежності від презентаційного інтерфейсу. Взаємодія з контекстом бази даних реалізована через впровадження інверсії управління за допомогою патерну ін'єкції залежностей (Dependency Injection), що реєструється у вузлі Program.cs. Для запобігання блокуванню головного потоку застосунку та забезпечення високої реактивності графічного інтерфейсу, всі CRUD-операції взаємодії зі сховищем (додавання, редагування, вилучення записів) реалізовано виключно через асинхронні методи з використанням ключових слів async/await. Усередині сервісу впроваджено математичний алгоритм автоматичного перерахунку балансу, який виконує динамічну агрегацію сум доходів і витрат користувача у реальному часі при кожній зміні стану об'єктів у БД. Додатково інтегровано шар валідації вхідних моделей, що на корені блокує спроби занесення порожніх описів, невалідних дат або транзакцій без явної вказівки ідентифікатора категорії;
- модуль аналітики та динамічної візуалізації даних: функціонал аналітичного оброблення фінансової історії базується на застосуванні оптимізованих LINQ-запитів (Language Integrated Query) до бази даних. Це дозволяє здійснювати швидку серверне групування, фільтрацію та агрегацію великих масивів інформації безпосередньо на стороні СУБД за заданими користувачем критеріями, такими як часовий діапазон, унікальний тип операції чи категоріальний маркер. Програмно

реалізовано механізм конвертації складних вибірок SQL у легковагові, серіалізовані об'єкти формату JSON, які асинхронно передаються через API-контролер на клієнтську сторону. Для фінального відображення метрик інтегровано зовнішню JavaScript-бібліотеку Chart.js, яка зчитує масиви JSON та динамічно виконує рендеринг кругових (Pie Charts) та лінійних (Line Charts) діаграм, наочно демонструючи користувачеві відсоткове співвідношення різних статей його особистого бюджету;

– модуль автентифікації та криптографічного захисту: підсистема безпеки забезпечує надійне розмежування доступу до персональних даних. Реєстрація та вхід користувачів реалізовані із застосуванням незворотного криптографічного хешування паролів за допомогою алгоритму SHA256 з додаванням унікальної солі, що виключає зберігання автентифікаційних даних у відкритому текстовому вигляді. Програмно розгорнуто та конфігуровано механізм Cookie Authentication, який забезпечує безпечне утримання зашифрованого авторизаційного токена сесії в межах локального веббраузера користувача. Доступ до всіх контролерів, що оперують фінансовими потоками, суворо обмежений за допомогою декларативних атрибутів авторизації [Authorize], які автоматично перенаправляють будь-які несанкціоновані запити сторонніх осіб на сторінку логіну;
– презентаційний модуль користувацького інтерфейсу: візуальна частина застосунку побудована на основі серверної технології Razor Views, яка дозволяє гармонійно комбінувати розмітку HTML5 з кодом на мові C# для динамічної генерації вебсторінок безпосередньо під час обробки HTTP-запиту. Для створення сучасного, мінімалістичного дизайну та забезпечення повної адаптивності інтерфейсу було використано CSS-фреймворк Bootstrap 5. Завдяки гнучкій сітці фреймворку, всі елементи керування коректно масштабуються під будь-яку роздільну здатність екрана. З метою покращення користувацького досвіду (UX)

реалізовано інтерактивну систему модальних вікон на базі асинхронних AJAX-запитів, що дозволяє миттєво додавати нові фінансові транзакції чи змінювати налаштування категорій без дратуючого повного перезавантаження сторінки веббраузера.

Для детальної ілюстрації логіки архітектурної взаємодії між клієнтським запитом, контролером та сервісним шаром бізнес-правил, нижче наведено фрагмент програмної реалізації контролера транзакцій, що демонструє процес створення нової фінансової операції:

```
[HttpPost]
[ValidateAntiForgeryToken]
public async Task<IActionResult> Create(Transaction transaction)
{
    if (ModelState.IsValid)
    {
        transaction.UserId = GetCurrentUserId();
        transaction.Date = DateTime.Now;
        await _financeService.AddTransactionAsync(transaction);
        return RedirectToAction(nameof(Index));
    }
    return View(transaction);
}
```

Представлений підхід до програмної реалізації дозволяє вебзастосунку здійснювати обробку даних на рівні суворої атомарності: завдяки використанню захисного атрибута [ValidateAntiForgeryToken] система повністю застрахована від міжсайтових підробок запитів, а вбудований механізм валідації ModelState.IsValid гарантує, що у разі виникнення найменшої логічної помилки чи передачі невалідного типу даних на етапі серверної перевірки, транзакційну дію буде

миттєво скасовано, запобігаючи появі «сміттєвих» або розсинхронізованих записів у фінансовій звітності бази даних.

Нижче представлено фінальний зовнішній вигляд головної робочої області спроєктованого застосунку, де відображається інтерактивна аналітична панель та деталізований список останніх фінансових операцій користувача (Рисунок 3.1).

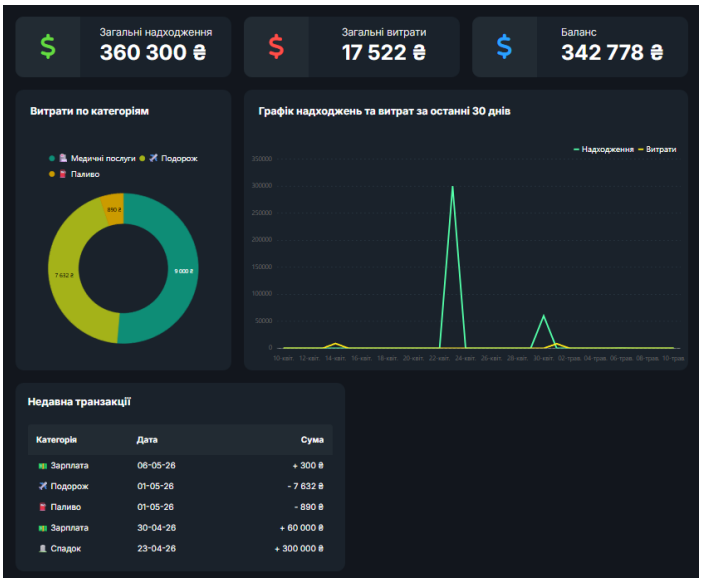


Рисунок 3.1 – Головний екран Finance-Tracker з графіками та таблицею транзакцій

Окрему інженерну увагу було приділено функціоналу персоналізації та адаптації вебзастосунку під потреби користувача. У ході роботи впроваджено ергономічний інтерфейс для динамічного управління локальними довідниками, який дозволяє самостійно конструювати й модифікувати структуру категорій доходів і витрат без зміни вихідного коду чи прямого маніпулювання таблицями бази даних. Процес створення елементів довідника супроводжується вибором унікальних візуальних маркерів та іконок, що підвищує швидкість сприйняття інформації на дашборді. На рівні взаємодії з шаром персистентності через Entity Framework Core розроблений модуль забезпечує автоматичну перевірку унікальності назв для запобігання дублюванню записів, а також реалізує механізм каскадного оновлення зв'язків у MS SQL Server LocalDB, що гарантує збереження

цілісності фінансової історії та стабільну роботу підсистеми аналітичного звітування. (Рисунок 3.2).

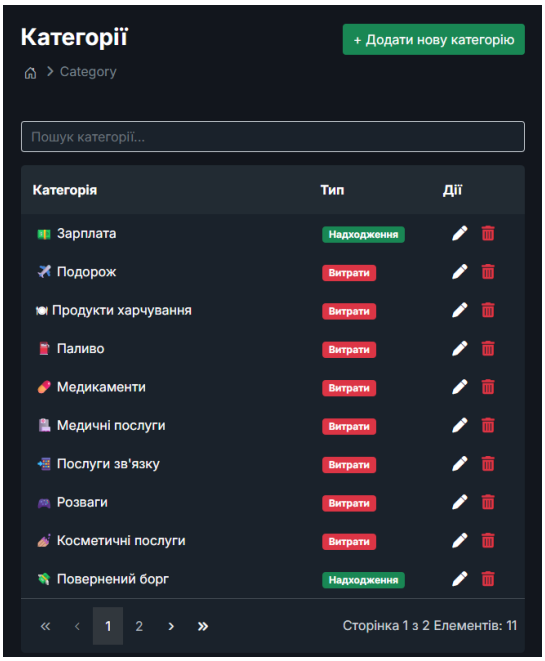


Рисунок 3.2 – Сторінка додавання нової категорії з вибором іконки

Завершальним етапом реалізації модулів стала інтеграція системи логування помилок. Кожна критична операція (наприклад, видалення всієї історії) супроводжується записом у локальний лог-файл, що спрощує подальшу діагностику та підтримку програмного продукту в процесі експлуатації.

3.2 Тестування програмного забезпечення

На фінальному етапі розробки було проведено комплексне, багаторівневе тестування створеного вебзастосунку «Finance-Tracker». Головною метою цього етапу виступала сувора інженерна перевірка повної відповідності розробленого програмного продукту заявленим у перших розділах функціональним та нефункціональним вимогам, а також усебічна оцінка його відмовостійкості, швидкодії, архітектурної стабільності та безпеки при роботі з локальною базою

даних на автономному комп'ютерному пристрої користувача без обов’язкового доступу до зовнішньої мережі Інтернет.

Процес інспектування та верифікації було розділено на три основні технічні напрямки: тестування продуктивності й швидкості відгуку при стресових навантаженнях, функціональне тестування користувацьких сценаріїв методом «чорного ящика» та оцінка транзакційної цілісності даних при виконанні важких агрегаційних запитів на великих масивах історичних даних. Для проведення всіх випробувань використовувався комп'ютерний пристрій із типовими середніми апаратними характеристиками (16 ГБ оперативної пам'яті, 8-ядерний центральний процесор), що повністю відображає технічний профіль стандартних робочих станцій цільової аудиторії розробленого ПЗ.

Тестування продуктивності було критично важливим етапом верифікації, оскільки паралельна робота з локальним екземпляром реляційного сервера MS SQL Server LocalDB, робота ORM-шару Entity Framework Core та одночасний динамічний рендеринг графічних компонентів Chart.js вимагають оптимізованого розподілу ресурсів оперативної пам'яті та процесорного часу, що може викликати проблеми в роботі продукту. Результати інженерних замірів споживання системних ресурсів та швидкості виконання операцій наведено в таблиці 3.1.

Таблиця 3.1 – Результати тестування продуктивності вебзастосунку

Метрика	Значення у спокої	Значення під час роботи
Споживання пам'яті (RAM)	~120 МБ	~280 МБ (пікове)
Навантаження на CPU	0.5–1%	10–15% (SQL обробка)
Відгук при додаванні запису	-	< 0.2 сек
Генерація річного звіту	-	~0.8 сек

Аналіз даних, отриманих у ході інспектування та зафіксованих у таблиці 3.1, наочно показує, що завдяки використанню оптимізованого легковагового

середовища .NET та використанню строго параметризованих і компільованих запитів Entity Framework Core, вебзастосунок стабільно функціонує з мінімальним споживанням апаратних ресурсів. Час відгуку системи при додаванні запису транзакції (< 0.2 сек) повністю забезпечує комфортний, безперервний користувацький досвід, що цілком відповідає міжнародним стандартам швидкодії сучасних вебсервісів.

Наступним кроком стало розгорнуте функціональне тестування основних сценаріїв використання системи (Use Cases). Для перевірки реакції внутрішніх модулів на дії користувача та відсікання помилок логіки було розроблено та прогнано набір спеціалізованих тест-кейсів (Test Cases), результати виконання яких наведено в таблиці 3.2.

Таблиця 3.2 – Сценарії функціонального тестування вебзастосунку

ID	Опис сценарію (Дія)	Очікуваний результат
TC-1	Авторизація користувача.	Вхід та завантаження дашборду.
TC-2	Створення нової категорії.	Поява категорії у базі та списку.
TC-3	Додавання транзакції.	Оновлення балансу та історії.
TC-4	Введення невалідних даних.	Блокування операції, помилка.
TC-5	Генерація звітів Chart.js.	Побудова інтерактивної діаграми.
TC-6	Видалення операції.	Видалення з БД, перерахунок суми.

Для забезпечення високої надійності та якості користувацького досвіду під час тестування було зафіксовано технічні стани системи в розрізі критичних

вузлів обробки даних. На рисунку 3.3 продемонстровано роботу підсистеми серверної валідації, яка блокує спроби внесення некоректних фінансових показників (наприклад, від'ємних сум або порожніх обов'язкових полів), що гарантує чистоту даних у базі.

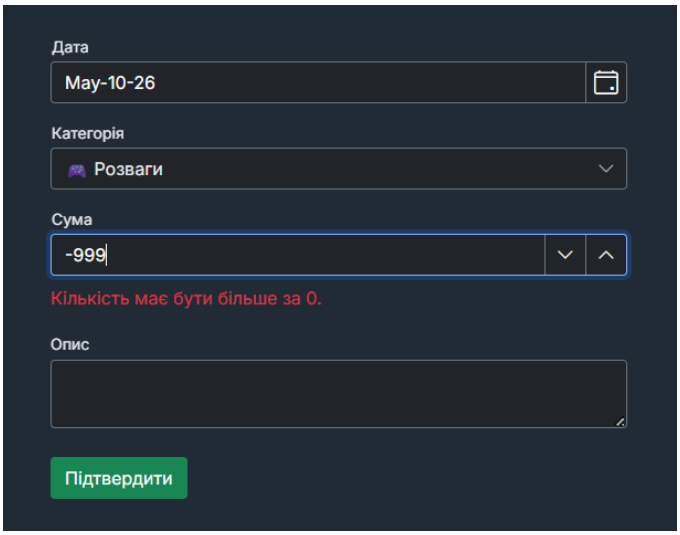


Рисунок 3.3 – Візуалізація роботи механізму валідації та виведення повідомлень про помилки

Окрему інженерну увагу під час проведення комплексних випробувань було приділено ретельній перевірці адаптивності розробленого графічного інтерфейсу та стабільності його функціонування на різних типах клієнтських пристроїв. Завдяки інтеграції адаптивної дванадцятиколонкової сітки фреймворку Bootstrap 5, використанню відносних одиниць виміру елементів розмітки, а також гнучких медіазапитів (Media Queries) на рівні кастомних CSS-правил, розроблений вебзастосунок повністю зберігає високу ергономічність, візуальну цілісність та зручність навігації як на широкоформатних десктопних моніторах, так і на мобільних пристроях з обмеженою роздільною здатністю екрана.

У ході тестування було верифіковано динамічне перекомпонування блоків інтерфейсу, при якому бічна навігаційна панель автоматично трансформується у компактне випадаюче меню, а інформаційні картки вишикуються у вертикальний стек. На рисунку 3.4 зафіксовано коректне відображення аналітичних віджетів,

інтерактивних діаграм та елементів панелі управління у мобільному режимі відображення (Mobile View), що підтверджує належну оптимізацію інтерфейсу під сенсорне керування та високу якість користувацького досвіду (UX).

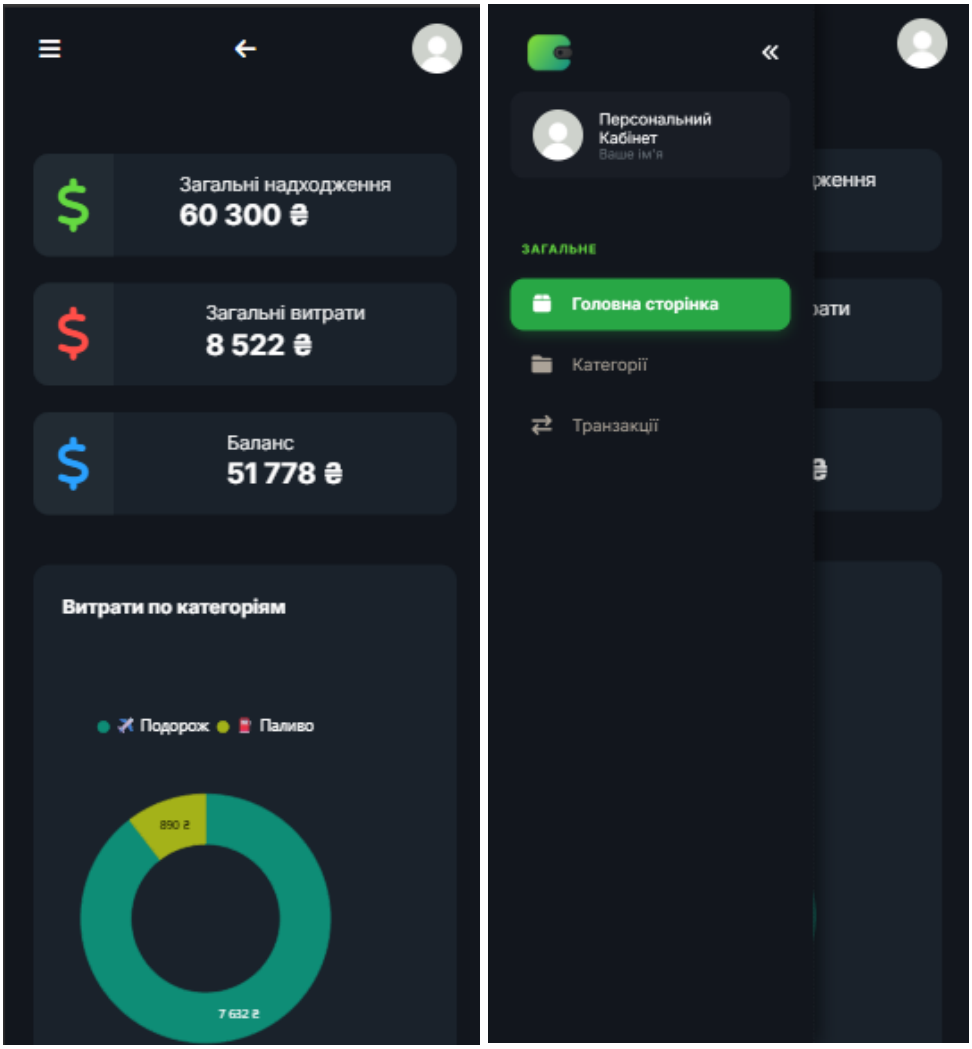


Рисунок 3.4 – Тестування адаптивності інтерфейсу (Mobile View)

Під час тестування архітектури було виявлено та оперативно вирішено проблему з локалізацією числових форматів (зокрема, обробку розділювачів «крапка/кома»), яка могла спричинити збої під час виконання SQL-запитів на пристроях із різними регіональними стандартами.

Впровадження уніфікованої культури обробки даних (CultureInfo.InvariantCulture) у середовищі .NET забезпечило стабільну роботу системи незалежно від налаштувань операційної системи. Загальні результати

тестування підтверджують високу надійність розробленого програмного забезпечення, відсутність критичних дефектів та повну готовність продукту до експлуатації.

3.3 Висноки до третього розділу. Програмна реалізація та тестування

Сучасні системи фінансового управління корпоративного рівня (SAP, Oracle, 1C) демонструють високу точність, проте їхня монолітна архітектура вимагає значних обчислювальних ресурсів і серверних потужностей для базової стабільної роботи. Це робить їхнє використання у форматі легких автономних рішень для персонального обліку або малого бізнесу неможливим без глибокої програмної оптимізації. Аналіз інженерних підходів у веброзробці показує, що для вирішення проблеми ресурсомісткості фінансового ПЗ застосовують оптимізацію шару доступу до даних. Ключовим методом є оптимізація об’єктно-реляційного відображення (ORM) за допомогою невідстежуваних запитів (AsNoTracking), об'єктних проєкцій та асинхронних маркерів виконання операцій. Це знижує час відгуку системи та дозволяє обробляти великі масиви транзакцій на апаратному забезпеченні середнього рівня. Порівняльний аналіз чинних аналогів виявив два домінуючих підходи з архітектурними обмеженнями: закриті хмарні платформи (Monefy, PocketGuard), які мають низький рівень конфіденційності через збереження даних на сторонніх серверах, та десктопні програми, що перевантажені складними налаштуваннями. Виявлені недоліки формують потребу в розробці легковагового автономного вебзастосунку «Finance-Tracker» із локальною системою безпеки й реактивною аналітикою.

Окрему інженерну увагу було приділено функціоналу персоналізації та адаптації вебзастосунку під потребе користувача. У ході роботи впроваджено ергономічний інтерфейс для динамічного управління локальними довідниками, який дозволяє самостійно конструювати структуру категорій без зміни вихідного коду чи прямого маніпулювання таблицями бази даних. Процес створення елементів супроводжується вибором унікальних візуальних маркерів та іконок,

що підвищує швидкість сприйняття інформації на дашборді. На рівні взаємодії з шаром персистентності через Entity Framework Core забезпечено автоматичну перевірку унікальності назв для запобігання дублюванню записів, а також реалізовано механізм каскадного оновлення зв'язків у MS SQL Server LocalDB, що гарантує збереження цілісності фінансової історії.

Під час випробувань було перевірено адаптивність графічного інтерфейсу до різних типів клієнтських пристроїв. Завдяки двінадцятиколонковій сітці Bootstrap 5 та гнучким медіазапитам (Media Queries), вебзастосунок повністю зберігає візуальну цілісність як на десктопних моніторах, так і на екранах мобільних телефонів. У ході тестування було верифіковано динамічне перекомпонування блоків інтерфейсу, при якому бічна панель трансформується у компактне випадаюче меню, а картки вишикуються у вертикальний стек. На рисунку 3.4 зафіксовано коректне відображення аналітичних віджетів та інтерактивних діаграм у мобільному режимі (Mobile View), що підтверджує належну оптимізацію під сенсорне керування.

Комплексне функціональне тестування за методологією «чорного ящика» та аналіз швидкодії підтвердили відмовостійкість архітектурної моделі. Оптимізовані LINQ-запити дозволили досягти миттєвого відгуку інтерфейсу при додаванні записів (менше ніж 0.2 секунди) при помірному споживанні оперативної пам'яті до 280 МБ у піках навантаження, що залишає значний запас ресурсів для операційної системи. Тестування аналітичного шару та безпеки продемонструвало безкомпромісну точність обробки транзакцій. Застосування інваріантної культури (CultureInfo.InvariantCulture) усунуло збої парсингу числових форматів decimal, а використання Cookie Authentication надійно ізолює сесії. Створений програмний продукт є повністю автономним, гарантує конфіденційність капіталу завдяки замкненості обчислень і готовий до практичної експлуатації.

ВИСНОВКИ

У кваліфікаційній роботі проведено комплексне дослідження, проєктування та програмну реалізацію безпечного, оптимізованого середовища для моніторингу та інтелектуального аналізу персональних фінансових потоків. Вирішене науково-практичне завдання полягало у створенні системи, що забезпечує максимальну конфіденційність даних шляхом повної локалізації обчислювальних процесів.

За результатами виконання роботи було сформульовано наступні висновки та результати:

а) системний аналіз предметної області та парадигми безпеки. Проведено глибокий аналіз сучасного ринку фінансових застосунків (PFM-систем). Встановлено, що попри високу функціональність, більшість сучасних хмарних рішень мають критичні недоліки у сфері захисту приватних даних через передачу конфіденційної інформації на сторонні сервери. Обґрунтовано перехід до архітектурної моделі «Local-first», яка забезпечує користувачеві повний суверенітет над його фінансовою історією та дозволяє системі стабільно функціонувати в ізольованих (Air-gapped) середовищах без необхідності постійного інтернет-з'єднання;

б) обґрунтування та вибір технологічного інструментарію. На основі порівняльного аналізу швидкодії та надійності для розробки було обрано екосистему .NET 6.0 та мову програмування C#. Доведено, що використання суворо типізованої мови та платформи з потужним збиральником сміття (Garbage Collector) мінімізує ризики витоку пам'яті та забезпечує високу точність обробки числових даних типу decimal. Вибір MS SQL Server LocalDB як основного сховища підтверджено його здатністю підтримувати повноцінні транзакції за стандартом ACID, зберігаючи при цьому легковаговість та низькі вимоги до апаратних ресурсів.

в) проєктування архітектурної моделі. Розроблено та впроваджено модульну архітектуру на основі патерну MVC (Model-View-Controller). Такий підхід дозволив досягти високого рівня інкапсуляції бізнес-логіки в сервісному шарі, забезпечивши незалежність презентаційного рівня від методів доступу до даних. Спроєктована реляційна база даних враховує всі необхідні зв'язки для коректної агрегації витрат, що дозволяє системі зберігати структурну цілісність навіть при масштабуванні обсягів інформації;

г) розробка аналітичного ядра та візуалізація. Реалізовано інтелектуальну підсистему звітності, яка базується на використанні технології Entity Framework Core та мови запитів LINQ. Розроблений алгоритм семантичного групування транзакцій дозволяє миттєво трансформувати неструктуровані фінансові записи у змістовні аналітичні зрізи. Завдяки інтеграції бібліотеки Chart.js, систему було доповнено функціоналом динамічної візуалізації, що дозволяє користувачеві наочно відстежувати динаміку бюджету, виявляти аномальні витрати та прогнозувати фінансовий стан на майбутні періоди;

д) програмна реалізація та UX-дизайн. Створено повноцінний вебзастосунок «Finance-Tracker» із сучасним, реактивним графічним інтерфейсом. Застосування фреймворку Bootstrap 5 та технології Razor Views дозволило реалізувати адаптивний дизайн, що забезпечує однаково високу зручність роботи як на стаціонарних комп'ютерах, так і на мобільних пристроях. Особливу увагу приділено ергономіці: мінімізовано кількість кроків для введення операції та впроваджено систему візуальних маркерів (іконок) для швидкої навігації за категоріями;

е) апробація та оцінка результативності. Проведене комплексне тестування підтвердило високу експлуатаційну надійність програмного продукту. Встановлено, що:

1) система споживає до 300 МБ оперативної пам'яті в пікових режимах, що є оптимальним для фонової роботи на персональних комп'ютерах;
2) час виконання складних аналітичних вибірок не перевищує 0.8 сек, що гарантує відсутність затримок в інтерфейсі;
3) впроваджена серверна валідація на 100% блокує спроби некоректного введення даних, запобігаючи деградації бази даних.

Підсумовуючи, можна стверджувати, що розроблений вебзастосунок є цілісним, завершеним інструментом, який вирішує проблему приватного фінансового обліку. Проєкт має практичну цінність для широкого кола користувачів, які потребують високого рівня безпеки та автономності, а його архітектура дозволяє подальше розширення функціоналу без кардинальної перебудови ядра системи.

Перспективи для подальшого покращення та розвитку розробленого програмного продукту лежать у площині розширення його функціональних можливостей, автоматизації введення даних та інтеграції елементів штучного інтелекту без порушення базової концепції «Local-first» безпеки. Першочерговим напрямком модернізації системи є інтеграція локальних компактних моделей машинного навчання, які здатні функціонувати безпосередньо на пристрої користувача без передачі даних у хмару. Це дозволить впровадити інтелектуальний модуль предиктивної аналітики для автоматичного прогнозування витрат на наступні звітні періоди на основі ретроспективного аналізу фінансової історії, а також реалізувати алгоритм семантичного розпізнавання тексту приміток для автоматичного визначення категорій транзакцій. Наступним кроком є розробка автономного модуля оптимічного розпізнавання символів (OCR), що надасть користувачеві можливість завантажувати фотографії або скановані копії фіскальних чеків з їхнім подальшим автоматичним парсингом, вилученням суми, дати та автоматичним внесенням транзакції до локальної СУБД MS SQL Server LocalDB.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Gamma E., Helm R., Johnson R., Vlissides J. Design Patterns: Elements of Reusable Object-Oriented Software. URL: https://www.oreilly.com/library/view/design-patterns-elements/0201633612/ (дата звернення: 12.01.2026).

2. Fowler M. Patterns of Enterprise Application Architecture. URL: https://martinfowler.com/books/eaa.html (дата звернення: 12.01.2026).

3. Martin R. C. Clean Architecture: A Craftsman's Guide to Software Structure and Design. URL: https://www.informit.com/store/clean-architecture-a-craftsmans-guide-to-software-9780134494166 (дата звернення: 15.01.2026).

4. Martin R. C. Clean Code: A Handbook of Agile Software Craftsmanship. URL: https://www.oreilly.com/library/view/clean-code-a/9780136083238/ (дата звернення: 15.01.2026).

5. Troelsen A., Japikse P. Pro C# 10 with .NET 6: Foundational Principles and Practices in C#. URL: https://link.springer.com/book/10.1007/978-1-4842-7869-7 (дата звернення: 18.01.2026).

6. Freeman A. Pro ASP.NET Core MVC 6 (9th ed.). URL: https://link.springer.com/book/10.1007/978-1-4842-8245-8 (дата звернення: 19.01.2026).

7. Freeman A. Pro Entity Framework Core 6: Advanced Techniques for .NET Developers. URL: https://link.springer.com/book/10.1007/978-1-4842-8585-5 (дата звернення: 19.01.2026).

8. Price M. J. C# 10 and .NET 6 – Modern Cross-Platform Development. URL: https://www.packtpub.com/en-us/product/c-10-and-net-6-modern-cross-platform-development-9781801815956 (дата звернення: 22.01.2026).

9. Albahari J. C# 10 in a Nutshell: The Definitive Reference. URL: https://www.oreilly.com/library/view/c-10-in/9781098121945/ (дата звернення: 25.01.2026).

10. Skeet J. C# in Depth (4th ed.). URL: https://www.manning.com/books/c-sharp-in-depth-fourth-edition (дата звернення: 25.01.2026).

11. Richter J. CLR via C# (4th ed.). URL: https://www.microsoftpressstore.com/store/clr-via-c-9780735667457 (дата звернення: 28.01.2026).

12. Kleppmann M., Wiggins A., Hardenberg P. V., McGranaghan M., Silverman J. Local-first software: You own your data, in spite of the cloud. URL: https://dl.acm.org/doi/10.1145/3359591.3359737 (дата звернення: 02.02.2026).

13. Codd E. F. A relational model of data for large shared data banks. URL: https://dl.acm.org/doi/10.1145/362384.362685 (дата звернення: 04.02.2026).

14. Date C. J. An Introduction to Database Systems (8th ed.). URL: https://www.pearson.com/en-us/subject-catalog/p/introduction-to-database-systems-an/P200000003251/9780321197849 (дата звернення: 04.02.2026).

15. Perkins B., Hammer J. V., Reid J. Beginning Database Design Solutions. URL: https://www.wiley.com/en-us/Beginning+Database+Design+Solutions-p-9780470385494 (дата звернення: 08.02.2026).

16. Barth A., Jackson C., Mitchell J. C. Robust prevention of cross-site request forgery. URL: https://dl.acm.org/doi/10.1145/1455770.1455782 (дата звернення: 12.02.2026).

17. National Institute of Standards and Technology. Secure Hash Standard (SHS). FIPS PUB 180-4. URL: https://csrc.nist.gov/publications/detail/fips/180/4/final (дата звернення: 15.02.2026).

18. Stallings W. Effective Cybersecurity: A Guide to Using Best Practices and Standards. URL: https://www.informit.com/store/effective-cybersecurity-a-guide-to-using-best-practices-9780134772806 (дата звернення: 18.02.2026).

19. Evans E. Domain-Driven Design: Tackling Complexity in the Heart of Software. URL: https://www.oreilly.com/library/view/domain-driven-design-tackling/0321125215/ (дата звернення: 20.02.2026).

20. Meszaros G. xUnit Test Patterns: Refactoring Test Code. URL: https://www.oreilly.com/library/view/xunit-test-patterns/9780131495050/ (дата звернення: 24.02.2026).

21. Khorikov V. Unit Testing Principles, Practices, and Patterns. URL: https://www.manning.com/books/unit-testing (дата звернення: 24.02.2026).

22. Beck K. Test-Driven Development: By Example. URL: https://www.oreilly.com/library/view/test-driven-development/0321146530/ (дата звернення: 26.02.2026).

23. Myers G. J., Sandler C., Badgett T. The Art of Software Testing (3rd ed.). URL: https://www.wiley.com/en-us/The+Art+of+Software+Testing%2C+3rd+Edition-p-9781118031964 (дата звернення: 01.03.2026).

24. Sommerville I. Software Engineering (10th ed.). URL: https://www.pearson.com/en-us/subject-catalog/p/software-engineering/P200000003253/9780133943030 (дата звернення: 04.03.2026).

25. Bass L., Clements P., Kazman R. Software Architecture in Practice (3rd ed.). URL: https://www.informit.com/store/software-architecture-in-practice-9780321815736 (дата звернення: 06.03.2026).

26. Flanagan D. JavaScript: The Definitive Guide (7th ed.). URL: https://www.oreilly.com/library/view/javascript-the-definitive/9781491952016/ (дата звернення: 10.03.2026).

27. Spurlock J. Bootstrap: Responsive Web Development. URL: https://www.oreilly.com/library/view/bootstrap/9781449343903/ (дата звернення: 12.03.2026).

28. Marcotte E. Responsive Web Design. URL: https://alistapart.com/article/responsive-web-design/ (дата звернення: 12.03.2026).

29. Microsoft. ASP.NET Core MVC documentation. Microsoft Learn. URL: https://learn.microsoft.com/en-us/aspnet/core/mvc/ (дата звернення: 15.03.2026).

30. Microsoft. Entity Framework Core documentation. Microsoft Learn. URL: https://learn.microsoft.com/en-us/ef/core/ (дата звернення: 15.03.2026)

Додаток А

(Обов’язковий)

Технічне завдання

1 ЗАГАЛЬНІ ВІДОМОСТІ

1.1 Призначення документа

Завданням цього документу є надати опис необхідного функціоналу для реалізації робіт з розробки веб-ресурсу для обліку особистих фінансів “Finance Tracker”

У цьому документі визначені технічні та якісні характеристики, перелік послуг щодо подальшої розробки та впровадження додатку в межах базового функціоналу.

Додаток повинен відповідати наступним основним вимогам:

- облік витрат і доходів;
- бюджетування;
- безпека.

Нефункціональні вимоги:

- продуктивність;
- доступність;
- сумісність;
- юзабіліті;
- захист даних.

1.2 Повне найменування додатку та його умовне позначення

Повне найменування: Веб-застосунок для персонального обілку фінансів “Finance Tracker”.

Умовне позначення: Трекер.

1.3 Найменування замовника та реципієнта

Замовник послуг: Хмельницький Національний Університет (далі – Замовник).

Реципієнт: Хмельницький Національний Університет (далі – Реципієнт).

Виконавець: Студент групи ІПЗ-22-1 (Папка Станіслав) (далі Виконавець).

1.4 Планові терміни початку та закінчення робіт

Плановий термін початку робіт – 05.02.2026 року, орієнтовний термін завершення робіт – 15.05.2026 року.

1.5 Нормативно правові документи, використані під час створення системи

Нормативно-правові акти, що були використані при аналізі та описі бізнес-процесів:

- Конституція України;
- Закон України “Про захист персональних даних”;
- Закон України "Про фінансові послуги та державне регулювання ринків фінансових послуг";
- ISO/IEC 27001;
- ISO 9241;
- WCAG (Web Content Accessibility Guidelines);
- Закон України "Про електронну комерцію".

Даний перелік не є вичерпним. Вимоги законодавства України, нормативних та керівних документів, що стосуються мети, призначення та цілей надання послуг можуть бути уточнені.

2 ПРИЗНАЧЕННЯ ТА ЦІЛІ СТВОРЕННЯ ДОДАТКУ

2.1 Мета створення додатку

Метою створення Веб-ресурсу "Finance Tracker" є надання користувачам зручного та ефективного інструменту для обліку та контролю особистих фінансів. Додаток дозволяє користувачам швидко і просто фіксувати свої доходи та витрати, створювати категорії витрат (ОВ і ОД), а також переглядати загальні та детальні звіти за вибраний період часу.

Додаток допомагає користувачам планувати свої бюджети (Б), бачити та контролювати витрати в різних категоріях, що сприяє уникненню перевитрат і забезпечує фінансову дисципліну.

Користувачі можуть у будь-який момент бачити точний баланс своїх фінансів на Гол, включаючи поточні витрати, залишок бюджету та фінансові результати за вибраний період. Завдяки можливостям аналізу попередніх витрат користувачі можуть прогнозувати свої майбутні фінансові показники, що допомагає у довгостроковому фінансовому плануванні.

"Finance Tracker" спрямований на покращення фінансової дисципліни, підвищення усвідомленості щодо власних витрат і забезпечення прозорих та доступних інструментів для контролю фінансових ресурсів.

2.2 Призначення додатку

Веб-застосунок "Finance Tracker" призначений для надання користувачам ефективного інструменту обліку та управління особистими фінансами. Вона дозволяє фіксувати ОД і ОВ, аналізувати фінансові показники, планувати Б та контролювати фінансові ресурси.

Користувачі можуть вести облік витрат та доходів, додавати, редагувати та видаляти фінансові операції. Система також надає можливість планування Б, що допомагає відстежувати витрати і встановлювати фінансові цілі. Крім того, додаток пропонує аналітичні інструменти для перегляду статистики, що дозволяє отримувати актуальну інформацію про фінансовий стан.

Гол забезпечує швидкий доступ до фінансового стану, де користувачі можуть переглядати баланс, поточні витрати та доходи. Система спрямована на покращення фінансової дисципліни та забезпечує більший контроль над витратами.

2.3 Характеристики об'єкта автоматизації "Finance Tracker"

Об'єктами автоматизації є система, що дозволяє користувачам вводити, зберігати та аналізувати свої фінансові дані, такі як зарплата, витрати та бюджети. Система має на меті надати користувачам зручний інструмент для управління особистими фінансами в офлайн-форматі.

Система забезпечує автоматизований облік фінансових даних, ведення звітності та збереження інформації з додатковим захистом від несанкціонованого доступу. Користувачі можуть самостійно вводити свої фінансові дані, контролювати витрати та планувати бюджети.

Користувачами системи є звичайні користувачі, які мають можливість переглядати, редагувати та видаляти інформацію про свої фінансові операції.

Система повинна підтверджувати факти внесення та збереження даних, щоб забезпечити точність та актуальність інформації.

Система повинна забезпечувати цілісність даних, що зберігаються, та захист від несанкціонованого доступу. Інформація, що зберігається в системі, включає:

- загальний баланс;
- витрати за категоріями;
- доходи;
- бюджет на місяць;
- статистику витрат і доходів;
- опис витрат та їхніх категорій.

Система також повинна надавати можливість підтримки та оновлення довідників, щоб зберігати актуальну інформацію про фінансові категорії та їхні характеристики.

3 ВИМОГИ ДО СИСТЕМИ

3.1 Вимоги до системи в цілому

При розробці системи мають бути дотримані такі принципи:

- функціональна достатність (повнота): система повинна забезпечувати всі необхідні функції для обліку фінансів;
- зручність та зрозумілість використання: інтерфейс повинен бути інтуїтивно зрозумілим для користувачів різного рівня навичок;
- відмовостійкість: система повинна бути здатна продовжувати працювати навіть у разі помилок чи збоїв;
- надійність зберігання даних: дані користувачів мають зберігатися без ризику втрати;
- придатність до модернізації та масштабування: система повинна легко адаптуватися до нових вимог і змінюватися з часом.

3.2 Вимоги до структури та функціонування

Система повинна забезпечити виконання таких функціональних задач:

- введення даних про доходи та витрати;
- фільтрування витрат за категоріями, датами та сумами.

Ці вимоги забезпечать користувачам зручний та ефективний інструмент для управління особистими фінансами.

4 Вимоги до структури та функціонування

4.1 Вимоги до надійності та відмовостійкості

ПЗ повинно функціонувати цілодобово, з можливістю оперативного відновлення після збоїв чи аварій. У штатному режимі роботи системи має бути забезпечено:

- надійне зберігання даних та обробка великих обсягів запитів;
- автоматичне завершення збереження даних у разі збоїв.

4.2 Вимоги до інтерфейсу

Інтерфейс системи має бути:

- зручним для навігації і використання;
- адаптивним до різних розмірів екранів;
- інтуїтивно зрозумілим, з мінімальними діями для виконання запитів користувачів.

4.3 Вимоги до захисту інформації

Вимоги до захисту інформації від несанкціонованого доступу включають:

- дотримання вимог законодавчих актів щодо захисту даних;
- використання засобів захисту даних.

4.4 Вимоги до розвитку та модернізації системи

Система повинна забезпечувати:

- масштабування системи з можливістю розширення функціоналу;
- інтеграцію нових автоматизованих перевірок без значних змін у поточному функціоналі.

4.5 Вимоги до стандартизації та уніфікації

Стандартизація системи повинна бути забезпечена використанням сучасних інструментів для розробки. Основні вимоги:

- використання стандартних технологій для побудови системи;
- уніфікація програмного забезпечення для спрощення обслуговування.

4.6 Вимоги до інформаційного забезпечення

Інформаційне забезпечення системи повинно враховувати:

- забезпечення цілісності даних та їх актуальності;
- мінімізацію надмірності даних та стандартизацію їх представлення.

5 АДМІНІСТРАТИВНА ІНФРАСТРУКТУРА

5.1 Розміщення системи

Доступність та тестування:

- протягом розробки та впровадження система повинна бути доступною для перегляду та тестування користувачами через браузер на будь-якому девайсі;

– для покращення експлуатації та тестування, система повинна мати окремі середовища для стабільного функціоналу та тестування нових функцій.

Середовища:

– PROD (Продуктивне середовище): Основне середовище для користувачів, надається Реципієнтом.

– DEV (Середовище розробки та тестування): Для розробки та тестування нових функцій, використовується для проміжного тестування.

Розгортання та адміністрування:

– початкове розгортання DEV середовища має бути реалізовано Виконавцем на ресурсах, наданих Замовником;

– розгортання та адміністрування PROD середовища буде виконуватись на обладнанні сторони отримувача продукту.

5.2 Логування

Мета та налаштування:

Для забезпечення безпеки даних та контролю доступу до функцій системи повинно бути реалізовано уніфіковане логування дій користувачів і змін даних.

Події для логування:

– запуск/зупинка сервісів: Логування подій запуску і зупинки окремих сервісів;

– помилки: Логування помилок, таких як комунікаційні проблеми, порушення цілісності даних, непередбачувані затримки в обробці інформації;
– критичні події: Логування критичних подій від системи моніторингу, наприклад, низька швидкість інтернету;
– використання системи: Логування запитів до системи, результатів обробки запитів та порушень порогових значень запитів.

Захист та форматування логів:
– формат логів: Логи повинні зберігатися в форматі текстових файлів;
– форматування подій: Протоколювання подій повинно дотримуватися встановленого формату, включаючи дату та час;
– маскування даних: Конфіденційні дані повинні бути масковані відповідно до встановленого формату;
– часові мітки: Часові мітки в реєстрах повинні зберігатися в UTC форматі.

Централізоване зберігання налаштувань:
– система повинна підтримувати централізоване зберігання налаштувань;
– налаштування мають бути представлені у вигляді конфігурації у форматі JSON;
– доступ до цих налаштувань має бути обмеженим. Адміністратор повинен мати можливість заповнювати ці налаштування, але звичайні користувачі не повинні мати доступ до їх перегляду або редагування.

6 ТЕХНОЛОГІЧНИЙ СТЕК

Система має бути створена з використанням технологічного стеку ASP.NET та MsSQL Local.

7 ПОРЯДОК КОНТРОЛЮ ТА ПРИЙМАННЯ СИСТЕМИ

7.1 Приймальні випробування

Створення робочої групи:

Для проведення приймальних випробувань буде сформована робоча група, до складу якої увійдуть представники як Замовника, так і Виконавця.

Процедура випробувань:

- випробування системи проводитимуться на стадії введення в дію з метою перевірки відповідності створеної системи вимогам технічного завдання (ТЗ);
- випробування повинні бути здійснені відповідно до програми та методики випробувань, розроблених Виконавцем і затверджених Замовником.

Передача результатів розробки:

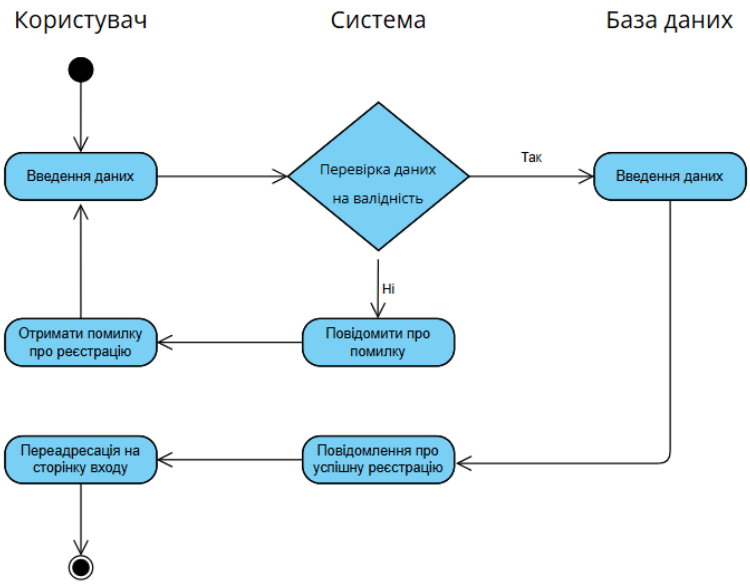
Після завершення розробки та успішного проходження випробувань, Виконавець зобов'язаний передати Замовнику проєктну документацію, що включає:

- технічне завдання;
- інструкції по встановленню та налаштуванню системи;
- інструкції користувачів системи;
- вихідні коди програмного забезпечення;
- акти приймання-передачі наданих послуг за етапами;
- акт передавання-прийняття примірника програмного забезпечення.

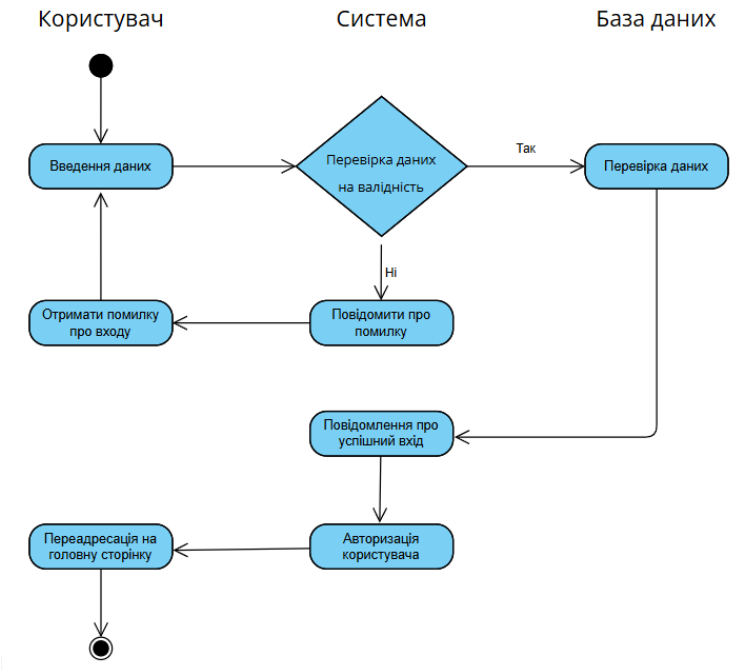
Додаток Б

(Обов’язковий)

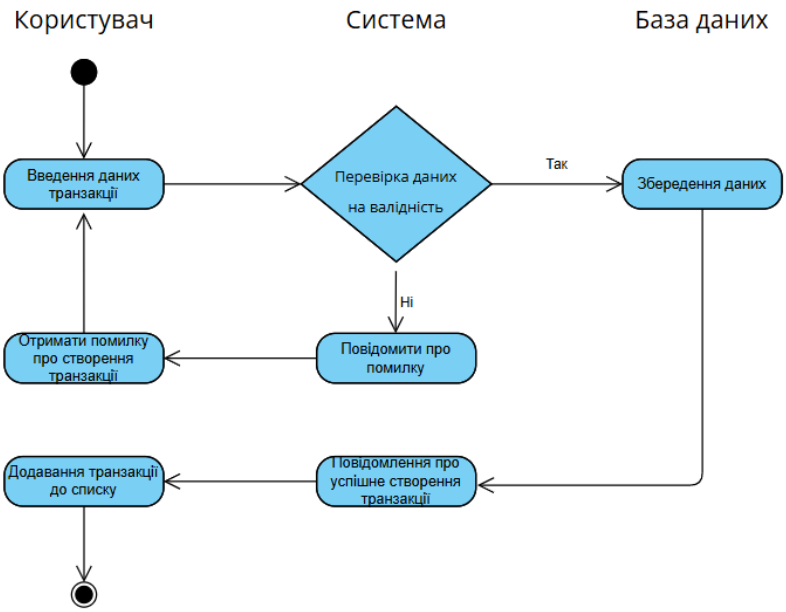
Діаграми діяльності



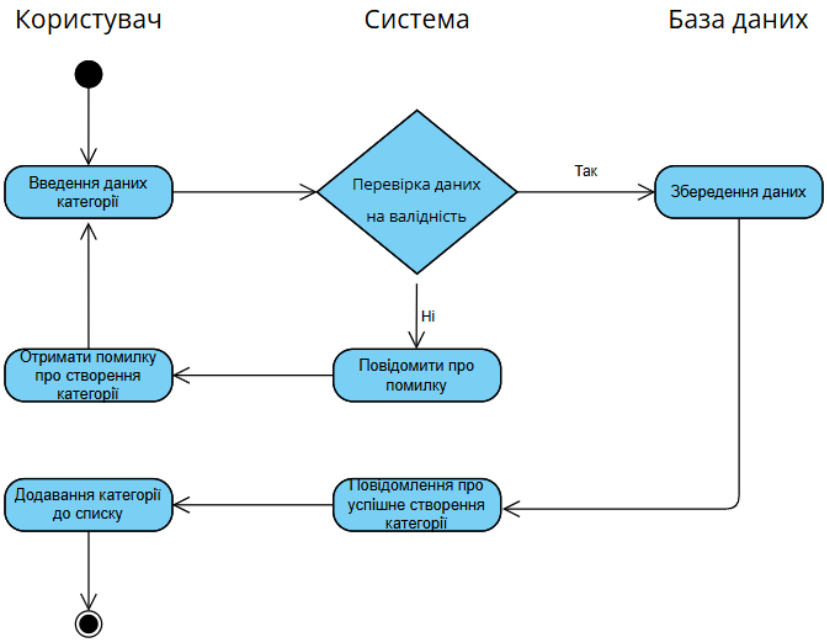
Додаток Б.1 — Діаграма діяльності для процесу «Реєстрація користувача»



Додаток Б.2— Діаграма діяльності для процесу «Вхід користувача»



Додаток Б.3— Діаграма діяльності для процесу «Додавання транзакції»



Додаток Б.4— Діаграма діяльності для процесу «Додавання категорій»

Додаток В

(Обов’язковий)

Код програми

Код контролера AccountController:

```
using Microsoft.AspNetCore.Mvc;
using Microsoft.AspNetCore.Authentication;
using Microsoft.AspNetCore.Authentication.Cookies;
using System.Security.Claims;
using Expense_Tracker.Models;
using System.Threading.Tasks;
using System.Collections.Generic;
using Microsoft.Extensions.Logging;

public class AccountController : Controller
{
    private readonly UserService _userService;
    private readonly ILogger<AccountController> _logger;

    public AccountController(UserService userService, ILogger<AccountController>
logger)
    {
        _userService = userService;
        _logger = logger;
    }

    [HttpGet]
    public IActionResult Login()
    {
        return View("~/Views/User/Login.cshtml");
    }

    [HttpPost]
    public async Task<IActionResult> Login(LoginViewModel model)
    {
        if (ModelState.IsValid)
        {
            var user = _userService.ValidateUser(model.Email, model.Password);
            if (user != null)
            {
                var claims = new List<Claim>
                {
                    new Claim(ClaimTypes.Name, user.UserName),
                    new Claim(ClaimTypes.Email, user.Email),
                    new Claim(ClaimTypes.Role, user.Role),
                    new Claim(ClaimTypes.NameIdentifier, user.Id.ToString())
                };

                var claimsIdentity = new ClaimsIdentity(claims,
CookieAuthenticationDefaults.AuthenticationScheme);
                await
HttpContext.SignInAsync(CookieAuthenticationDefaults.AuthenticationScheme, new
ClaimsPrincipal(claimsIdentity));

                return RedirectToAction("Index", "/");
            }

            ModelState.AddModelError("", "Невірна пошта або пароль");
        }

        return View("~/Views/User/Login.cshtml", model);
```

```
    }

    [HttpGet]
    public IActionResult Register()
    {
        return View("~/Views/User/Register.cshtml");
    }

    [HttpPost]
    public IActionResult Register(RegisterViewModel model)
    {
        if (ModelState.IsValid)
        {
            var user = new User
            {
                Email = model.Email,
                UserName = model.UserName,
                Password = model.Password
            };

            var result = _userService.RegisterUser(user);

            if (result)
            {
                return RedirectToAction("Login");
            }
            else
            {
                ModelState.AddModelError("", "Не вдалося зареєструвати користувача.");
            }
        }

        return View("~/Views/User/Register.cshtml", model);
    }

    [HttpPost]
    public async Task<IActionResult> Logout()
    {
        try
        {
            await
HttpContext.SignOutAsync(CookieAuthenticationDefaults.AuthenticationScheme);
            return Ok();
        }
        catch (Exception ex)
        {
            return StatusCode(500, $"Помилка при виході з акаунту: {ex.Message}");
        }
    }

    [HttpGet]
    public IActionResult Profile()
    {
        var userEmail = User.FindFirstValue(ClaimTypes.Email);


        var user = _userService.GetUserByEmail(userEmail);

        if (user == null)
        {
            return RedirectToAction("Login");
        }

        var profileViewModel = new ProfileViewModel
        {
```

```
            UserName = user.UserName,
            Email = user.Email
        };

        return View("~/Views/User/Profile.cshtml", profileViewModel);
    }


    [HttpPost]
    public IActionResult Profile(ProfileViewModel model)
    {
        if (ModelState.IsValid)
        {
            var userIdString = User.FindFirstValue(ClaimTypes.NameIdentifier);

            if (string.IsNullOrEmpty(userIdString))
            {
                return RedirectToAction("Login");
            }

            var userId = int.Parse(userIdString);

            var user = _userService.GetUserById(userId);

            if (user == null)
            {
                return RedirectToAction("Login");
            }

            user.UserName = model.UserName;
            user.Email = model.Email;

            if (!string.IsNullOrEmpty(model.Password))
            {
                user.Password = _userService.HashPassword(model.Password);
            }

            var result = _userService.UpdateUser(user);

            if (result)
            {
                var claims = new List<Claim>
            {
                new Claim(ClaimTypes.Name, user.UserName),
                new Claim(ClaimTypes.Email, user.Email),
                new Claim(ClaimTypes.Role, user.Role),
                new Claim(ClaimTypes.NameIdentifier, user.Id.ToString())
            };

                var claimsIdentity = new ClaimsIdentity(claims,
CookieAuthenticationDefaults.AuthenticationScheme);

HttpContext.SignInAsync(CookieAuthenticationDefaults.AuthenticationScheme, new
ClaimsPrincipal(claimsIdentity));

                TempData["SuccessMessage"] = "Профіль успішно оновлено!";
                return RedirectToAction("Profile");
            }

            ModelState.AddModelError("", "Не вдалося оновити профіль.");
        }

        return View("~/Views/User/Profile.cshtml", model);
    }
```

```
}
```

Код контролера AdminController:

```
using Expense_Tracker.Models;
using Microsoft.AspNetCore.Authorization;
using Microsoft.AspNetCore.Mvc;
using Microsoft.EntityFrameworkCore;

namespace Expense_Tracker.Controllers
{
    [Authorize(Roles = "Admin")]
    public class AdminController : Controller
    {
        private readonly ApplicationDbContext _context;
        private readonly UserService _userService;

        public AdminController(ApplicationDbContext context, UserService userService)
        {
            _context = context ?? throw new ArgumentNullException(nameof(context));
            _userService = userService ?? throw new
ArgumentNullException(nameof(userService));
        }

        public async Task<IActionResult> Users()
        {
            var users = await _context.Users.ToListAsync();
            return View(users);
        }

        [HttpPost]
        public async Task<IActionResult> DeleteUser(int id)
        {
            var user = await _context.Users.FindAsync(id);
            if (user == null)
            {
                return NotFound();
            }

            _context.Users.Remove(user);
            await _context.SaveChangesAsync();
            return RedirectToAction(nameof(Users));
        }

        [HttpGet]
        public IActionResult AddOrEditUser(int? id)
        {
            if (id == null)
            {
                return View(new UserViewModel());
            }

            var user = _userService.GetUserById(id.Value);

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

            var model = new UserViewModel
            {
                Id = user.Id,
                UserName = user.UserName,
```

```
                Email = user.Email,
                Role = user.Role
            };

            return View(model);
        }

        [HttpPost]
        public IActionResult AddOrEditUser(UserViewModel model)
        {
            if (ModelState.IsValid)
            {
                if (model.Id == 0)
                {
                    var newUser = new User
                    {
                        UserName = model.UserName,
                        Email = model.Email,
                        Role = model.Role
                    };

                    if (!string.IsNullOrEmpty(model.Password))
                    {
                        newUser.Password = _userService.HashPassword(model.Password);
                    }

                    try
                    {
                        var result = _userService.RegisterUser(newUser);
                        if (result)
                        {
                            TempData["SuccessMessage"] = "Користувача успішно
створено!";
                        }
                        else
                        {
                            ModelState.AddModelError("", "Користувач з таким email вже
існує.");
                        }
                    }
                    catch (Exception ex)
                    {
                        ModelState.AddModelError("", "Помилка при створенні
користувача: " + ex.Message);
                    }
                }
                else
                {
                    var user = _userService.GetUserById(model.Id);
                    if (user == null)
                    {
                        return NotFound();
                    }

                    user.UserName = model.UserName;
                    user.Email = model.Email;
                    user.Role = model.Role;

                    if (!string.IsNullOrEmpty(model.Password))
                    {
                        user.Password = _userService.HashPassword(model.Password);
                    }

                    try
                    {
```

```
                        var result = _userService.UpdateUser(user);
                        if (result)
                        {
                            TempData["SuccessMessage"] = "Дані користувача успішно
оновлено!";
                        }
                        else
                        {
                            ModelState.AddModelError("", "Не вдалося оновити
користувача.");
                        }
                    }
                    catch (Exception ex)
                    {
                        ModelState.AddModelError("", "Помилка при оновленні
користувача: " + ex.Message);
                    }
                }
            }

            return View(model);
        }

    }
}
```

Код контролера CategoryController:

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

namespace Expense_Tracker.Controllers
{
    public class CategoryController : Controller
    {
        private readonly ApplicationDbContext _context;

        public CategoryController(ApplicationDbContext context)
        {
            _context = context;
        }

        public async Task<IActionResult> Index()
        {
            return _context.Categories != null ?
                        View(await _context.Categories.ToListAsync()) :
                        Problem("Сутність 'ApplicationDbContext.Categories' пуста.");
        }

        public IActionResult AddOrEdit(int id = 0)
        {
            if (id == 0)
                return View(new Category());
            else
                return View(_context.Categories.Find(id));

        }
```

```
        [HttpPost]
        [ValidateAntiForgeryToken]
        public async Task<IActionResult> AddOrEdit([Bind("CategoryId,Title,Icon,Type")]
Category category)
        {
            if (ModelState.IsValid)
            {
                if (category.CategoryId == 0)
                    _context.Add(category);
                else
                    _context.Update(category);
                await _context.SaveChangesAsync();
                return RedirectToAction(nameof(Index));
            }
            return View(category);
        }

        [HttpPost, ActionName("Delete")]
        [ValidateAntiForgeryToken]
        public async Task<IActionResult> DeleteConfirmed(int id)
        {
            if (_context.Categories == null)
            {
                return Problem("Сутність 'ApplicationDbContext.Categories'  пуста.");
            }
            var category = await _context.Categories.FindAsync(id);
            if (category != null)
            {
                _context.Categories.Remove(category);
            }

            await _context.SaveChangesAsync();
            return RedirectToAction(nameof(Index));
        }

    }
}
```

Код контролера DashboardController:

```
using Expense_Tracker.Models;
using Microsoft.AspNetCore.Mvc;
using Microsoft.EntityFrameworkCore;
using System.Globalization;

namespace Expense_Tracker.Controllers
{
    public class DashboardController : Controller
    {
        private readonly ApplicationDbContext _context;

        public DashboardController(ApplicationDbContext context)
        {
            _context = context;
        }

        public async Task<ActionResult> Index()
        {
            DateTime StartDate = DateTime.Today.AddDays(-30);
```

```
            DateTime EndDate = DateTime.Today;

            List<Transaction> SelectedTransactions = await _context.Transactions
                .Include(x => x.Category)
                .Where(y => y.Date >= StartDate && y.Date <= EndDate &&
                            (y.Category.Type == "Income" || y.Category.Type ==
"Expense") &&
                            y.Amount >= 0)
                .ToListAsync();

            int TotalIncome = SelectedTransactions
                .Where(i => i.Category.Type == "Income")
                .Sum(j => j.Amount);
            ViewBag.TotalIncome = TotalIncome.ToString("C0");

            int TotalExpense = SelectedTransactions
                .Where(i => i.Category.Type == "Expense")
                .Sum(j => j.Amount);
            ViewBag.TotalExpense = TotalExpense.ToString("C0");

            int Balance = TotalIncome - TotalExpense;
            CultureInfo culture = CultureInfo.CreateSpecificCulture("uk-UA");
            culture.NumberFormat.CurrencySymbol = "₴";
            culture.NumberFormat.CurrencyNegativePattern = 1;
            ViewBag.Balance = String.Format(culture, "{0:C0}", Balance);

            ViewBag.DoughnutChartData = SelectedTransactions
                .Where(i => i.Category.Type == "Expense")
                .GroupBy(j => j.Category.CategoryId)
                .Select(k => new
                {
                    categoryTitleWithIcon = k.First().Category.Icon + " " +
k.First().Category.Title,
                    amount = k.Sum(j => j.Amount),
                    formattedAmount = k.Sum(j => j.Amount).ToString("C0"),
                })
                .OrderByDescending(l => l.amount)
                .ToList();

            List<SplineChartData> IncomeSummary = SelectedTransactions
                .Where(i => i.Category.Type == "Income")
                .GroupBy(j => j.Date)
                .Select(k => new SplineChartData()
                {
                    day = k.First().Date.ToString("dd-MMM"),
                    income = Math.Max(0, k.Sum(l => l.Amount))
                })
                .ToList();

            List<SplineChartData> ExpenseSummary = SelectedTransactions
                .Where(i => i.Category.Type == "Expense")
                .GroupBy(j => j.Date)
                .Select(k => new SplineChartData()
                {
                    day = k.First().Date.ToString("dd-MMM"),
                    expense = Math.Max(0, k.Sum(l => l.Amount))
                })
                .ToList();

            string[] Last30Days = Enumerable.Range(0, 31)
                .Select(i => StartDate.AddDays(i).ToString("dd-MMM"))
                .ToArray();

            ViewBag.SplineChartData = Last30Days
                .Select(day => new
```

```
                {
                    day = day,
                    income = IncomeSummary.FirstOrDefault(i => i.day == day)?.income ??
0,
                    expense = ExpenseSummary.FirstOrDefault(e => e.day == day)?.expense
?? 0,
                })
                .ToList();

            ViewBag.RecentTransactions = await _context.Transactions
                .Include(i => i.Category)
                .OrderByDescending(j => j.Date)
                .Take(5)
                .ToListAsync();

            return View();
        }
    }

    public class SplineChartData
    {
        public string day { get; set; }
        public int income { get; set; }
        public int expense { get; set; }
    }
}
```

Код контролера HomeController:

```
using Expense_Tracker.Models;
using Microsoft.AspNetCore.Mvc;
using System.Diagnostics;

namespace Expense_Tracker.Controllers
{
    public class HomeController : Controller
    {
        private readonly ILogger<HomeController> _logger;

        public HomeController(ILogger<HomeController> logger)
        {
            _logger = logger;
        }

        public IActionResult Index()
        {
            return View();
        }

        public IActionResult Privacy()
        {
            return View();
        }



        [ResponseCache(Duration = 0, Location = ResponseCacheLocation.None, NoStore =
true)]
        public IActionResult Error()
        {
            return View(new ErrorViewModel { RequestId = Activity.Current?.Id ??
HttpContext.TraceIdentifier });
        }
```

```
    }
}
```

Код контролера TransactionController:

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

namespace Expense_Tracker.Controllers
{
    public class TransactionController : Controller
    {
        private readonly ApplicationDbContext _context;

        public TransactionController(ApplicationDbContext context)
        {
            _context = context;
        }

        public async Task<IActionResult> Index()
        {
            var applicationDbContext = _context.Transactions.Include(t => t.Category);
            return View(await applicationDbContext.ToListAsync());
        }

        public IActionResult AddOrEdit(int id = 0)
        {
            PopulateCategories();
            if (id == 0)
                return View(new Transaction());
            else
                return View(_context.Transactions.Find(id));
        }

        [HttpPost]
        [ValidateAntiForgeryToken]
        public async Task<IActionResult>
AddOrEdit([Bind("TransactionId,CategoryId,Amount,Note,Date")] Transaction transaction)
        {
            if (ModelState.IsValid)
            {
                if (transaction.TransactionId == 0)
                    _context.Add(transaction);
                else
                    _context.Update(transaction);
                await _context.SaveChangesAsync();
                return RedirectToAction(nameof(Index));
            }
            PopulateCategories();
            return View(transaction);
        }

        [HttpPost, ActionName("Delete")]
        [ValidateAntiForgeryToken]
        public async Task<IActionResult> DeleteConfirmed(int id)
        {
            if (_context.Transactions == null)
            {
```

```
                return Problem("Entity set 'ApplicationDbContext.Transactions'  is
null.");
            }
            var transaction = await _context.Transactions.FindAsync(id);
            if (transaction != null)
            {
                _context.Transactions.Remove(transaction);
            }

            await _context.SaveChangesAsync();
            return RedirectToAction(nameof(Index));
        }


        [NonAction]
        public void PopulateCategories()
        {
            var CategoryCollection = _context.Categories.ToList();
            Category DefaultCategory = new Category() { CategoryId = 0, Title =
"Оберіть категорію" };
            CategoryCollection.Insert(0, DefaultCategory);
            ViewBag.Categories = CategoryCollection;
        }
    }
}
```

Додаток Г

(Обов’язковий)

Презентація

ХМЕЛЬНИЦЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

Кафедра інженерії програмного забезпечення

Кваліфікаційна робота на тему:

Вебзастосунок для персонального обліку фінансів користувача

Виконав:

Папка Станіслав Федорович

Студент 4 курсу, гр. ІПЗ-22-1

Керівник:

Праворська Н.І.

канд. пед. наук, доцент

Актуальність теми

Проблематика дослідження

- **Контроль ресурсів:** відсутність системного обліку фінансів у 68% населення;
- **Ергономіка:** висока складність та перевантаженість інтерфейсів наявних рішень;
- **Приватність:** незадовільний рівень захисту даних у публічних хмарних сервісах;
- **Доступність:** потреба у лаконічному та безкоштовному інструменті автоматизації.

Рішення

Розробка інтуїтивного вебзастосунку Finance-Tracker з функціями автоматизованого обліку, аналітики та візуалізації фінансових даних.

Мета та завдання проєкту

Мета роботи

Розробка захищеного та функціонального вебзастосунку для автоматизації персонального фінансового обліку на базі платформи ASP.NET Core MVC.

Основні завдання

- Аналіз предметної області та наявних аналогів;
- Формування функціональних та технічних вимог до ПЗ;
- Проєктування архітектури та реляційної моделі даних;
- Програмна реалізація модулів системи та аналітичного ядра;
- Проведення комплексного тестування працездатності.

Порівняльний аналіз аналогів

Критерій	Budget Bakers	YNAB	Finance-Tracker (Проєкт)
Складність	Висока	Висока	**Низька**
Ціна	9,99$/місяць	14,99$/місяць	**Безкоштовно**
Автономність	Частково	Ні	**Повна (Local-first)**
Тип сховища	Хмара	Хмара	**Локальна БД**

Вимоги до програмного забезпечення

Функціональні вимоги

- Автентифікація та авторизація користувачів;
- CRUD операції для транзакцій (дохід/витрата);
- Керування категоріями фінансових операцій;
- Формування інтерактивної аналітичної звітності.

Нефункціональні вимоги

- **Безпека:** хешування паролів за стандартом SHA256;
- **Швидкодія:** час відгуку системи не більше 0.2 сек;
- **Надійність:** підтримка транзакційної цілісності БД;
- **Адаптивність:** коректна робота у сучасних веббраузерах.

Архітектура та шаблони проєктування

Monolithic Layered

Розділення системи на рівень представлення, бізнес-логіку та рівень доступу до даних.

Патерн MVC

Відокремлення даних (Model) від інтерфейсу (View) через керуючий компонент (Controller).

DI Container

Ін'єкція залежностей для полегшення модульного тестування та слабкої зв'язності компонентів.

Декомпозиція системи

Identity Module

Забезпечує керування доступом, реєстрацію та безпеку профілю користувача.

Analytics Engine

Модуль агрегації та підготовки статистичних даних для бібліотеки Chart.js.

Core Finance Engine

Ядро системи для обробки транзакцій та миттєвого перерахунку балансу.

Infrastructure Layer

Взаємодія з базою даних MS SQL Server через Entity Framework Core.

Технологічний стек

Backend

C# / .NET 6.0
ASP.NET Core MVC
Entity Framework Core

Строга типізація та стабільність

Data Storage

MS SQL Server LocalDB

- Локальне зберігання;
- Повна приватність;

Frontend

Bootstrap 5
Chart.js
Razor Views

Реактивність та візуалізація

Програмна реалізація модулів

FinanceService

Реалізовано асинхронну логіку (async/await) для обробки масивів транзакцій без блокування UI. Це забезпечує плавність роботи інтерфейсу.

Data Intelligence

Використання **LINQ to Entities** для агрегації даних (Sum, GroupBy) безпосередньо на стороні бази даних для максимальної швидкодії.

Безпека даних (LocalDB Integration)

База даних ініціалізується у папці App_Data користувача. Використання локального реляційного сховища забезпечує портативність та виключає передачу фінансової історії на сторонні сервери.

Технічне та програмне забезпечення

Апаратне забезпечення

- **CPU:** 2.0 GHz (2+ ядра);
- **RAM:** 4 ГБ (споживання ПЗ ~300 МБ);
- **Диск:** 500 МБ вільного місця.

Програмне середовище

- **ОС:** Windows 10/11;
- **Платформа:** .NET Runtime 6.0;
- **Браузер:** Chrome, Edge, Firefox.

Результати тестування

27	120 мс	100%
Unit тестів (xUnit)	Час виконання запиту	Стійкість до SQLi
Покриття сервісів — 85%	Висока швидкість LocalDB	Параметризація EF Core

Стабільність: Успішно протестовано сценарій «холодного старту» бази даних та автоматичного відновлення схеми при першому запуску застосунку на новому пристрої.

```
Slow test file: [Stage E2E] › tests/e2e/billing/billing-overview.spec.ts (5.9m)
Consider running tests from slow files in parallel. See: https://playwright.dev/docs/test-parallel
27 passed (6.9m)
```

Графічний інтерфейс користувача

Інтерфейс спроєктований за принципами мінімалізму та фокусу на даних:

- **Дашборд:** миттєвий огляд поточного балансу та графік витрат;
- **Транзакції:** швидке додавання нових записів;
- **Аналітика:** інтерактивні діаграми Chart.js із можливістю фільтрації;
- **Адаптивність:** автоматичне підлаштування розмітки під мобільні пристрої.

Висновки

Завдання проєкту	Результат виконання
Аналіз предметної області	Проведено, обґрунтовано вибір Local-first підходу.
Формування вимог	Визначено повний перелік функціональних та ТТХ.
Проєктування архітектури	Розроблено MVC-структуру та реляційну схему БД.
Вибір технологій	Обґрунтовано та застосовано стек .NET / SQL Server.
Програмна реалізація	Створено функціональний вебзастосунок Finance-Tracker.
Тестування	Всі 27 xUnit тестів та функціональні сценарії пройдено.

Апробація результатів дослідження

Наукові публікації

Результати роботи та основні архітектурні рішення були представлені на Всеукраїнській науково-практичній конференції.

***Папка С. Ф.**, Праворська Н. І. Веб-застосунок для персонального обліку фінансів. // Актуальні проблеми комп'ютерних наук : збірник наукових праць за матеріалами XVII Всеукраїнської науково-практичної конференції «АПКН-2025» (м. Хмельницький, 14–15 листопада 2025 р.). — Хмельницький : ХНУ, 2025. — С. 339–342.*

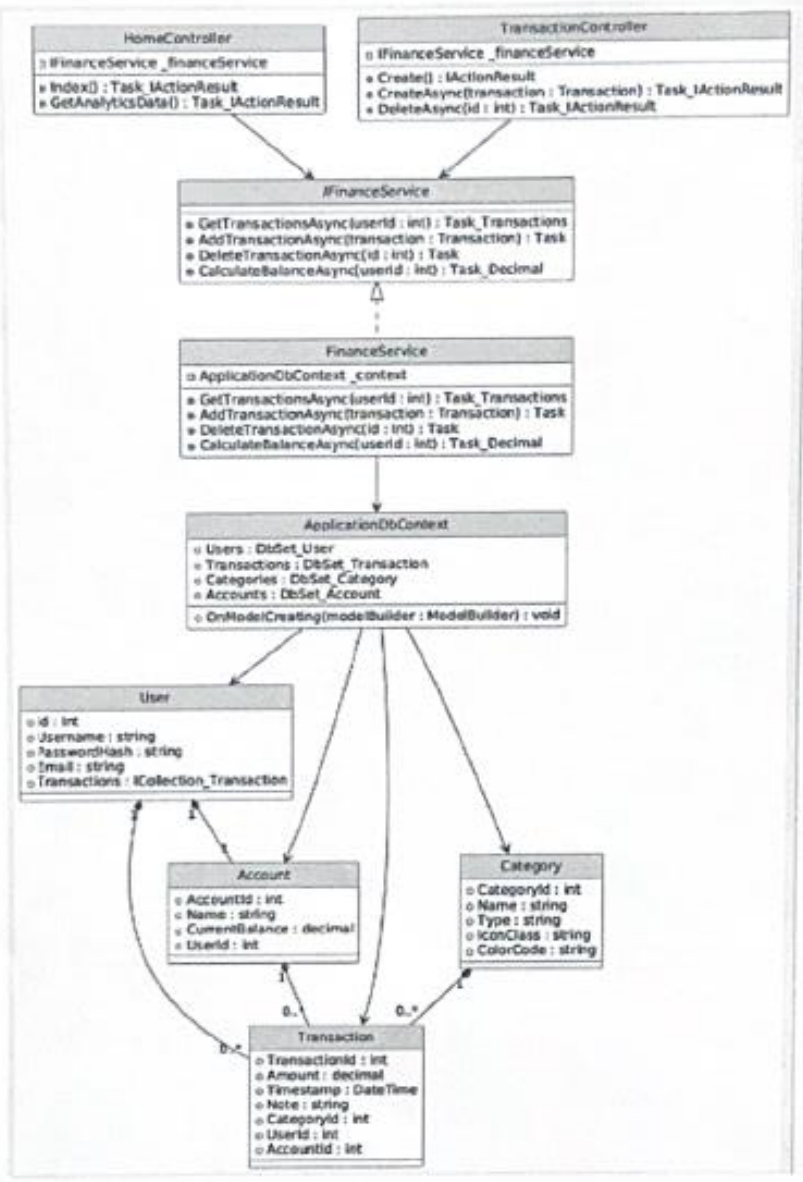
Дякую за увагу!

Готовий відповісти на ваші запитання

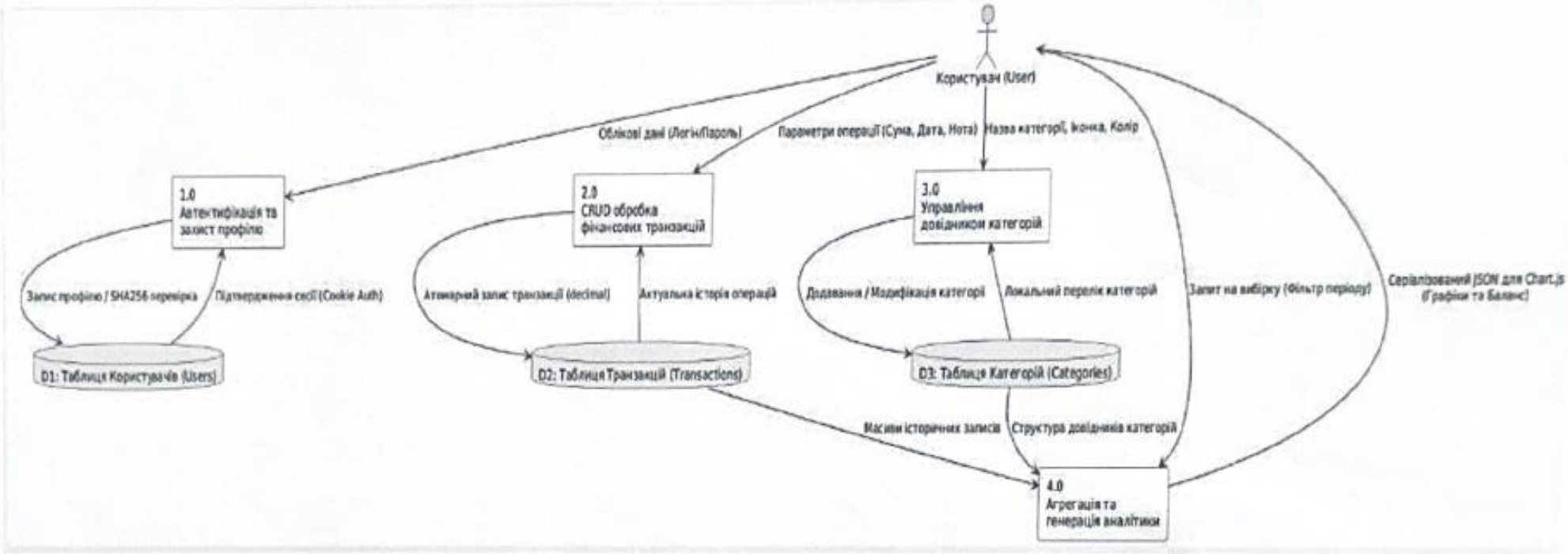
Папка Станіслав Федорович

stanislavp@khmnu.edu.ua

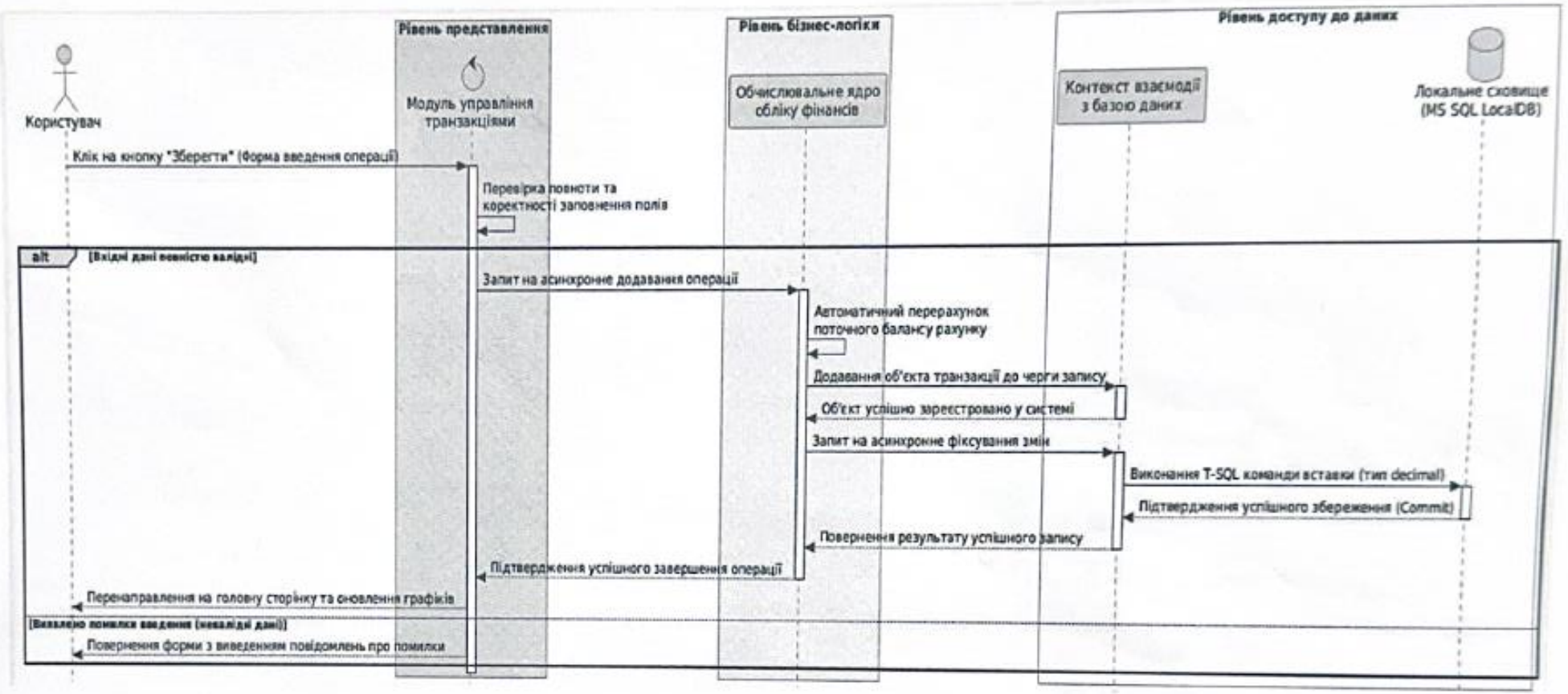
ГРАФІЧНІ МАТЕРІАЛ



					КвРІПЗ.2201102.01.10.Е8			
Змн.	Арк.	№ докум.	Підпис	Дата	Вебзастосунок для персонального обліку фінансів користувача Діаграма класів	Літ.	Маса	Масштаб
Розробив		Папка С.Ф.		15.05				
Керівник		Праворська Н.І.		25.05		Аркуш 1	Аркушів 3	
Н. Контр.		Янина О.М.		25.05		ХНУ. ІПЗ-22-1		
Затверд.		Бедратюк Л. П.		25.05				



					КвРІПЗ.2201102.01.10.Е8	
Змн.	Арк.	№ докум.	Підпис	Дата		
Розробив		Папка С.Ф.		25.05	Веб-застосунок для персонального обліку фінансів користувача Діаграма класів	Літ. / Маса / Масштаб
Керівник		Праворська Н.І.		25.05		Аркуш 2 / Аркушів 3
Н. Контр.		Яшина О.М.		25.05		ХНУ. ІПЗ-22-1
Затверд.		Бедратюк Л. П.		26.05		



Змн.	Арк.	№ докум.	Підпис	Дата	КвРПЗ.2201102.01.10.Е8		
Розробив		Папка С.Ф.		25.05	Вебзастосунок для персонального обліку фінансів користувача	Літ.	Маса	Масштаб
Керівник		Праворська Н.І.		25.05				
					Діаграма процесів	Аркуш 3	Аркушів 3	
Н. Контр.		Яшина О.М.		25.05		ХНУ. ІПЗ-22-1		
Затверд.		Бедратюк Л. П.		25.05				

СУПРОВІДНІ ДОКУМЕНТИ

Завідувачу кафедри інженерії програмного
забезпечення проф. Леоніду БЕДРАТЮКУ
здобувача вищої освіти
Папка Станіслава Федоровича
факультет ІТ, IVкурс, група ІПЗ-22-1

ЗАЯВА

З правилами чинного Положення про систему забезпечення академічної доброчесності в Хмельницькому національному університеті, згідно з яким виявлення академічного плагіату є підставою для відмови в допуску кваліфікаційної роботи до захисту і застосування заходів дисциплінарної та академічної відповідальності, ознайомлений. Про використання програмно-технічних засобів для перевірки кваліфікаційних робіт здобувачів вищої освіти на наявність академічного плагіату оповіщений та надаю свою згоду на обробку й збереження університетом моєї роботи в інституційному репозитарії Хмельницького національного університету.

Також надаю університету право на передачу моєї роботи для обробки та збереження в базах даних програмно-обчислювального комплексу StrikePlagiarism та/або програмно-технічного засобу AntiPlagiarism і використання роботи для виявлення академічного плагіату в інших роботах, які перевіряються програмно-технічними засобами та користувачами, що мають доступ до цих програмно-технічних засобів, виключно в обмежених цілях для виявлення текстових збігів у роботах.

Робота надається для перевірки в електронному варіанті. Електронна версія моєї роботи збігається (ідентична) з друкованою.

01.05.2026
дата

підпис

Протокол аналізу звіту подібності науковим керівником

Заявляю, що я ознайомився (-лась) з Повним звітом подібності, який був згенерований Системою виявлення і запобігання плагіату щодо роботи:

Автор: Станіслав ПАПКА

Співавтор:

Назва: Вебзастосунок для персонального обліку фінансів користувача

Науковий керівник: канд. пед. наук, доцент Наталія ПРАВОРСЬКА

Підрозділ: Кафедра інженерії програмного забезпечення

Коефіцієнт подібності 1:8.79%

Коефіцієнт подібності 2:2.18%

Мікропробіли: 29

Заміна букв: 1

Інтервали: 0

Білі знаки: 1254

Дата створення звіту: 2026-05-28 22:12:25.0

Після аналізу Звіту подібності констатую наступне:

☑ **Запозичення, виявлені в роботі є законними і не є плагіатом. Рівень подібності не перевищує допустимої межі. Таким чином робота незалежна і приймається.**

☐ **Запозичення не є плагіатом, але перевищено граничне значення рівня подібностей. Таким чином робота повертається на доопрацювання.**

☐ **Виявлено запозичення і плагіат або навмисні текстові спотворення (маніпуляції), як передбачувані спроби укриття плагіату, які роблять роботу невідповідною вимогам законодавства (Ст. 32. ЗУ Про вищу освіту, пункт 3.1, Ст. 42. ЗУ Про освіту) та вимог НАЗЯВО (Критерій 5), а також кодексу етики і процедурам. Таким чином робота не приймається.**

Обгрунтування:

Дата 25.05.26

експерт

Anti-Plagiarism (http://ap.km.ua) v-16.718

Максимальне співпадіння з одним документом 4.0%

Словники перевірки: UA, US, RU. **Помилок в документах: 15%**

ID: 272698 Назва: БКР_Вебзастосунок для персонального обліку фінансів користувача Додано в БД: 2026-05-28 Автора: Станіслав ПАПКА Керівники: канд. пед. наук, доцент Наталія ПРАВОРСЬКА Консультанти: Опоненти:	Документ		Сумарний збіг по Базі Даних	
	Символи	Лексеми	Символи	Лексеми
	98534	619	11256 (11%)	136 (22%)

Джерело плагіату

ID	Опис	Наявність плагіату в документі	
		Символи	Лексеми

ХМЕЛЬНИЦЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

РЕЦЕНЗІЯ НА КВАЛІФІКАЦІЙНУ РОБОТУ
освітнього ступеня «Бакалавр»

Дипломник Папка Станіслав Федорович

Тема Вебзастосунок для персонального обліку фінансів користувача

Спеціальність 121 – Інженерія програмного забезпечення

Обсяг кваліфікаційної роботи:

Кількість листів креслень 3; кількість сторінок записки 59

1.Короткий зміст пояснювальної записки та прийнятих рішень У кваліфікаційній роботі досліджено й проаналізовано предметну область автоматизації обліку особистих фінансів, а також формалізовано всі функціональні та нефункціональні вимоги до проектованої системи. Було проведено глибокий порівняльний аналіз чинних ринкових аналогів (зокрема хмарних PFM-платформ та десктопних бухгалтерських програм), виявлено їхні ключові архітектурні обмеження у сфері конфіденційності й ергономіки, що дозволило науково обґрунтувати актуальність розробки нового автономного програмного забезпечення.

2. Висновок про відповідність роботи поставленому завданню Кваліфікаційна робота виконана відповідно до поставленого завдання та з дотриманням всіх вимог.

3. Характеристика виконання кожного розділу роботи, ступінь використання останніх досягнень науки і техніки та передових методів роботи У вступі обґрунтовано актуальність теми, визначено мету та завдання дипломного проєктування. У першому розділі проведено аналіз предметної області обліку фінансів, розглянуто чинні аналоги й визначено функціональні та нефункціональні вимоги до системи. У другому розділі досліджено сучасні архітектурні патерни, обґрунтовано вибір монолітної структури, моделі клієнт-сервер та шаблону MVC. У третьому розділі налаштовано інфраструктуру платформи .NET 6.0, виконано практичну розробку програмних модулів мовою C# із використанням Entity Framework Core й MS SQL Server LocalDB, а також описано створення інтерфейсу на базі Bootstrap 5 та Chart.js, в результаті чого створено вебзастосунок «Finance-Tracker». Також у цьому розділі проведено комплексне функціональне тестування системи відповідно до встановлених вимог, що експериментально підтвердило її високу швидкодію та повністю коректну роботу.

4. Позитивні сторони роботи Тематика кваліфікаційної роботи є високоактуальною, оскільки на сьогодні на ринку України більшість рішень для моніторингу капіталу є закритими хмарними платформами, які не забезпечують належного рівня конфіденційності й суверенітету над особистими даними. У межах проєкту було успішно реалізовано парадигму безпеки «Local-first», що гарантує повну локалізацію обчислювальних процесів без передачі інформації у зовнішнє мережеве середовище.

5. Негативні сторони роботи У розробленому вебзастосунку фільтрація та перегляд фінансових транзакцій реалізовані лише за базовими локальними категоріями (з можливістю створення власних) – у подальшому було б доцільно впровадити інтерактивний розширений пошук із можливістю динамічної фільтрації за діапазонами сум, точними часовими інтервалами та текстовими тегами. Також для покращення користувацького досвіду варто було б реалізувати систему автоматичних миттєвих сповіщень або попереджень на головному дашборді, коли

поточні витрати користувача за певною категорією наближаються до встановленого місячного ліміту бюджету

6. Оцінка графічного оформлення та пояснювальної записки Графічне оформлення виконано відповідно до теми кваліфікаційної роботи та подано у вигляді діаграм і рисунків. Пояснювальна записка оформлена згідно вимог чинних стандартів.

7. Відгук про кваліфікаційну роботу в цілому Кваліфікаційна робота заслуговує на високу позитивну оцінку. Матеріал пояснювальної записки чітко структурований, послідовний, викладений глибокою технічною мовою та логічно вибудований, що дозволяє повноцінно зрозуміти всі нюанси архітектурного проєктування локального вебзастосунку для автоматизації обліку особистих фінансів. Наведений графічний матеріал, зокрема деталізовані DFD-діаграми потоків даних, UML-схеми взаємодії процесів та екранні форми інтерфейсу, дає можливість наочно оцінити високу якість інженерного проєктування ізольованої системи, реляційної бази даних та підсистеми аналітичної візуалізації.

8. Інші зауваження

9. Оцінка кваліфікаційної роботи Кваліфікаційна робота виконана у повному обсязі, відповідає поставленій задачі та заслуговує на оцінку «відмінно».

РЕЦЕНЗЕНТ Висіна Тетяна Анатоліївна, к.ф.-м.н., доцент кафедри КІС, ХНУ

"01" жовтня 2026 р. _____
(підпис)

РЕЦЕНЗІЯ НА КВАЛІФІКАЦІЙНУ РОБОТУ

першого освітнього рівня «Бакалавр»

Студента: Папки Станіслава Федоровича
Група: ІПЗ-22-1

Тема: *«Вебзастосунок для персонального обліку фінансів користувача»*

Спеціальність: 121 – Інженерія програмного забезпечення

Короткий зміст пояснювальної записки

Кваліфікаційна робота присвячена розробці вебзастосунку «Finance-Tracker», метою якого є забезпечення ефективного обліку доходів і витрат, високої швидкості обробки даних та надійного захисту фінансової інформації. Актуальність теми обумовлена потребою в автоматизації фінансових процесів малого бізнесу, що вимагає простих у використанні та безпечних рішень. Завдання роботи включають аналіз предметної області, вибір технологій, проектування архітектури системи, розробку модулів та проведення тестування.

Відповідність отриманих результатів роботи поставленим завданням

Завдання, сформульовані у вступі, включають: 1. Аналіз предметної області та сучасних тенденцій у фінансових технологіях. 2. Вибір оптимального стеку технологій. 3. Проектування архітектури системи. 4. Розробка модулів та тестування.

Результати, викладені у висновках, свідчать про те, що: - Аналіз предметної області проведено, але з недостатньою деталізацією. - Вибір технологій обґрунтований, але

потребує більшого аналізу альтернатив. - Архітектура системи спроектована, але без чіткого опису рівнів. - Реалізація модулів описана, але без фрагментів коду.

Оцінка відповідності: в цілому відповідають.

Оцінка розділів

Розділ 1: Змістовий аналіз предметної області, її структурних та функціональних особливостей

Опис предметної області проведено, але є загальним і не містить конкретних прикладів. Проблеми, які вирішує ПЗ, окреслені, але без статистичних даних. Структура та функції фінансових систем описані, але не деталізовані. Визначення користувачів та їх потреб є загальним, а безпека даних підкреслюється, але без конкретних механізмів.

Позитивні сторони: описано етапи розвитку фінансових технологій, вказано на важливість людино-орієнтованого дизайну, визначено основні проблеми.

Недоліки: недостатня деталізація опису, відсутність конкретних прикладів, загальність у визначенні користувачів.

Розділ 2: Проектування архітектури програмного забезпечення

Описано модульну архітектуру на основі патерну MVC, але не наведено альтернатив для порівняння. Вибір архітектури обґрунтовано вимогами до безпеки, але без чіткого зв'язку з попередніми вимогами. Шаблони проєктування згадуються, але не аналізуються. Взаємозв'язки між модулями не описані.

Позитивні сторони: чітке визначення модулів, узгодженість реалізації з архітектурою.

Недоліки: відсутність аналізу альтернатив, недостатня деталізація функцій модулів, відсутність опису міжмодульних залежностей.

Розділ 3: Програмна реалізація та тестування програмного забезпечення

Описано реалізацію модулів, але без фрагментів коду. Структура бази даних описана, але без ER-діаграм. Визначено системні вимоги, але без чіткої структури. Тестування проведено, але без конкретних прикладів тестів.

Позитивні сторони: детальний опис реалізації модулів, узгодженість з архітектурою, проведення тестування.

Недоліки: відсутність фрагментів коду, ER-діаграм, конкретних прикладів тестів.

Позитивні сторони

Розробка вебзастосунку «Finance-Tracker» демонструє оригінальність у підходах до автоматизації фінансового обліку. Використання сучасних технологій, таких як ASP.NET Core та Entity Framework, свідчить про технологічну доцільність. Описані модулі мають чітке призначення, що підвищує зручність використання.

Недоліки

Недоліки роботи включають недостатню деталізацію в описах предметної області, відсутність конкретних прикладів у розділі про аналіз технологій, а також недостатню візуалізацію структури бази даних і реалізації модулів. Це може ускладнити розуміння роботи для читача.

Відгук в цілому

Кваліфікаційна робота є актуальною та має практичну значущість у контексті автоматизації фінансових процесів. Зміст роботи відповідає темі та завданню, але потребує покращення в деталізації та структуризації. Новизна ідей та рішень присутня, але потребує більшої обґрунтованості. Якість реалізації є задовільною, але є можливості для вдосконалення.

Оцінка кваліфікаційної роботи

Кваліфікаційна робота заслуговує оцінки "добре". Вона виконана в основному в повному обсязі, але має недоліки, які потребують доопрацювання.

Рекомендації

Рекомендується доопрацювати роботу, зокрема в частині деталізації описів, додавання прикладів, візуалізації та чіткішого обґрунтування вибору технологій. Це підвищить якість роботи та її сприйняття.

OpenAI API-асистент
Session ID: 59bd5010-a457-437b-aec8-f9126ed762d9
Підписано автоматично, модель gpt-4o-mini
Дата: 29.05.2026

РІШЕННЯ ЕКСПЕРНОЇ КОМІСІЇ
КАФЕДРИ ІНЖЕНЕРІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ
ПРО ДОПУСК КВАЛІФІКАЦІЙНОЇ РОБОТИ ДО ЗАХИСТУ

Підтверджуємо ознайомлення з результатами звіту/звітів перевірки роботи, продукованими програмно-технічним засобом (ами), на наявність текстових збігів.
Назва кваліфікаційної роботи: «Вебзастосунок для персонального обліку фінансів користувача»
Автор: Папка Станіслав Федорович
Освітня програма: Освітньо-професійна програма «Інженерія програмного забезпечення»
Спеціальність: 121 – Інженерія програмного забезпечення
Науковий керівник: Праворська Наталія Іванівна, кандидат пед. наук, доцент
Після аналізу звіту/звітів зроблено такий висновок:

№	Висновок	Позначка про відповідність
1	Запозичення, виявлені в роботі є законними і не є плагіатом. Рівень подібності не перевищує допустимої межі. Таким чином робота незалежна і приймається до захисту.	**відповідає**
2	Виявлені запозичення не є академічним плагіатом, розміщені в розділах, які не описують безпосередньо авторське дослідження, але кількість цитат перевищує обсяг, виправданий поставленою метою роботи. Робота приймається до захисту, але має бути відкоригована.	
3	Виявлені запозичення не є плагіатом, але перевищено граничне значення рівня подібностей. Виявлені запозичення частково розміщені в розділах, які описують безпосередньо авторське дослідження, а кількість цитат перевищує обсяг, виправданий поставленою метою роботи. В зв'язку з цим мета роботи та поставлені завдання не були досягнуті. Робота може бути допущена до захисту після того, як буде відкоригована та допрацьована і успішно пройде повторну перевірку на академічний плагіат.	
4	Робота містить навмисні текстові спотворення, передбачувані спроби укриття текстових запозичень або інші прояви академічного плагіату. Робота містить фабрикацію або фальсифікацію даних. Робота не допускається до захисту.	
5	Інше:	

Підтвердження:

Кваліфікаційна робота виконана автором самостійно із дотриманням усіх установлених норм та правил академічної доброчесності. Усі наукові положення, теоретичні підходи, інженерні рішення, а також фрагменти вихідного коду програмного забезпечення, які запозичені з літературних або електронних джерел, супроводжуються обов'язковими та коректними посиланнями на першоджерела у відповідних розділах пояснювальної записки.

Текст кваліфікаційної роботи та розроблені програмні модулі пройшли обов'язкову первинну перевірку на наявність текстових запозичень та ознак плагіату за допомогою спеціалізованого ліцензійного програмного забезпечення, затвердженого закладом вищої освіти. За результатами комп'ютерного аналізу встановлено, що викладений матеріал є оригінальним та відповідає чинним нормативним вимогам щодо унікальності наукових текстів. Повний розгорнутий технічний звіт та протокол перевірки надійності системи та унікальності тексту додаються до загального пакета супровідних документів.

Дата 25.05.2026р.

Завідувач кафедри _____ Леонід БЕДРАТЮК

Гарант освітньої програми _____ Леонід БЕДРАТЮК

Керівник кваліфікаційної роботи _____ Наталія ПРАВОРСЬКА