

МІНІСТЕРСТВО ФІНАНСІВ УКРАЇНИ
ДЕРЖАВНИЙ ПОДАТКОВИЙ УНІВЕРСИТЕТ

ФАКУЛЬТЕТ ФІНАНСІВ ТА ЦИФРОВИХ ТЕХНОЛОГІЙ
КАФЕДРА КОМП'ЮТЕРНИХ ТА ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ І
СИСТЕМ

КВАЛІФІКАЦІЙНА РОБОТА

на тему: «Розробка вебсайту для планування сімейного бюджету»

Здобувача вищої освіти другого (магістерського) рівня,
II курсу, групи КМІ-23-1, спеціальності 122 «Комп'ютерні науки»,
освітня програма «Інформаційні управляючі системи і технології в економіці»


(підпис)

Артеменко Марк
Сергійович
(прізвище, ім'я, по батькові)

Науковий керівник


(підпис)

Омельчук Антон
Анатолійович к.т.н.
(прізвище, ім'я, по батькові,
науковий ступінь, вчене звання)

Керівник освітньо-
професійної програми


(підпис)

Горбовий Артур Юліанович,
д.т.н, професор
(прізвище, ім'я, по батькові, науковий
ступінь, вчене звання)

Допущено до захисту за рішенням кафедри протокол від «10» січня 2025 №11

Завідувач кафедри


(підпис)

Філоненко Михайло
Миколайович, к.ф.-м.н.,
доцент
(прізвище, ім'я, по батькові,
науковий ступінь, вчене звання)

Ірпінь – 2025

Реєстрація № 7

27.01.2025
(дата)


(підпис відповідальної особи)

Результати перевірки

до захисту
(до захисту, на доопрацювання)

27.01.2025
(дата)


(підпис)

Результат захисту

4 (добре)
(оцінка)

05.02.2025
(дата)


(підпис)

Голова комісії:


(підпис)

В.М. Франчук

Члени комісії:


(підпис)

А.Ю. Горбовий


(підпис)

М.М. Філоненко


(підпис)

І.Д. Погореловська

БЛАНК ОЦІНЮВАННЯ КВАЛІФІКАЦІЙНОЇ РОБОТИ

Об'єкти контролю	Діапазон оцінки в балах	Фактична кількість балів	Примітки	Дата, П.І.Б.: керівника, голови комісії членів комісії, секретаря, (підписи)
ОЦІНЮВАННЯ КВАЛІФІКАЦІЙНОЇ РОБОТИ КОМІСІЄЮ ПО ЗАХИСТУ				
1. Зміст кваліфікаційної роботи (якість теоретичного матеріалу, його зв'язок з практичним, вміння представляти матеріал в логічній послідовності, наявність висновків та пропозицій)	від 0 до 30	22		05.02.2025 Франчук В.М. <i>В.М.</i> Горбовий А.Ю. <i>А.Ю.</i>
2. Дотримання вимог щодо оформлення	від 0 до 10	7		Філоненко М.М. <i>М.М.</i>
3. Вчасне виконання роботи на всіх етапах у встановлені кафедрою терміни	від 0 до 5	4		Погореловська І.Д. <i>І.Д.</i> Суховерхівка Л.П. <i>Л.П.</i>
4. Рецензент	від 0 до 5	5		
РАЗОМ БАЛІВ	від 0 до 50	38		

Оцінювання кваліфікаційної роботи науковим керівником - 37 балів.

ПЕРЕВЕДЕННЯ ОЦІНКИ ЗА 100-БАЛЬНОЮ ШКАЛОЮ ДО НАЦІОНАЛЬНОЇ ШКАЛИ ТА ШКАЛИ ЄКТС

Оцінка за 100-бальною шкалою	Оцінка за шкалою ЄКТС	Оцінка за національною шкалою	Дата	П.І.Б.: голови, членів комісії, секретаря, (підпис)
75	C	4 (добре)	05.02.2025	Франчук В.М. <i>В.М.</i> Горбовий А.Ю. <i>А.Ю.</i> Філоненко М.М. <i>М.М.</i> Погореловська І.Д. <i>І.Д.</i> Суховерхівка Л.П. <i>Л.П.</i>

ПРАВИЛА ПЕРЕВЕДЕННЯ ОЦІНОК ДО НАЦІОНАЛЬНОЇ ШКАЛИ ТА ШКАЛИ ЄКТС

Оцінка за 100-бальною шкалою	Оцінка за шкалою ЄКТС	Оцінка за національною шкалою
90-100	A	5 (відмінно)
80-89	B	4 (добре)
70-79	C	
60-69	D	3 (задовільно)
50-59	E	
35-49	FX	2 (незадовільно)

Факультет фінансів та цифрових технологій
Кафедра комп'ютерних та інформаційних технологій і систем
Галузь знань 12 «Інформаційні технології»
Спеціальність 122 «Комп'ютерні науки»
Освітня програма «Інформаційні управляючі системи і технології в економіці»

Затверджую

Завідувач кафедри  Філоненко М.М.

«23» грудня 2023

Завдання на кваліфікаційну роботу
здобувачу вищої освіти
Артеменку Марку Сергійовичу
(прізвище, ім'я, по батькові)

1. Тема кваліфікаційної роботи ***«Розробка вебсайту для планування сімейного бюджету»***

Затверджена наказом ректора від «20» грудня 2023 № 1701

2. Строк здачі закінченої роботи *«07» січня 2025*

3. Мета роботи: створення середовища для надання можливості відслідковування та керування особистими та сімейними фінансовими ресурсами.

Об'єкт дослідження: автоматизація процесу керування та контролю особистих та сімейних фінансів.

Предмет дослідження: методи та засоби реалізації проекту, забезпечення його функціонування й автоматизації процесу управління фінансами.

Орієнтовна база дослідження: контроль фінансів, менеджмент особистих грошових ресурсів, бази даних, реалізація програмної логіки.

4. Вхідні дані до роботи: аналіз проблематики управління особистими фінансовими ресурсами, дослідження методів та засобів реалізації програмної системи.

5. Перелік графічного матеріалу: рисунки моделі бази даних, структурні схеми, лістинги програм.

6. Зміст кваліфікаційної роботи (перелік питань за кожним розділом)

1. Аналіз Сучасного стану обраної задачі

1.1 Аналіз теми управління особистими фінансами

1.2 Дослідження наявних рішень проблеми

1.3 Дослідження та підбір основних методів та засобів для розробки проекту

2. Практичні результати досліджень

2.1 Розробка бази даних

2.2 Реалізація взаємодії з базою даних

2.3 Реалізація функціоналу взаємодії

3. Аналіз комерційного впровадження додатку та пропозиції щодо удосконалення наявного проекту

3.1 Фінансове обґрунтування доцільності комерційного впровадження додатку

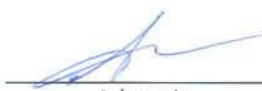
3.2 Пропозиції щодо удосконалення наявного проекту

7. Календарний план виконання роботи

№ п/п	Назва етапів кваліфікаційної роботи	Строк виконання етапів роботи	
		За планом	Фактично
1	2	3	4
1	Вибір теми, підбір літературних джерел, формування структури кваліфікаційної роботи	До 15.12.2023	15.12.2023
2	I розділ (розкрити сутність і стан дослідження наукової проблематики роботи, після чого навести її загальну характеристику; огляд літератури за темою і вибір конкретних напрямів досліджень)	27.05.2024	30.05.2024
3	II розділ (аналіз експериментальної частини про методики досліджень)	02.09.2024	10.09.2024
4	III розділ (проведені теоретичні і (або) експериментальні дослідження та проектні рішення)	18.11.2024	20.11.2024
5	Висновки	10.12.2024	10.12.2024
6	Доопрацювання роботи з урахуванням зауважень наукового керівника	23.12.2024	23.12.2024
7	Отримання відгуку керівника	28.01.2025	28.01.2025
8	Отримання зовнішньої рецензії	31.01.2025	31.01.2025

8. Дата видачі завдання «23» грудня 2023

9. Керівник кваліфікаційної роботи


(підпис)

Омельчук А. А.
(прізвище, ініціали)

10. Керівник освітньо-професійної програми


(підпис)

Горбовий А.Ю.
(прізвище, ініціали)

11. Завдання до виконання отримав здобувач вищої освіти


(підпис)

Артеменко М. С.
(прізвище, ініціали)

АНОТАЦІЯ

«Розробка вебсайту для планування сімейного бюджету»

(назва кваліфікаційної роботи)

Мета роботи полягає в створенні середовища для надання можливості відслідковування та керування особистими та сімейними фінансовими ресурсами.

Завданнями роботи є:

1. Аналіз нинішнього становища та проблематики керування фінансами у сучасному світі.
2. Дослідження наявних рішень проблеми.
3. Вибір підходящого інструментарію для розробки системи.
4. Реалізації основних компонентів програми які зумовлюють працездатність проекту.
5. Складання рекомендацій щодо вдосконалення та подальшого покращення проекту.

Предметом дослідження є засоби та методи, що необхідні задля реалізації та функціонування проекту й автоматизації процесу управління фінансами.

Об'єктом дослідження виступає автоматизація процесу керування та контролю особистих та сімейних фінансів.

Базою дослідження виступає: методи контролю фінансів та менеджменту особистих грошових ресурсів, бази даних, середовища та методи реалізації програмної логіки.

За результатами дослідження сформульовані рекомендації щодо подальшого розвитку та покращення проекту, а також висновки по проведеній роботі.

Одержані результати можуть бути використані комерційного впровадження додатку контролю особистих та сімейних фінансів.

Кваліфікаційна робота містить 50 сторінок, 21 рисунок, список використаних джерел з 40 найменувань. 2 додатки.

Ключові слова: *фінанси, база даних, модель, контролер, сервіс, запит.*

Рік виконання кваліфікаційної роботи: 2025 р.

Рік захисту роботи: 2025 р.

ANNOTATION

«Website development for family budget planning»

(title of qualification work)

The goal of the work is to create an environment to enable tracking and management of personal and family financial resources

The objectives of the work are:

1. Analysis of the current situation and problems of financial management in the modern world.
2. Research of existing solutions to the problem.
3. Selection of suitable tools for system development.
4. Implementation of the main components of the program that determine the operability of the project.
5. Preparation of recommendations for improvement and further improvement of the project.

The subject of the study is the means and methods necessary for the implementation and functioning of the project and the automation of the financial management process

The object of the research is the automation of the process of managing and controlling personal and family finances.

The research is based on: methods of financial control and management of personal financial resources, databases, environments and methods of implementing program logic.

Based on the results of the study, recommendations were formulated for further development and improvement of the project, as well as conclusions on the work carried out.

The results obtained can be used for the commercial implementation of an application for controlling personal and family finances.

The qualification work contains 50 pages, 21 figures, a list of sources used with 40 names. 2 appendices.

Keywords: finance, database, model, controller, service, query.

Year of completion of qualification work: 2025.

Year of thesis defense: 2025.

Зміст

Вступ	2
Розділ 1. Аналіз сучасного стану обраної задачі.	5
1.1 Аналіз теми управління особистими фінансами.....	5
1.2 Дослідження наявних рішень проблеми.....	8
1.3 Дослідження та підбір основних методів та засобів для розробки проекту.	11
Розділ 2. Практичні результати досліджень.	18
2.1 Розробка бази даних.	18
2.2 Реалізація взаємодії з базою даних.....	22
2.3 Реалізація функціоналу взаємодії.	28
Розділ 3. Аналіз комерційного впровадження додатку та пропозиції щодо удосконалення наявного проекту.	39
3.1 Фінансове обґрунтування доцільності комерційного впровадження додатку.	39
3.2 Пропозиції щодо удосконалення наявного проекту.....	43
Висновки	49
Список використаних джерел.....	51
Додаток А.....	54
Додаток Б	57

Вступ

Ефективне управління грошовими ресурсами та фінансова грамотність – у сучасному світі ці поняття набувають дедалі більшої ваги. Контроль за сімейним та особистим бюджетом стає однією з основних потреб кожної людини. При раціональному використанні особистих фінансових ресурсів, людина отримує можливість досягати поставлених цілей, уникати боргів і забезпечувати стабільність у буденному житті. Разом із цим, швидкий розвиток інформаційних технологій та доступність інтернету створюють нові можливості для автоматизації та спрощення процесів фінансового управління.

Актуальність теми. У наш час мається проблема браку системного підходу до поводження як з особистими так і з сімейними фінансами. Нерідко це призводить до неефективного розподілу та використання наявних грошових ресурсів. Нині наявні на ринку інструменти підходящі для управління фінансами часто можуть являтися складними у використанні. Більшість рішень призначені для керування більш масивними фінансовими операціями, що робить їх менш доступними для регулярного користувача.

Ступінь розробки проблеми при дослідженні актуальної спеціалізованої та наукової літератури, а також статей на дану тему, демонструє про значний інтерес як до питання контролю особистих фінансів так застосування цифрових інструментів для автоматизації й полегшення даного процесу. Проте більшість рішень проблеми зосереджені на більш широкому плануванні контролю бюджетів, які не завжди будуть підходити для регулярного користувача. Це в свою чергу створює необхідність дослідження та розробки рішень які могли б використовуватися для менш спеціалізованих задач.

Мета і завдання дослідження. Головною метою даного дипломного проекту є створення зручного середовища яке надасть можливість керування особистими та сімейними фінансовими ресурсами. Що в свою чергу сприятиме підвищенню фінансової грамотності у користувачів, сприятиме униканню необдуманих витрат та забезпечить раціональне використання наявних коштів.

Задля досягнення поставленої мети виступає необхідність вирішення наступних завдань:

1. Аналіз нинішнього становища та проблематики керування фінансами у сучасному світі.
2. Дослідження наявних рішень проблеми.
3. Вибір підходящого інструментарію для розробки системи.
4. Реалізації основних компонентів програми які зумовлюють працездатність проекту.
5. Складання рекомендацій щодо вдосконалення та подальшого покращення проекту.

Об'єкт дослідження. Об'єктом дослідження виступає автоматизація процесу керування та контролю особистих та сімейних фінансів.

Предмет дослідження. Предметом дослідження є засоби та методи, що необхідні задля реалізації та функціонування проекту й автоматизації процесу управління фінансами.

Методи дослідження. У процесі реалізації проекту було використано:

1. Вивчення актуальної літератури та веб-ресурсів. Огляд наукових публікацій, сучасних статей, що розглядають проблематику управління особистими фінансами.
2. Порівняльний аналіз наявних рішень. Проведено дослідження популярних веб-додатків та мобільних застосунків для обліку особистих фінансів.
3. Було використано методи системного аналізу для визначення методів реалізації проекту.

Наукова новизна полягає у реалізації веб-ресурсу для контролю особистих та сімейних фінансів, що буде не лише надавати можливості для контролю власних грошових ресурсів, а й буде достатньо простим та зручними використанні.

Практичне значення. Створений проект, надає можливості контролю та моніторингу особистого та сімейного бюджету, сприяє поліпшенню фінансової обізнаності та грамотності користувачів. Його використання утворить позитивний вплив на раціональне використання фінансових ресурсів.

Структура роботи. Магістерська робота поділяється на три основні розділи. У першому проведено аналіз проблематики сучасного стану контролю фінансів та наявних рішень проблеми. Другий розділ надає опис етапів розробки веб-додатку, що включає підбір інструментарію для його реалізації, проектування та розробку основних компонентів програми які відповідають за її працездатність. Третій розділ містить рекомендації щодо покращення проекту і його подальшого покращення.

Розділ 1. Аналіз сучасного стану обраної задачі.

1.1. Аналіз теми управління особистими фінансами.

З розвитком ринкових механізмів господарювання, а також із глобалізацією економічних відносин, формування й використання фінансових ресурсів громадянами набуває нового значення. Відбувається процес збільшення можливостей щодо ефективного управління особистих коштів; використання особистих фондів грошових ресурсів; зростає вплив фінансових рішень на загальний стан особистих і суспільних фінансів. При цьому головною метою економічного розвитку є не тільки загальне зростання добробуту суспільства, а й покращення добробуту кожної окремої людини.

До основних факторів, що впливають на процес управління особистими фінансами відносяться:

- національний менталітет, обумовлений історичними факторами;
- соціально-демографічні фактори: рівень освітченості громадян, стан майнових накопичень, вік громадянина, а також гендерна приналежність.

Об'єктивною необхідністю для кожної сучасної людини є ефективне управління особистими фінансами, що приносить не тільки практичну економічну вигоду, а й сприяє фінансовій стабільності. Для забезпечення сталого розвитку економіки країни все більшого значення набуває формування у громадян культури управління фінансами, в тому числі особистого фінансового планування та фінансової грамотності. Процес управління особистими фінансами повинен починатися зі створення особистого фінансового плану. Подібний план являє собою стратегічний підхід до досягнення певних поставлених фінансових цілей, заснований на грамотному використанні доступних фінансових інструментів з урахуванням індивідуальних можливостей і передбачуваних потреб в конкретних умовах[1].

Етапи створення особистого фінансового плану поділяються на наступні декілька кроків:

1. Постановка мети.

Основне завдання фінансового планування - перетворити мрії та бажання на конкретні цілі. Чітко визначена мета, яка має конкретні часові рамки та необхідну кількість коштів, стає конкретним завданням, яке потрібно виконати. Без визначеної мети всі подальші дії не мають сенсу. Важливо розуміти, чого саме ви хочете досягти. Цілі повинні бути чіткими і конкретними, а не абстрактними. Наприклад, якщо ваша мета - заробити мільйон, купити квартиру, машину або організувати подорож, вашим надійним помічником стане бюджетування. Таким чином, основою фінансового планування є раціональний розподіл коштів для досягнення поставленої мети.

2. Оцінка поточного фінансового стану.

Оцінка доходів, витрат, активів і пасивів, а також поточної ринкової ситуації для подальших фінансових розрахунків, щоб, в кінцевому рахунку, показати, досяжні поставлені цілі в даних умовах.

Багато хто також вважає, що їхнє фінансове становище покращиться раз і назавжди, якщо їхні доходи зростуть, адже тоді їм буде достатньо для покриття всіх витрат. Однак існує тенденція, що зі зростанням рівня доходів зростають і витрати. Чим вищий рівень доходу людини, тим менше вона обмежує свої потреби і тим більше витрат вона готова нести для їх задоволення.

При складанні особистого фінансового плану потрібно враховувати не тільки свої цілі і можливості, але і свої потреби, а також адекватність своїх цілей і бажань. Тільки усвідомивши, наскільки продуманими, виправданими і доцільними є витрати, можна оцінити, наскільки далеко ви знаходитесь від своєї мети (або, навпаки, наскільки близькі до неї) і які дії потрібно зробити, щоб досягти бажаного результату. Недостатньо просто виконувати свій особистий бюджет - потрібно робити це ефективно[2-3].

3. Складання особистого бюджету.

Ведучи бюджет, ви не зможете повністю уникнути непотрібних витрат, але звести їх до мінімуму - цілком досяжне завдання. Ретельне планування і контроль за виконанням своїх планів допоможе вам стати більш дисциплінованою, відповідальною і цілеспрямованою людиною.

4. Контроль.

Контролюючи рух грошей, отримується можливість не тільки враховувати окремі доходи і витрати, але й підбивати підсумки балансу. Баланс - це форма бухгалтерського обліку, яка дозволяє оцінити поточний фінансовий стан на певну дату за допомогою доходів і витрат, активів і пасивів.

5. Аналіз отриманого фінансового звіту.

Він допоможе зрозуміти, на скільки реалістичний фінансовий план. Що стосується розбіжності бажань і можливостей, у людини з'являється необхідність зробити вибір на користь одного з двох можливих варіантів подальших дій: або обмежити власні бажання, або збільшити власні можливості.

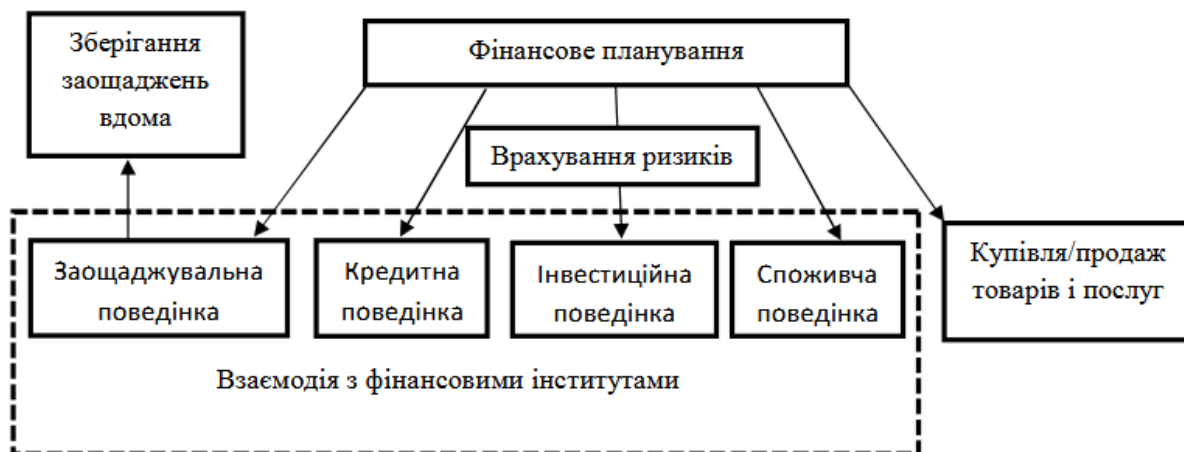


Рис. 1.1 – Схема управління особистими фінансами.

Початкові етапи створення особистого фінансового плану несуть за собою мету наочної демонстрації: гроші для інвестування можна знайти навіть у межах власного бюджету, при умові наявності уміння правильного його контролю. Однак, основна проблема найчастіше полягає не у самій відсутності коштів, а у відсутності достатніх навичок для їх раціонального використання. Фінансове планування охоплює управління всіма доступними ресурсами і засобами для досягнення поставлених цілей і бажань. Опанувавши мистецтво фінансового планування, людина отримує здатність забезпечувати не тільки свої нагальні потреби, а й реалізувати довгострокові прагнення.

Управління особистими фінансами являє собою процес, що потребує постійної уваги і відбувається протягом усього життя. Адже Фінансова грамотність населення несе створення фундаменту не лише для особистого розвитку кожної людини, а й для загального матеріального добробуту громади й суспільства. Сприяє накопиченню капіталу, забезпеченню його захисту та раціонального приросту, що в кінцевому підсумку надає фінансову стабільність і процвітання.

Успішне управління особистими фінансами, а отже, й успішне існування в сучасному світі створює необхідність володіння певним обсягом фінансових знань. Саме тому наявність фінансової грамотності являється ознакою успішності людини. Це вміння, що зумовлює можливість розпоряджатися своїми грошовими ресурсами, а також змогу брати відповідальність за фінансові рішення. Лише володіючи цими навичками, можна побудувати систему раціональної фінансової поведінки, що спрямовується на подолання дефіциту фінансових і економічних ресурсів. Втім нерідко людям звична імпульсивна та нераціональна поведінка у відношенні фінансових питань, і саме від рівня фінансової грамотності залежить здатність побудови коректної фінансової стратегії[4].

Поняття фінансової поведінки в широкому розумінні охоплює всі аспекти діяльності домогосподарств або ж окремих особистостей, які так чи інакше пов'язані з управлінням грошовими коштами, включаючи їх отримання та

витрачання. Дане багатогранне поняття охоплює собою різні види фінансової активності: планування фінансів, накопичення та утримування заощаджень, мінімізація ризиків, створення інвестицій, страхування, а також кредитно-позикова діяльність. Також до фінансової поведінки можна віднести менш формальні дії, такі як: участь у іграх з використанням грошей, проведення покупок або ж продажу товарів чи послуг, що проходять поза межами офіційних фінансових установ. Також відноситься виконання розрахунково-касових операцій. Загалом сукупність даних елементів разом і формує комплексну систему взаємодії громадян із фінансовими ресурсами та інструментами.

1.2 Дослідження наявних рішень проблеми.

Проблема контролю особистих фінансів не нова й існує багато прикладів її рішень. У тому числі сайти й додатки які допомагають людям керувати власними фінансами.

Mint.

Це популярний веб-додаток для управління особистими фінансами, створений у 2006 році Аароном Пацером. Він створений як інструмент для відстеження витрат, доходів і бюджетування. У 2009 році Mint придбала Intuit, компанію, що стоїть за такими фінансовими продуктами, як QuickBooks і TurboTax. Після цього сервіс отримав додатковий розвиток інтеграції з іншими інструментами управління фінансами.

При розробці використовувалися сучасні технології, спрямовані на забезпечення надійності, масштабованості та безпеки. Для створення клієнтської частини програми використовувався JavaScript спільно з такими фреймворками, як React або Angular. Це забезпечило швидкий перегляд інтерфейсу та простоту використання. Для серверної частини використовувалися Python, Ruby або Java, які добре підходять для обробки великих обсягів даних і забезпечують стабільну роботу бекенда[5].

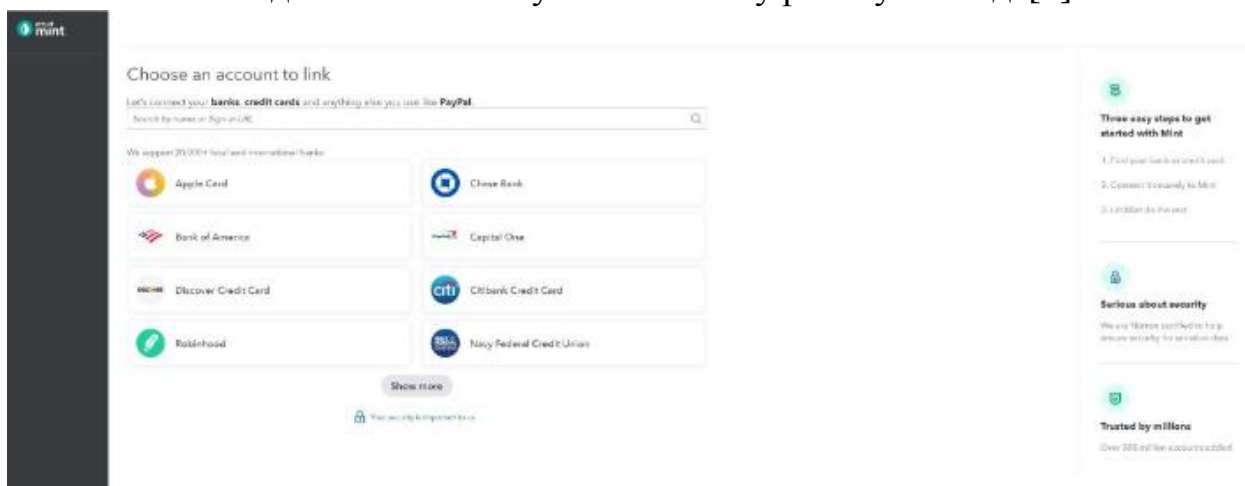


Рис. 1.2 – Додаток Mint.

Бази даних, такі як PostgreSQL або MongoDB, використовувалися для зберігання даних користувачів, включаючи їхні фінансові операції, бюджетні плани та звіти. Інтеграція з банківськими системами відбувалася через API, що дозволяло автоматично імпортувати транзакції з банківських рахунків і кредитних карток користувачів.

Оскільки Mint працює з конфіденційною інформацією, особливу увагу було приділено безпеці. Шифрування SSL/TLS під час передавання використовується для захисту даних, а також методи багатофакторної автентифікації, щоб забезпечити доступ до облікових записів лише авторизованим користувачам. Крім того, для забезпечення максимального рівня довіри було впроваджено політику відповідності стандартам фінансової безпеки, таким як PCI DSS.

YNAB (You Need A Budget)

Додаток був створений в 2004 році американським бухгалтером Джессі Мечамом. Спочатку це був особистий проект Джессі, який він розробив, щоб керувати власними фінансами під час навчання в університеті. Ідея виникла через потребу ефективніше управляти обмеженими ресурсами та уникнути боргів[6].

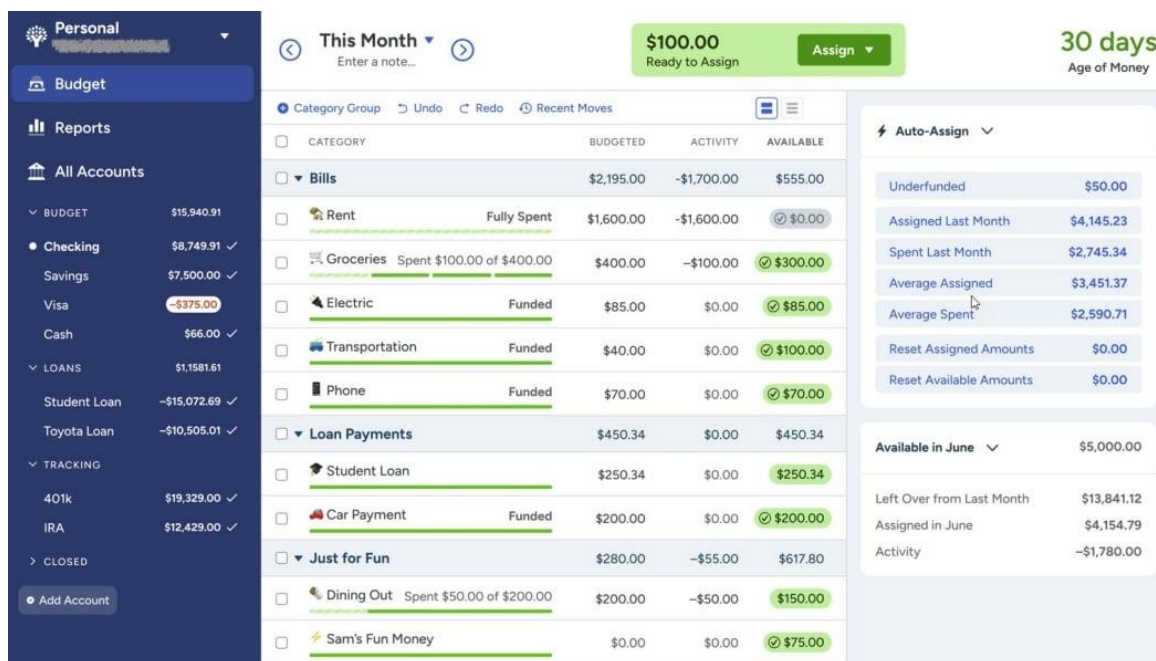


Рис. 1.3 – Додаток YNAB.

Для фронтенду використовувався React.js, який дозволяє створити динамічний і швидкий інтерфейс користувача. Завдяки React.js програма є інтерактивною та забезпечує чудовий досвід роботи з мінімальними затримками взаємодії. React.js також полегшує керування станом програми та повторне використання компонентів.

Бекенд побудований на Ruby on Rails, що забезпечує швидку розробку та підтримку бізнес-логіки програми. Ruby on Rails — один із найпопулярніших

веб-фреймворків, який забезпечує ефективну роботу з базами даних, автоматизацію рутинних процесів і високу продуктивність серверної частини.

Для зберігання даних використовується PostgreSQL, реляційна база даних із потужними функціями для керування складними структурами даних. PostgreSQL було обрано через його надійність, масштабованість і підтримку транзакцій, що важливо для фінансових програм, де дані завжди мають бути узгодженими.

З часом додаток розвивався. На даний момент він пропонує широкий спектр функцій. YNAB інтегрується з банківськими рахунками, автоматично імпортує транзакції, що робить відстеження витрат легким і точним. Користувачі можуть створювати індивідуальні категорії бюджету, а також остаточний план витрат відповідно до своїх потреб. Спільний бюджет дозволяє сім'ям або партнерам координувати свої фінансові плани та досягати своїх цілей одночасно. Крім того, вбудовані звіти та аналітика допомагають зрозуміти, куди йдуть гроші, і визначити можливості для економії.

Goodbudget

Додаток був створений в 2009 році компанією Dayspring Technologies, яка базується в Сан-Франциско, США. Dayspring Technologies спеціалізується на розробці інструментів для підвищення фінансової грамотності та доступу до технологій. Метою Goodbudget було зробити традиційні методи бюджетування доступними в цифровому вигляді, адаптувавши їх до сучасного способу життя.



Рис. 1.4 – Додаток Goodbudget

Goodbudget використовує низку технологій для створення веб-версій, мобільних додатків і синхронізації між пристроями. Angular.js був обраний для веб-інтерфейсу, який забезпечує динамічний та інтерактивний досвід

користувача. Серверна частина (backend) працює на Python з використанням фреймворку Django, що дозволяє швидко розробляти та підтримувати функціональний додаток. База даних побудована на базі PostgreSQL, що гарантує надійне зберігання фінансових даних[7].

Додаток заснований на методі «конверт», який був популярний ще до появи цифрових технологій. У цій техніці гроші розподіляються між віртуальними конвертами, кожен з яких призначений для певної категорії витрат, наприклад, «Їжа», «Транспорт» або «Розваги».

1.3 Дослідження та підбір основних методів та засобів для розробки проекту.

Для розробки за основу побудови проекту було обрано ASP.NET Core.

ASP.NET Core представляє собою високопродуктивну сучасну платформу, що призначається для розробки веб-додатків. Він широко використовується розробниками вирізняючись своєю гнучкістю, ефективністю, а також можливістю підтримки сучасних технологій[8].

Дана платформа була офіційно представлена компанією Microsoft у червні 2016 року. Вона заснована на класичному ASP.NET, що існує з 2002 року. При її створенні розробники Microsoft вирішили повністю переписати платформу, для забезпечення її кросплатформеності, гнучкості та зробити її більш орієнтованою до сучасних вимог, поставлених до розробки веб-додатків. Основною платформою для обговорення та внесення змін до ASP.NET Core є GitHub.

Оглядаючи ключові причин вибору ASP.NET Core для розробки, найбільш вагомою стає його здатність підтримування розробки на різних операційних системах, включаючи Windows, macOS і Linux. До того ж, фреймворк підтримує різноманітні інноваційні підходи до розробки програмного забезпечення, до яких можна віднести: роботу з мікросервісами, контейнеризацію та інтеграцію з хмарними сервісами.

Окрім того, дана структура несе в собі ряд інших переваг:

- Високий ступінь масштабованості;
- Розвинену безпеку;

- Легкість розгортання.

Застосування при розробці ASP.NET Core надає змогу для розробників реалізувати сучасні веб-рішення, що будуть мати змогу задовольняти нинішні потреби бізнесів. Забезпечувати зручність для кінцевого користувача, не нехтуючи високою продуктивністю.

Дане середовище розробки забезпечує:

- Гнучкість при використанні різноманітних середовищ розробки.
- Підтримку сучасних принципів програмування, до прикладу модульність та реалізація залежностей (DI) і RESTful API.
- Інтеграцію з іншими інструментами та платформами, такими як Docker, Kubernetes і Azure.

Основні функції, які забезпечує це середовище:

- ASP.NET Core дозволяє розробляти і запускати програми на різних операційних системах.
- Цей фреймворк продовжує активно розвиватися спільнотою розробників із відкритим доступом до вихідного коду на GitHub.
- ASP.NET Core забезпечує швидкий запуск програми та високу продуктивність завдяки оптимізованому механізму обробки запитів.
- Дозволяє підключати тільки ті бібліотеки, які потрібні для конкретного проекту.
- Фреймворк оптимізований для роботи з хмарними платформами, такими як Microsoft Azure[9].

Під час процесу розробки веб-сайту ASP.NET Core виконує кілька важливих ролей. Фреймворк дозволяє розробляти як монолітні програми, так і створювати масштабні системи. Структура проекту також включає наступні основні моменти:

- Реалізація контролерів. Основне їх завдання полягає в обробці запитів від користувачів.
- Використання моделей. Використовуються для обміну даними між іншими компонентами.

- Видавнича справа. Відповідає за створення HTML-сторінок, які повинні відображатися користувачам.
- Послуги. Відповідає за побудову логіки обробки даних.

Завдяки кросплатформній функціональності платформа дозволяє створювати сайти, які будуть мати спроможність розгортання на серверах Windows, у хмарі або ж навіть на хостингу Linux. Завдяки ж можливості інтеграції з популярними інструментами розробки, такими як Visual Studio і Visual Studio Code, фреймворк не тільки дозволяє працювати з ними, але й надає змогу працювати з проектами найрізноманітнішої складності. ASP.NET Core дозволяє легко створити RESTful API, який можна використовувати для інтеграції з мобільними додатками, сторонніми системами або SPA, такими як Angular, React або Vue.js. Має вбудовану систему ідентифікації для реалізації функцій автентифікації та авторизації. Надає ресурси зручно створювати сайти з реєстрацією користувачів, керуванням ролями та доступом. Забезпечує легку реалізацію багатомовних сайтів шляхом локалізації бібліотек, що може мати особливе значення для глобальних проектів. Підтримці контейнеризації (наприклад, Docker) і хмарних служб сайт на основі ASP.NET Core, дає змогу легкого масштабування за допомогою завантажень.

Фреймворк також пропонує гнучкі механізми перевірки даних, які спрощують перевірку введених користувачем даних. Інтеграція з іншими службами .NET, такими як Dependency Injection, забезпечує легку та ефективну взаємодію між компонентами системи.

Згадані контролери та моделі і їх тісний зв'язок з побудовою архітектури з ASP.NET, дають чудове підґрунтя для використання архітектурної моделі MVC. Вона призначена для розробки програмного забезпечення та розділення його на три основні компоненти: Model, View і Controller.

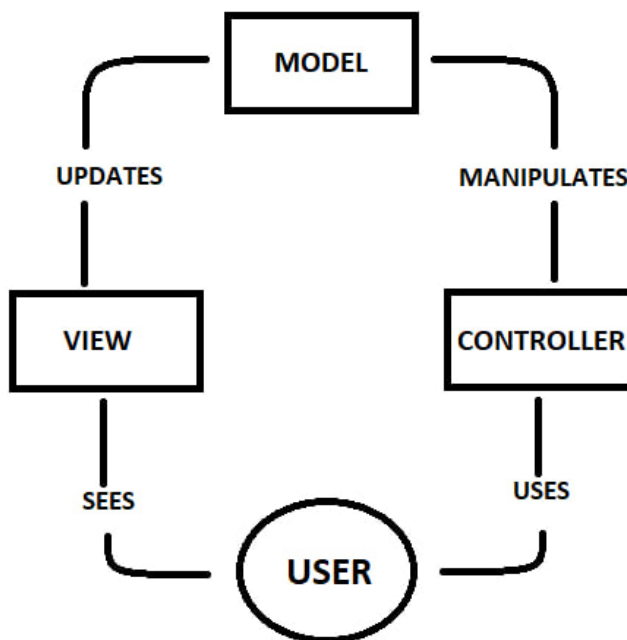


Рис. 1.5 – Загальна схема побудови моделі MVC.

Дана модель була вперше запропонована Тригве Рейнског у 1979 році під час роботи над Smalltalk-80 у дослідницькому центрі Xerox PARC. З тих пір ця модель набула широкого використання в розробці програмного забезпечення. Особливо в веб-додатках. Основна ідея методу полягає в розділенні програми на три ключових компоненти: Модель, Представлення, Контролер[10].

1. Компонент «Модель» відповідає за представлення даних, а також за реакцію на команди контролерів і відповідну зміну по відношенню до них. Цей компонент відповідає за керування даними, перевірку даних, бізнес-логіку та взаємодію з базою даних. Основні функції включають: керування даними програми, впровадження бізнес-логіки та перегляд оновлень у зв'язку зі зміною даних.

2. Компонент «Вид» забезпечує коректне відображення даних для користувача, відповідно до зміни моделей. Відповідає за відображення інформації для користувачів. Отримання даних з моделі та представлення їх у зрозумілій формі. Функції компонента включають: відображення даних, управління реалізацією логіки інтерфейсу користувача.

3. Компонент «Контролер» обробляє дії користувача з метою коректної зміни даних щодо нього. Контролер є посередником між користувачами і системою.

Його завдання полягає в прийнятті запитів від користувачів, обробці та їх зв'язках з Model і View для виконання дій. Його основні функції: керування запитами від користувачів, виклик методів моделі для обробки даних і вибір представлення для відображення результату.

Провести опис механізму роботи MVC, можна розділивши його на такі кроки:

1. Користувач робить запит. Наприклад, клацає посилання або вводить URL-адресу в браузері.
2. Контролер отримує запит. Він аналізує запит і вирішує, яку дію вжити.
3. Взаємодія з моделлю. Контролер викликає модель для отримання даних або виконання бізнес-логіки.
4. Модель передає дані контролеру, а контролер створює представлення для відображення результату.
5. Відповісти користувачеві. View генерує HTML, який надсилається користувачеві у відповідь на запит.

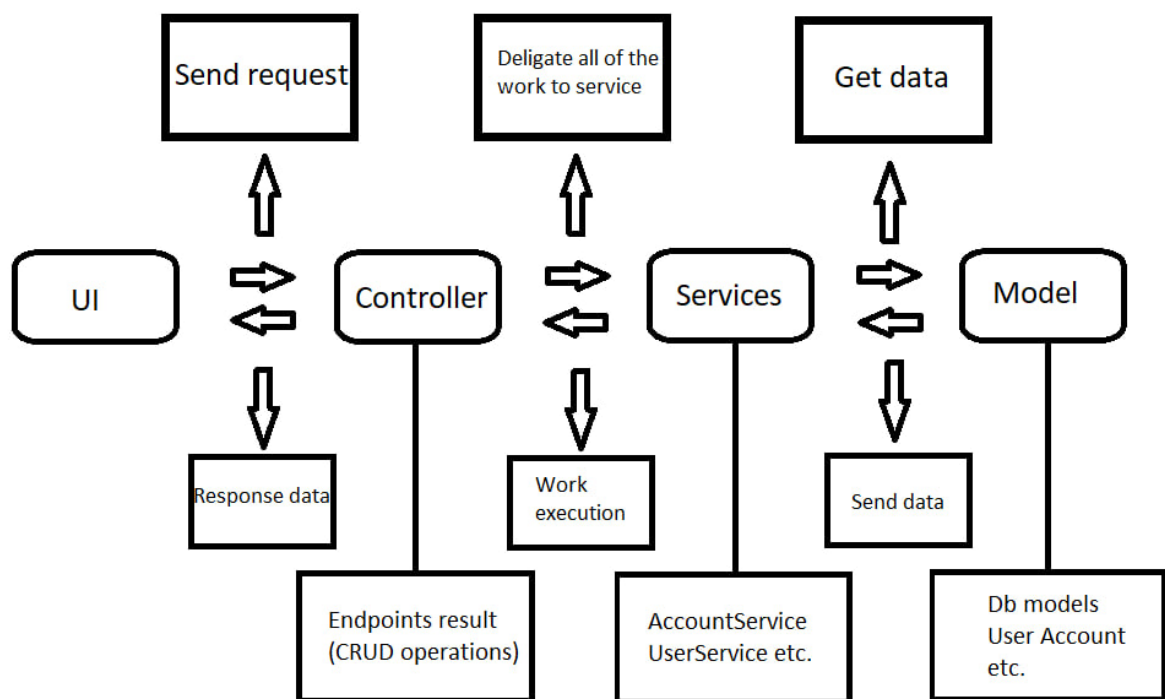


Рис.1.6 – Схема механізму взаємодії компонентів MVC.

MVC працює за принципом чіткого розподілу обов'язків між трьома його компонентами: моделлю, представленням і контролером. Все починається з

взаємодії користувача з системою через інтерфейс. Користувач надсилає запит, наприклад, натиснувши посилання або ввівши URL-адресу в браузері. Цей запит передається до контролера, який є свого роду координатором у моделі MVC.

«Контролер» аналізує запит, визначає, яка бізнес-логіка потрібна для його обробки, і викликає модель. «Модель», у свою чергу, відповідає за всю роботу з даними: вона може отримувати їх з бази даних, виконувати обчислення або оновлювати наявні дані. Після завершення цих операцій «Модель» повертає дані «Контролеру». Отримавши ці дані, він передає їх «View», компоненту, який відповідає за рендеринг. «View» формує інтерфейс, використовуючи отримані дані, і генерує HTML-код, який потім надсилається назад у браузер користувача. В результаті користувач отримує динамічний і зрозумілий візуальний результат на основі свого початкового запиту.

MVC забезпечує чітке розділення логіки програми, що дозволяє зосередити її окремі частини на конкретних завданнях. Цей підхід робить додатки більш організованими, легшими в обслуговуванні та легшими для масштабування.

Переваги MVC можна продемонструвати відштовхуючись від його основної ідеї — поділу системи на три компоненти з різними ролями. Такий поділ надає програмі більше ефективності у розвитку та підтримці положення. Можна виділити основні переваги:

1. Model, View і Controller забезпечують завдання, які зменшують складність програми.
2. Простота тестування: окремі компоненти MVC можна тестувати незалежно.
3. Масштабованість: MVC легко розширюється завдяки своїй модульності.
4. Повторне використання: такі компоненти, як модель, можна повторно використовувати в різних частинах надбудови.
5. Покращення команди розробників: різні частини системи можуть розроблятися одночасно різними командами.

Шаблон MVC широко використовується у веб-розробці через його здатність організовувати складні системи та забезпечувати гнучкість і

масштабованість. Популярні веб-фреймворки, до яких належать ASP.NET Core MVC, Ruby on Rails, Spring MVC, Laravel, реалізують цю модель[11].

Використання MVC у веб-розробці дозволяє створювати надійні, масштабовані та добре організовані програми, які легко підтримувати та розробляти.

Розділ 2. Практичні результати досліджень.

2.1 Розробка бази даних.

Для створення бази даних необхідної для функціонування проекту була вибрана MSSQL.

Microsoft SQL Server представляє собою реляційну систему управління базами даних (RDBMS). Розроблена корпорацією Microsoft, вона являється однією з найпопулярніших і найбільш потужних систем для роботи з даними, як в корпоративних, так і в комерційних середовищах. MSSQL здатна підтримувати широкий спектр функціоналу, що включає в себе: управління транзакціями, реплікацію даних, здатність до інтеграції з іншими продуктами Microsoft і багаторівневу безпеку.

Архітектура MSSQL побудована орієнтуючись саме на забезпечення надійності, продуктивності і гнучкості у роботі з даними. Основу системи представляє Database Engine. Він відповідає за виконання всіх операцій із базою даних, включаючи обробку запитів, управління транзакціями, індексацію та зберігання даних. Це центральний компонент, який забезпечує основні функціональні можливості MSSQL[12-13].

Універсальним вибором її робить саме здатність інтеграції з іншими продуктами Microsoft: Azure, Power BI, Excel і в особливості Visual Studio та .NET.

На рисунку 2.1 зображено основну структуру бази даних, на якій зазначені зв'язки між таблицями.

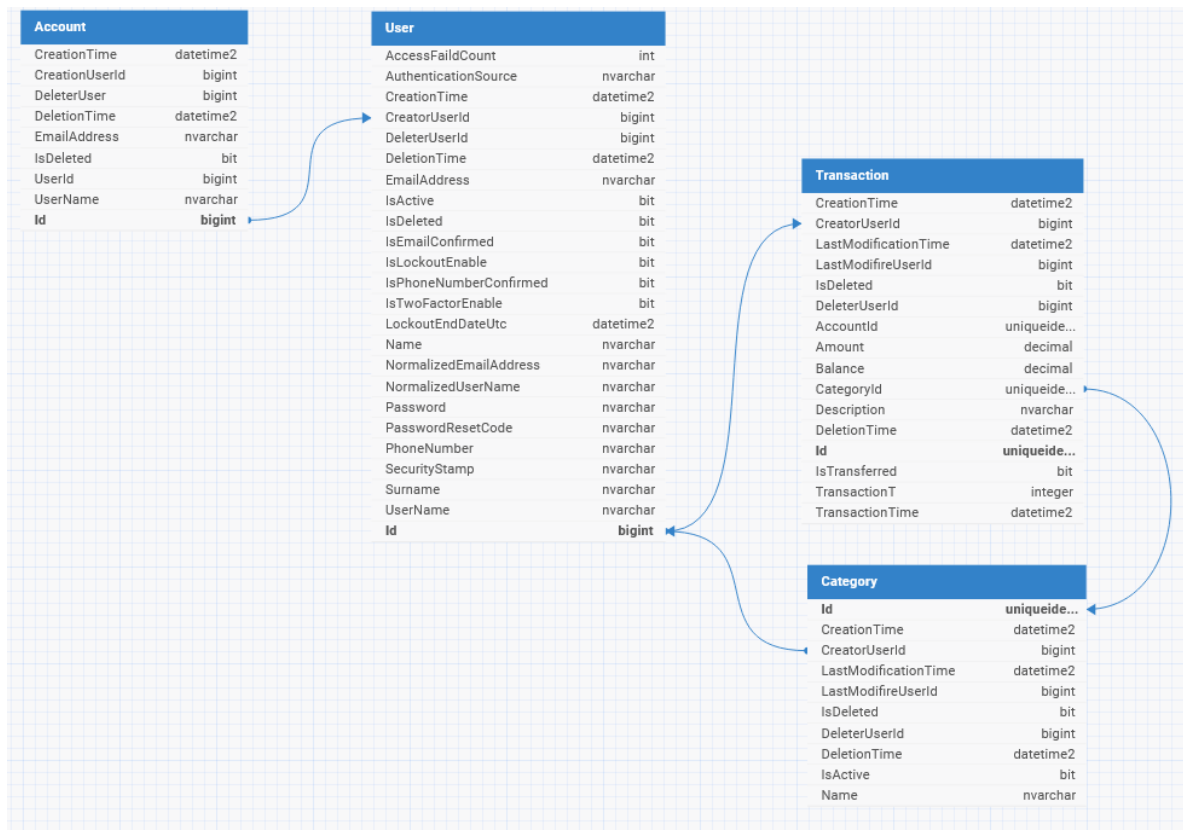


Рис. 2.1 – Структура бази даних.

З рисунку можна винести:

- Таблиця Account має зовнішній ключ UserId, він пов'язує її з таблицею User за допомогою поля Id. Це дозволяє зберігати прив'язку кожного облікового запису до конкретного користувача, що в свою чергу забезпечує чітку асоціацію між записами.
- Таблиця ж Transaction має зовнішні ключі UserId і CategoryId, які пов'язують її з таблицями User та Category через відповідні поля Id. Це надає змогу асоціювати транзакції як із конкретними користувачами, так і з певними категоріями, що забезпечує структуровану організацію фінансових даних у базі. Таким чином, зв'язки між таблицями допомагають ефективно моделювати відносини між об'єктами системи та забезпечують цілісність даних.

Приклад SQL-скрипту, який генерується для створення таблиці Users:

Лістинг 1 - SQL-скрипту, для створення таблиці Users

```
CREATE TABLE [Users] (
    [Id] INT NOT NULL PRIMARY KEY IDENTITY,
    [Name] NVARCHAR(MAX) NOT NULL,
    [Email] NVARCHAR(MAX) NULL,
    [CreatedAt] DATETIME2 NOT NULL
);
```

Необхідний SQL-скрипт формується у наслідок роботи міграцій. Процес його генерації починається зі змін у моделях даних. При додаванні нового класу або зміні вже існуючого, при додаванні нової властивості, Entity Framework відстежує ці зміни за допомогою механізму відстеження моделей – snapshot, який зберігає поточний стан моделей даних. При запуску створення міграції, відбувається порівняння поточного стану моделей із попереднім збереженням. На основі цього порівняння й генерується код міграції, який в свою чергу створить код для створення нової таблиці з усіма визначеними полями.

Приклад SQL-скрипту, який генерується для створення таблиці Transactions із зовнішнім ключем:

Лістинг 2 - SQL-скрипт для створення таблиці Transactions із зовнішнім ключем.

```
CREATE TABLE [Transactions] (
    [Id] INT NOT NULL PRIMARY KEY IDENTITY,
    [Amount] DECIMAL(18, 2) NOT NULL,
    [UserId] INT NOT NULL,
    CONSTRAINT [FK_Transactions_Users_UserId] FOREIGN KEY
([UserId]) REFERENCES [Users] ([Id]) ON DELETE CASCADE
);
```

Задля забезпечення цілісності й логічності в базі даних використовуються внутрішній ключ (Primary Key, PK) та зовнішній ключ (Foreign Key, FK). Вони реалізують структурування даних, а також забезпечують ефективне управління інформацією у пов'язаних таблицях[14].

Собою вони представляють:

- Внутрішній ключ (Primary Key) — унікальний ідентифікатор кожного запису у таблиці. Він має три основні характеристики: завжди має унікальне значення; не може містити порожні значення (NULL); використовується для ідентифікації рядка таблиці.
- Зовнішній ключ (Foreign Key) представляє собою поле або ж набір полів у таблиці, що посилається на внутрішній ключ іншої таблиці.

Зв'язок встановлюється через зовнішній ключ, що додається в таблицю, яка залежить від іншої. Це надає змогу прив'язувати кожен запис залежної таблиці з відповідним записом головної таблиці.

Відстеження належного оформлення та створення таблиць можливе за допомогою Microsoft SQL Server Management Studio. Собою вона представляє середовище для управління, конфігурації та адміністрування Microsoft SQL Server. Вона надає розробникам, адміністраторам баз даних (DBA) та іншим фахівцям доступ до всіх функцій SQL Server через графічний інтерфейс, а також спрощує виконання більш складних операцій, до яких входять: створення, адміністрування, оптимізація та моніторинг таблиць.

На рисунку 2.2 зображено приклад реалізації таблиць в Microsoft SQL Server Management Studio:

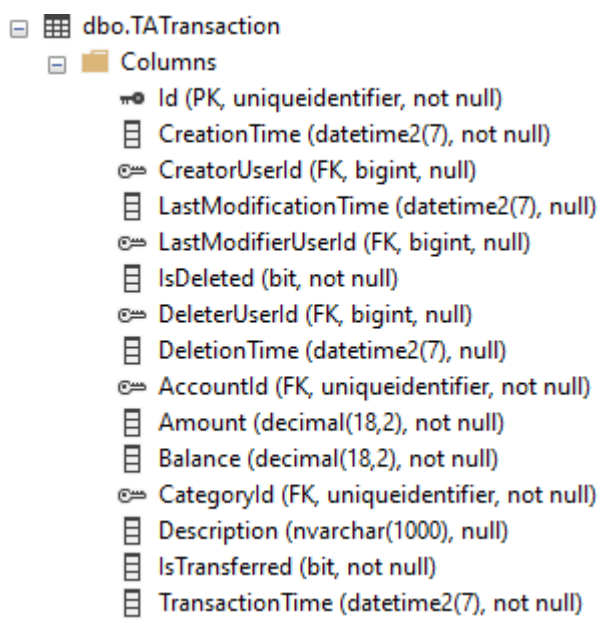


Рис. 2.2 – Оформлення таблиць в Microsoft SQL Server ManagementStudio.

На зображенні представлена структура таблиці Transaction у базі даних. Демонструється структура таблиці, в якій передбачено збереження необхідної інформації: сум транзакцій, балансу, а також метаінформації, та зв'язків між іншими таблицями бази даних. Вона слугує основою задля збереження інформації про всі основні фінансові операції. Її структура має забезпечити гнучкість, цілісність даних і легкість їх використання[15].

2.2 Реалізація взаємодії з базою даних.

Ври використанні ASP.NET та C#, взаємодія з базою даних в багатьох випадках виконується за допомогою Entity Framework Core. Саме даний фреймворк спрощує та забезпечує належну взаємодію програмного коду та реляційних баз даних.

Представляє собою він об'єктно-реляційний маппер або ж ORM, що використовується для платформи .NET. Основна його значимість полягає у тому, що він забезпечує інструментами для роботи з даними у вигляді об'єктів. Це мінімізує необхідність написання SQL-запитів. Дозволяючи створювати класи, які представляють собою таблиці бази даних, отримується можливість працювати з ними як зі звичайними об'єктами мови C#[18-19].

На рисунку 2.3 зображено приклад реалізації таблиці за допомогою створення класу.

```

namespace TestApp.App.Accounts
{
    [Table("TAAccount")]
    public class Account : FullAuditedEntity<Guid, User>
    {
        public const int MaxNameLength = 50;

        [ForeignKey("AccountTypeId")]
        public virtual AccountType AccountType { get; set; }

        public virtual int AccountTypeId { get; set; }

        [ForeignKey("CurrencyId")]
        public virtual Currency Currency { get; set; }

        public virtual int CurrencyId { get; set; }

        [Required]
        [MaxLength(MaxNameLength)]
        public virtual string Name { get; set; }
    }
}

```

Рис. 2.3– Реалізація таблиці за допомогою створення її класу.

Подібний підхід до роботи з базами даних називається CodeFirst. Він використовується при роботі з Entity Framework, а його суть полягає у тому що розробка починається з написання коду моделей даних у вигляді класів у C#. На відміну від стандартного методу коли, спочатку відбувається створення бази даних та її структури вручну, з якими потім ведеться робота, у випадку з CodeFirst розробник описує структуру даних у коді, а Entity Framework автоматично генерує відповідну структуру бази даних на основі цього коду, що дає змогу на пряму працювати з базою даних, корегувати таблиці або вносити корективи[20].

Головною перевагою Code First являється можливість використання об'єктно-орієнтованих принципів для проектування моделі, що звільняє від

необхідності думати про низькорівневі аспекти реляційних баз даних, такі як ручне створення SQL-запитів.

Приклад визначення моделі User при використанні підходу:

Лістинг 3 - Визначення моделі User.

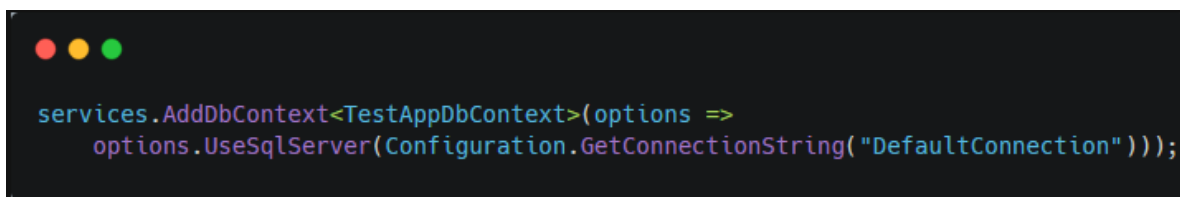
```
public class User
{
    public int UserId { get; set; }
    public string UserName { get; set; }
    public string Email { get; set; }
    public int? Age { get; set; }
    public DateTime CreatedAt { get; set; } = DateTime.Now;
}
```

Таким чином отримується можливість більш зручної маніпуляції даними, адже всі дії з ними можна реалізовувати напряму в коді.

Code First також надає змогу тонко налаштовувати моделі через анотації даних або API Fluent. До прикладу, можна використовувати атрибути [Required], [MaxLength(50)] чи [ForeignKey] (як показано на Рис. 2.3) для встановлення необхідних правил для колонок, таких як обов'язковість, максимальна довжина чи зовнішній ключ[16-17].

Подібна зручність розробки отримується саме за допомогою використання Entity Framework.

Окрім створення моделі, є ще декілька важливих моментів, що забезпечують роботу:

A screenshot of a code editor with a dark background and light-colored text. The code is in C# and shows the configuration of a DbContext. It includes the 'services.AddDbContext' method call with a lambda expression for options, and the 'options.UseSqlServer' method call with a connection string. The code is as follows:

```
services.AddDbContext<TestAppDbContext>(options =>
    options.UseSqlServer(Configuration.GetConnectionString("DefaultConnection")));
```

Рис. 2.4 - Налаштування підключення до бази даних

Дане підключення визначає, як програма підключається до бази даних і може містити інформацію про тип бази (SQL Server, PostgreSQL тощо), адресу сервера, ім'я бази, логін і пароль.

TestAppDbContext — це клас контексту, який є точкою доступу до бази даних, а UseSqlServer в свою чергу вказує на використання SQL Server як СУБД (система управління базою даних)[21].

На рисунку 2.5 зображено написання контексту бази даних:



```
public class TestAppDbContext : DbContext
{
    public DbSet<User> Users { get; set; }
    public DbSet<Transaction> Transactions { get; set; }

    protected override void OnModelCreating(ModelBuilder modelBuilder)
    {
        modelBuilder.Entity<User>().HasKey(u => u.Id);
        modelBuilder.Entity<Transaction>().Property(t => t.Amount).IsRequired();
    }
}
```

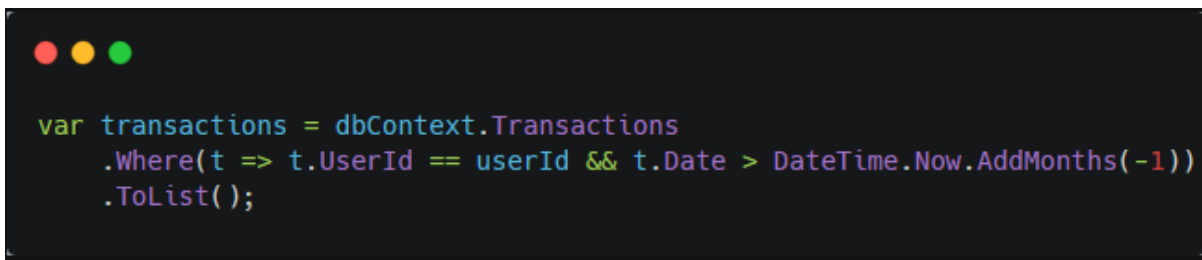
Рис. 2.5 – Контекст бази даних.

Клас DbContext у EntityFramework є ключовим елементом, який встановлює зв'язок між програмою і базою даних. В даному випадку це TestAppDbContext, що успадковується від DbContext.

Контекст бази містить:

- DbSet для кожної таблиці бази даних, він представляє собою колекцію об'єктів у вигляді моделей.
- OnModelCreating — налаштування моделі даних через метод, який використовується для конфігурації правил мапування.

Приклад виконання запитів до бази даних:

A screenshot of a code editor window with a dark background. The code is written in C# and uses LINQ to query a database. It starts with 'var transactions = dbContext.Transactions', followed by a lambda expression filter '.Where(t => t.UserId == userId && t.Date > DateTime.Now.AddMonths(-1))', and ends with '.ToList();'. The code is color-coded: 'var' is blue, 'transactions' is green, 'dbContext.Transactions' is purple, '.Where' is blue, 't' is green, 't.UserId' is purple, 'userId' is green, '&&' is blue, 't.Date' is purple, '>' is blue, 'DateTime.Now' is purple, 'AddMonths' is blue, '(-1)' is green, and '.ToList()' is blue.

```
var transactions = dbContext.Transactions
    .Where(t => t.UserId == userId && t.Date > DateTime.Now.AddMonths(-1))
    .ToList();
```

Рис. 2.6 – використання LINQ для запитів

Language Integrated Query або ж LINQ представляє собою спосіб запитів, що в даному прикладі використовується для звернення до бази даних через об'єкт контексту[23].

В даному випадку, продемонстрований на Рис.2.6 запит повертає всі транзакції користувача за останній місяць.

Напевно найбільш значущим механізмом який забезпечує коректну роботу виступають міграції.

Міграції надають можливість автоматизації процесу створення, оновлення, а також керування базою даних, відповідно до змін у програмному коді.

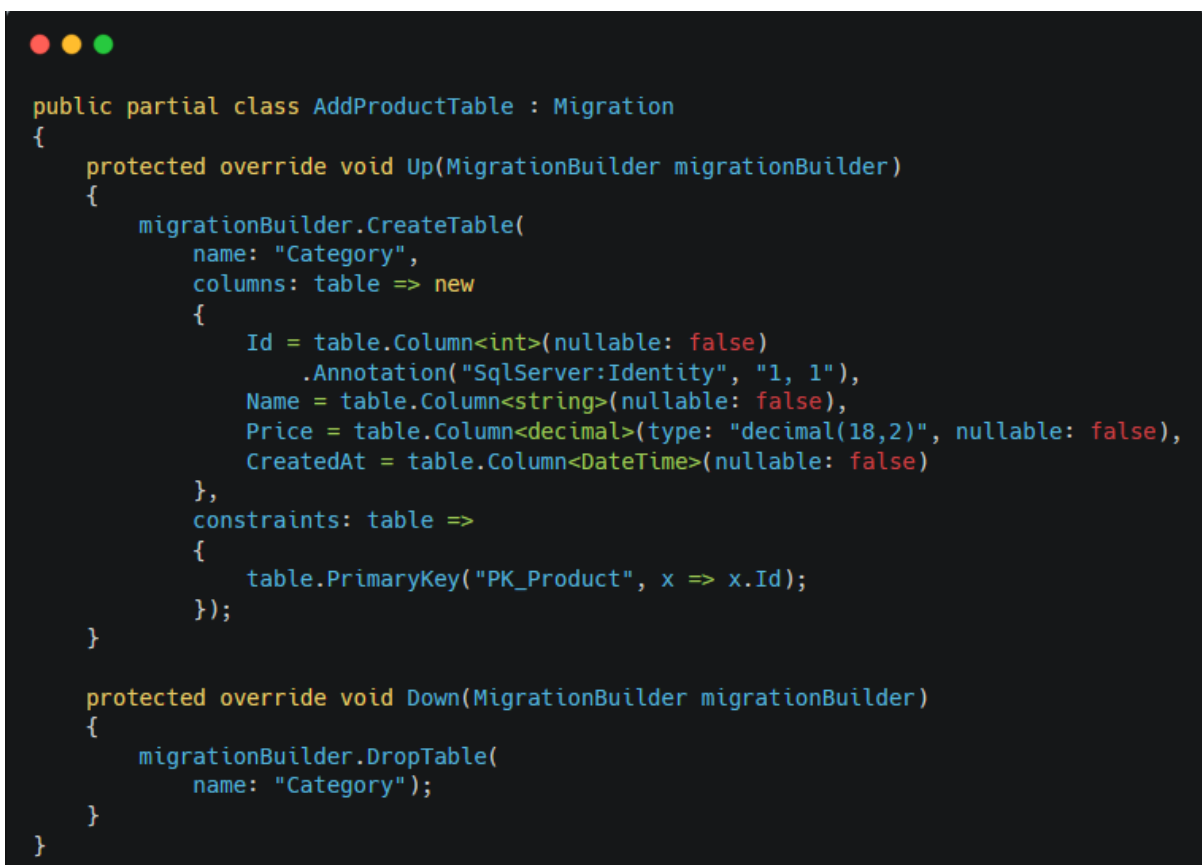
Основна мета міграцій:

Синхронізація об'єктно-орієнтованої моделі даних у коді програми з реляційною структурою таблиць бази даних. Задля забезпечення автоматизації процесу написання SQL-запитів, які необхідні для створення або модифікації таблиць. Забезпечення цілісності між кодом і базою даних.

При роботі з моделями даних у C#, виникає необхідність додати або змінити властивості класів, які представляють таблиці бази. Міграції автоматично генерують SQL-скрипти для застосування цих змін у реальній базі даних. Саме це й дозволяє працювати з даними на рівні об'єктів у коді, залишаючи завдання, пов'язані зі структурою бази, на Entity Framework[24].

Процес роботи з міграціями:

- Визначення моделі даних;
- Додавання міграції;
- Генерація файлу (Рис. 2.7) з логікою за допомогою команди `dotnet ef migrations add`;
- Виконання SQL-запитів описаних в міграції.



```
public partial class AddProductTable : Migration
{
    protected override void Up(MigrationBuilder migrationBuilder)
    {
        migrationBuilder.CreateTable(
            name: "Category",
            columns: table => new
            {
                Id = table.Column<int>(nullable: false)
                    .Annotation("SqlServer:Identity", "1, 1"),
                Name = table.Column<string>(nullable: false),
                Price = table.Column<decimal>(type: "decimal(18,2)", nullable: false),
                CreatedAt = table.Column<DateTime>(nullable: false)
            },
            constraints: table =>
            {
                table.PrimaryKey("PK_Product", x => x.Id);
            });
    }

    protected override void Down(MigrationBuilder migrationBuilder)
    {
        migrationBuilder.DropTable(
            name: "Category");
    }
}
```

Рис. 2.7 – Згенерований у файлі міграції код.

Даний код реалізується за допомогою двох методів:

- Метод `Up`. Він визначає які самі зміни будуть внесені в базу даних.
- Метод `Down`. У цьому методі визначається яким саме чином можна буде скасувати зміни. Він здатен видалити таблицю `Category` за необхідності.

Міграції надають можливість відкочування бази даних до попереднього стану, за необхідності скасування внесених попередньо змін. Це робиться за допомогою команди `dotnet ef database update <назва_міграції>`, де `<назва_міграції>` вказує на певну точку в історії змін.

SQL згенерований міграцією:

Лістинг 4 – SQL-скрипт для створення таблиці Category.

```
CREATE TABLE [Category] (
    [Id] INT NOT NULL IDENTITY(1,1),
    [Name] NVARCHAR(MAX) NOT NULL,
    [Description] NVARCHAR(MAX) NULL,
    [CreatedAt] DATETIME2 NOT NULL,
    CONSTRAINT [PK_Category] PRIMARY KEY ([Id])
);
```

В результаті буде створено таблицю Category, що складається зі стовпців:

- Id — Унікальний ідентифікатор категорії;
- Name — Назва категорії (Вона є обов'язковою);
- Description — Опис категорії (Вона є необов'язковою);
- CreatedAt — Дата створення категорії.

Тепер дана таблиця стає доступною для збереження та управління даними, без необхідності ручного написання SQL-запитів.

2.3 Реалізація функціоналу взаємодії.

Важливим аспектом, який є основою будь-якої реляційної бази даних, є CRUD-операції. Вони й використовуються для управління даними й взаємодії з ними. У контексті Entity Framework і підходу Code First, CRUD-операції реалізуються через взаємодію з контекстом бази даних (DbContext) та моделями, які представляють таблиці бази даних[25].

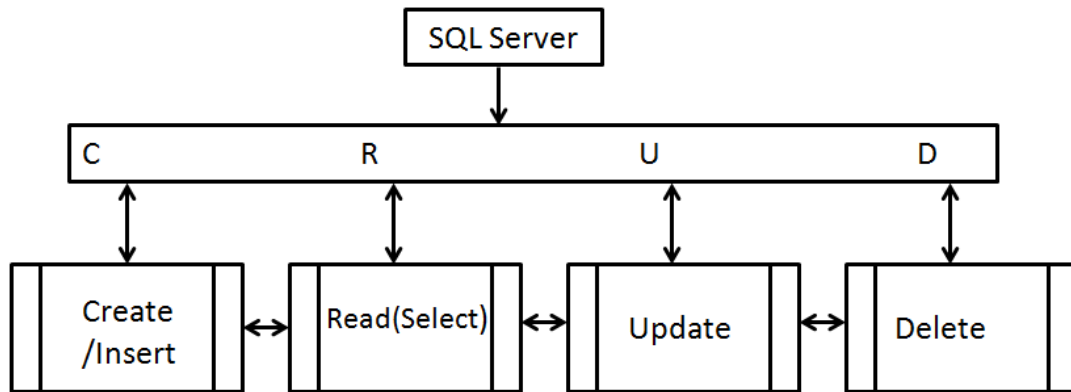


Рис. 2.8 – CRUD-операції.

Вони представляють собою:

1. Створення (Create). Ця операція дозволяє додавати нові записи до бази даних. В Entity Framework це реалізується шляхом створення нового об'єкта відповідного класу моделі, додавання його до відповідного набору (DbSet) у контексті бази даних і подальшого виклику методу SaveChanges[26].

Приклад створення транзакції:

Лістниг 5 – Реалізація створення транзакції.

```

var newTransaction = new TATransaction
{
    Amount = 100.50m,
    Balance = 500.00m,
    CategoryId = 1,
    Description = "Оплата за інтернет",
    IsTransferred = false,
    TransactionTime = DateTime.Now
};
context.TATransactions.Add(newTransaction);
context.SaveChanges();
  
```

2. Читання (Read). Дана операція призначена для отримання (або ж зчитування) даних. В Entity Framework для цього застосовуються LINQ-запити або стандартні методи, такі як Find, FirstOrDefault, ToList[27].

Приклад зчитування всіх транзакцій за категорію:

Лістинг 6 – Зчитування всіх транзакцій категорії.

```
var transactions = context.TATransactions
    .Where(t => t.CategoryId == 1)
    .ToList();

foreach (var transaction in transactions)
{
    Console.WriteLine($"ID:    {transaction.Id},    Amount:    {transaction.Amount},
    Description: {transaction.Description}");
}
```

3. Оновлення (Update). Призначені для внесення змін у базу даних. Виконується це шляхом віднаходження необхідного об'єкта, внесення до нього змін, і завершального виклику SaveChanges[28].

Приклад оновлення транзакції:

Лістинг 7 – Оновлення транзакції.

```
var transactionToUpdate = context.TATransactions.Find(1);
if (transactionToUpdate != null)
{
    transactionToUpdate.Amount = 150.00m;
    transactionToUpdate.Description = "Оновлена транзакція";
    context.SaveChanges();
}
```

4. Видалення (Delete). Дана операція використовується для видалення записів з бази. Знаходиться необхідний об'єкт, передається до методу Remove, а потім викликається SaveChanges[29].

Приклад видалення транзакції:

Лістинг 8 – Видалення транзакції.

```
var transactionToDelete = context.TATransactions.Find(1);
if (transactionToDelete != null)
{
    context.TATransactions.Remove(transactionToDelete);
    context.SaveChanges();
}
```

Таким чином CRUD-операції є основою взаємодії з базою даних, забезпечуючи створення, отримання, оновлення та видалення даних.

Для коректної виконання операцій, при побудові системи на основі моделі MVC, необхідна реалізація контролерів та сервісів[34].

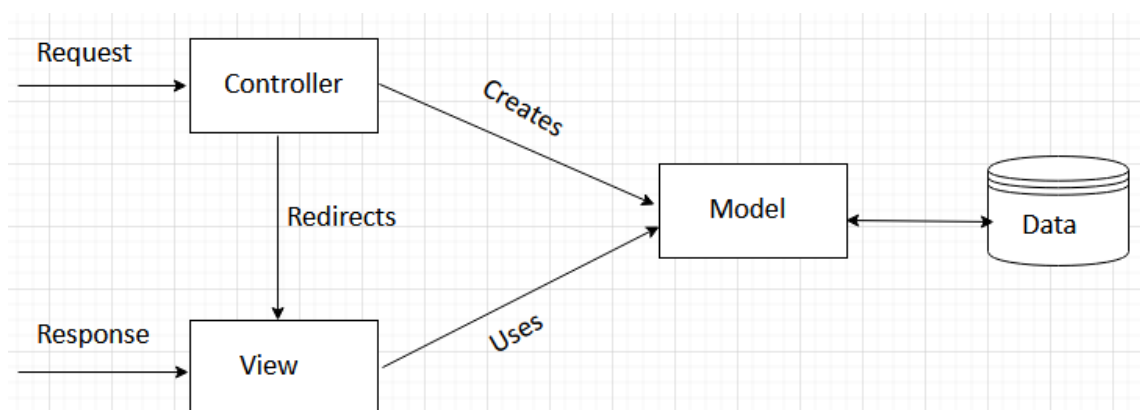


Рис. 2.9 – Схема циклу взаємодії складових моделі MVC.

Для реалізації даного функціоналу, застосовується Boilerplate. Даний шаблон, забезпечує набір функцій для роботи з структурою MVC. Він чудово підходить для роботи з Entity Framework й забезпечує інтеграцію з ним. У випадку ASP.NET, він дозволяє розробляти багатошарову архітектуру з відокремленням клієнтської логіки від бізнес логіки, що дозволяє підвищити масштабованість і зручність підтримки коду[32-33].

Контролери представляють собою прошарок між клієнтом і сервером і відповідають за їх взаємодію. Дані компоненти обробляють HTTP-запити,

виконують базову перевірку, а також викликають відповідні сервіси для виконання бізнес-логіки та повертають відповіді клієнту. Вони представляють з себе точкою входу для запитів, що надходять до серверної частини програми.

Їх робота заключається у взаємодії із запитами GET, POST, PUT, DELETE. Вони реалізують кінцеві точки (endpoints) API. Визначають, який запит і яким чином має бути опрацьований. До прикладу: при отриманні запиту на отримання списку транзакцій, контролер викликає метод відповідного сервісу, який повертає дані[30-31].

Контролери несуть відповідальність лише за управління запитами та відповідями, уникаючи реалізації бізнес-логіки. Таким чином забезпечується чіткий поділ між рівнем презентації (запити/відповіді) та рівнем бізнес-логіки (сервіси).



```

namespace YourProject.Controllers
{
    [ApiController]
    [Route("api/[controller]")]
    public class TransactionController : ControllerBase
    {
        private readonly ITransactionAppService _transactionService;

        public TransactionController(ITransactionAppService transactionService)
        {
            _transactionService = transactionService;
        }

        [HttpGet]
        public async Task<ActionResult<List<TransactionDto>>> GetAll()
        {
            var transactions = await _transactionService.GetAllAsync();
            return Ok(transactions);
        }

        [HttpGet("{id}")]
        public async Task<ActionResult<TransactionDto>> GetById(long id)
        {
            var transaction = await _transactionService.GetByIdAsync(id);
            if (transaction == null)
            {
                return NotFound($"Transaction with ID {id} not found.");
            }
            return Ok(transaction);
        }
    }
}

```

Рис. 2.10 – Реалізація контролера для запиту GET.

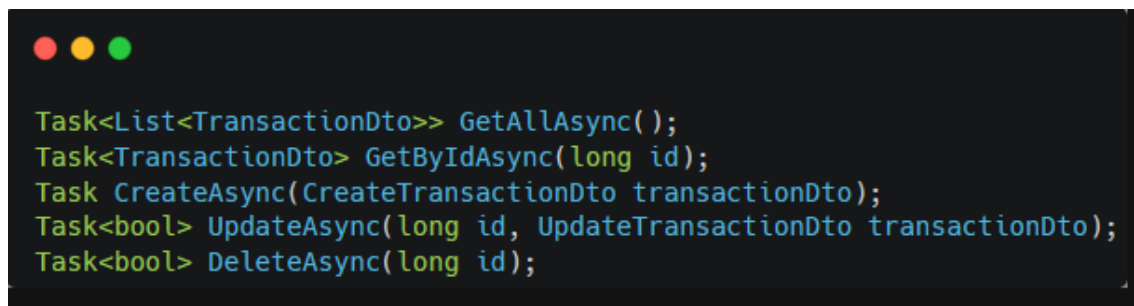
У конструкторі впроваджено (inject) сервіс ITransactionAppService, що й відповідає за бізнес логіку. Таким чином контролер має змогу передавати запити окремо сервісу, не виконуючи бізнес-логіку самостійно.

Контролер використовує наступні методи:

- GetAll. Він викликає метод сервісу для отримання списку всіх транзакцій. Після чого повертає список транзакцій у форматі JSON.
- GetById. даний метод отримує транзакцію за її ідентифікатором. У випадку транзакція не знайдена, повертає HTTP 404.
- Create. Приймає об'єкт DTO із даними для створення нової транзакції. Викликає сервісний метод задля створення запису.
- Update. Використовується для оновлення існуючої транзакції за її ідентифікатором. У випадку коли транзакції не існує, повертає HTTP 404.
- Delete. Видаляє транзакцію за її ідентифікатором. У разі відсутності транзакції повертає HTTP 404.

Data Transfer Object або ж DTO використовується для реалізації процесу передавання даних між клієнтом і сервером[39].

В кінці, контролер викликає методи сервісу для виконання бізнес-логіки.



```
Task<List<TransactionDto>> GetAllAsync();
Task<TransactionDto> GetByIdAsync(long id);
Task CreateAsync(CreateTransactionDto transactionDto);
Task<bool> UpdateAsync(long id, UpdateTransactionDto transactionDto);
Task<bool> DeleteAsync(long id);
```

Рис. 2.11 – Інтерфейс сервісу.

На рисунку 2.11 представлено інтерфейс сервісу, що визначає базові CRUD-операції:

- Отримання всіх транзакцій;
- Отримання транзакції за ID;
- Створення нової транзакції;
- Оновлення існуючої транзакції;
- Видалення транзакції.

На рисунку 2.12 зображена реалізація сервісу з методом для отримання всіх транзакцій.

```

public class TransactionAppService : ITransactionAppService
{
    private readonly IRepository<TATransaction, long> _transactionRepository;
    private readonly IRepository<Category, long> _categoryRepository;

    public TransactionAppService(
        IRepository<TATransaction, long> transactionRepository,
        IRepository<Category, long> categoryRepository)
    {
        _transactionRepository = transactionRepository;
        _categoryRepository = categoryRepository;
    }

    public async Task<List<TransactionDto>> GetAllAsync()
    {
        var transactions = await _transactionRepository.GetAllIncluding(t => t.Category).ToListAsync();

        return transactions.Select(t => new TransactionDto
        {
            Id = t.Id,
            Amount = t.Amount,
            Description = t.Description,
            TransactionTime = t.TransactionTime,
            IsTransferred = t.IsTransferred,
            CategoryName = t.Category?.Name
        }).ToList();
    }
}

```

Рис. 2.12 – Реалізація сервісу.

У конструкторі сервісу впроваджується (inject) репозиторії для роботи з таблицею Transaction та таблицею Category. За допомогою цього отримується можливість отримувати та обробляти дані з бази.

Методи які реалізує сервіс:

- Метод GetAllAsync. Отримує всі транзакції разом із даними про категорії за допомогою GetAllIncluding. Після чого повертає список DTO-об'єктів, які містять дані для клієнта.
- Метод GetByIdAsync. Виконує пошук транзакції за її ID. У випадку коли транзакція не знайдена, повертає null. Перетворює об'єкт транзакції в DTO перед поверненням.
- Метод CreateAsync. Він створює новий об'єкт транзакції з використанням даних з CreateTransactionDto. Після чого додає нову транзакцію в репозиторій через InsertAsync.
- Метод UpdateAsync. Виконує пошуку існуючої транзакції за ID. У випадку коли транзакція знайдена, оновлює її поля на основі даних з UpdateTransactionDto. Викликає метод UpdateAsync для збереження змін.
- Метод DeleteAsync. Видаляє транзакцію з бази даних, при умові що вона існує.

Переваги такої реалізації:

1. Чітке розділення обов'язків: Логіка роботи з даними зосереджена в сервісах, що спрощує підтримку і модифікацію.
2. Масштабованість: У разі зміни бізнес-логіки її можна змінити лише в сервісі, не зачіпаючи інші шари програми.
3. Повторне використання: Методи сервісу можна використовувати в інших частинах програми, наприклад, у фонових завданнях чи інших контролерах.
4. Легкість тестування: Сервіси можна легко тестувати незалежно від контролерів і бази даних.

Авторизація і автентифікація являють собою ключові елементи безпеки у веб-додатках. Вони використовуються для перевірки особи користувача та визначення його доступу до ресурсів або функціоналу.

- Автентифікація (Authentication) — це процес перевірки особи користувача. Наприклад, перевірка правильності пари логін/пароль.
- Авторизація (Authorization) — це процес визначення, чи має автентифікований користувач право доступу до певних ресурсів або функцій.

У рамках ASP.NET Core ці процеси реалізуються за допомогою JWT (JSON Web Tokens), Cookie Authentication, або інших механізмів[37].

Процес автентифікації та авторизації:

1. Користувач надсилає свої дані для входу. Для прикладу, використовуючи форму логіну (логін і пароль) або OAuth-токен. Сервер отримує ці дані через API-метод контролера.
2. Сервер проводить перевірку даних користувача. Логін і пароль перевіряються щодо збережених у базі даних записів. У разі успішної перевірки сервер генерує токен (наприклад, JWT) або створює сесію.
3. Сервер повертає токен клієнту. Токен зберігається на клієнтській стороні (наприклад, у localStorage або cookies).
4. Клієнт використовує токен для доступу до захищених ресурсів. У кожному запиті клієнт додає токен в заголовок Authorization.
5. Сервер перевіряє токен і надає доступ. Якщо токен валідний і користувач має відповідні права, сервер виконує запит.

Код налаштування JWT:

Лістинг 9 – Налаштування JSON Web Tokens.

```
public class App
{
    public void ConfigureServices(IServiceCollection services)
```

```

{
    services.AddAuthentication(JwtBearerDefaults.AuthenticationScheme)
        .AddJwtBearer(options =>
        {
            options.TokenValidationParameters = new TokenValidationParameters
            {
                ValidateIssuer = true,
                ValidateAudience = true,
                ValidateLifetime = true,
                ValidateIssuerSigningKey = true,
                ValidIssuer = "https://domain.com",
                ValidAudience = "https://domain.com",
                IssuerSigningKey = new
SymmetricSecurityKey(Encoding.UTF8.GetBytes("Key"))
            };
        });
    services.AddAuthorization();
    services.AddControllers();
}
public void Configure(IApplicationBuilder app, IWebHostEnvironment env)
{
    app.UseRouting();
    app.UseAuthentication();
    app.UseAuthorization();
    app.UseEndpoints(endpoints => endpoints.MapControllers());
}
}

```

Таким чином відбувається налаштування параметрів перевірки токена, а саме: секретний ключ, термін дії, перевірка видавця і аудиторії.

Код контролера для автентифікації:

Лістинг 10 – Реалізація контролера для автентифікації.

```

[ApiController]
[Route("api/[controller]")]
public class AuthController : ControllerBase
{
    private readonly IUserAppService _userAppService;
    public AuthController(IUserAppService userAppService)
    {

```

```

    _userAppService = userAppService;
}

[HttpPost("login")]
public async Task<IActionResult> Login([FromBody] LoginDto loginDto)
{
    var user = await userAppService.AuthenticateAsync(loginDto.Username,
loginDto.Password);
    if (user == null)
    {
        return Unauthorized("Invalid credentials.");
    }
    var token = GenerateJwtToken(user.Username, user.Id);
    return Ok(new { Token = token });
}

private string GenerateJwtToken(string username, long userId)
{
    var securityKey = new
SymmetricSecurityKey(Encoding.UTF8.GetBytes("Key"));
    var credentials = new SigningCredentials(securityKey,
SecurityAlgorithms.HmacSha256);

    var claims = new[]
    {
        new Claim(JwtRegisteredClaimNames.Sub, username),
        new Claim("UserId", userId.ToString()),
        new Claim(JwtRegisteredClaimNames.Jti, Guid.NewGuid().ToString())
    };
    var token = new JwtSecurityToken(
        issuer: "https://domain.com",
        audience: "https://domain.com",
        claims: claims,
        expires: DateTime.Now.AddHours(1),
        signingCredentials: credentials
    );

    return new JwtSecurityTokenHandler().WriteToken(token);
}
}

```

Даний контролер виконує дві основні функції:

- Перевіряє облікові дані.
- Якщо облікові дані відповідають вимогам, відбувається генерація токена JWT, який уже повертається клієнту.

Після того як відпрацював контролер, викликається DTO, для того аби передати дані щодо автентифікації.

Подібний підхід несе за собою певні переваги:

- По-перше – безпека. Маючи обмежений термін дії, використання токенів ускладнює несанкціонований доступ.
- По-друге – масштабованість. JWT представляють собою чудовий варіант для мікросервісної архітектури.
- По-третє – гнучкість. Використовуючи claims, не зіставить труднощів реалізувати авторизацію за ролями.

Розділ 3. Аналіз комерційного впровадження додатку та пропозиції щодо удосконалення наявного проекту.

3.1 Фінансове обґрунтування доцільності комерційного впровадження додатку.

Для обґрунтування доцільності комерційного впровадження додатку найкращим рішенням буде проведення розрахунків оцінки витрат на реалізацію та прибутків після впровадження. За допомогою подібного моделювання процесу витрат на розробку, можна оцінити доцільність впровадження.

Загальна вартість розробки:

$$E_t = E_1 + E_2$$

Витрати на розробку:

$$E_1 = E_{ds} + E_{fr} + E_{bk} + E_{ceo}$$

Де:

E_{ds} – Найм фріланс-дизайнерів;

Завдання полягає в створенні UI/UX дизайну для сайту.

При орієнтовному часі, необхідному на розробку в 2 місяці, із середньою ставкою в \$30 або ж 1260 грн/год, а також робочих година – 160 год/місяць, отримаємо наступні витрати:

$$E_{ds} = 1260_{\text{грн}} * 160_{\text{год}} * 2_{\text{м}} = 403200_{\text{грн}}$$

E_{fr} – Найм фронтенд-розробників;

Основне завдання полягає в розробці інтерфейсу сайт, який був створений дизайнером. Орієнтований час, необхідний на розробку – 4 місяці. Середня ставка \$35 або ж 1470 грн/год, при робочих годинах – 160 год/місяць.

$$E_{fr} = 1470_{\text{грн}} * 160_{\text{год}} * 4_{\text{м}} = 940800_{\text{грн}}$$

E_{bk} – Найм бекенд-розробників для розширення функціоналу;

Основна задача полягає в розробці нового та розширенні існуючого функціоналу додатку. Час необхідний на розробку – 5 місяців. Оплата – \$40 або ж 1680 грн/год, при 160 год/місяць робочих годин.

$$E_{ds} = 1680_{\text{грн}} * 160_{\text{год}} * 5_{\text{м}} = 1344000_{\text{грн}}$$

E_{ceo} – Найм проект-менеджерів.

До основних задач входить керування процесу розробки та роботи команди, а також побудова стратегії просування. Тривалість найму – 6 місяців. Оплата – \$50 або ж 2100 грн/год, при 160 год/місяць.

$$E_{ds} = 2100_{\text{грн}} * 160_{\text{год}} * 6_{\text{м}} = 2016000_{\text{грн}}$$

Таким чином:

$$E_1 = 403200_{\text{грн}} + 940800_{\text{грн}} + 1344000_{\text{грн}} + 2016000_{\text{грн}} = 4704000_{\text{грн}}$$

Витрати на впровадження:

$$E_2 = E_h + E_{ads} + E_l + E_{dm}$$

Де:

E_h - Витрати на хостинг;

Для підрахунків взято середню вартість VPS-хостингу – \$50, що дорівнює 2100 грн/місяць. При тривалості хостингу в 12 місяців, витрати вийдуть:

$$E_h = 2100_{\text{грн}} * 12_{\text{м}} = 25200_{\text{грн}}$$

E_{ads} – Реклама;

Для реклами можна взяти просування в Google Ads або ж Facebook Ads. Бюджет витрат на місяць взято – 63000 грн. Продовжуваність рекламної кампанії – 6 місяців.

$$E_{ads} = 63000_{\text{грн}} * 6_{\text{м}} = 378000_{\text{грн}}$$

E_l – Юридична сторона витрат, сертифікація;

До цього пункту включаються такі витрати як: реєстрація компанії, витрати на ліцензії, а також підготовку документів. Витрати мають одноразовий характер.

$$E_h = 84000_{\text{грн}}$$

E_{dm} – Реєстрація домену та SSL-сертифікат.

Загальні вартості для реєстрації домену, а також отримання сертифікату зіставляють:

$$E_h = 840_{\text{грн}} + 4200_{\text{грн}} = 5040_{\text{грн}}$$

Таким чином:

$$E_2 = 25200_{\text{грн}} + 378000_{\text{грн}} + 84000_{\text{грн}} + 5040_{\text{грн}} = 537600_{\text{грн}}$$

Загальні витрати на розробку зіставляють:

$$E_t = E_1 + E_2 = 4704000_{\text{грн}} + 537600_{\text{грн}} = 5196240_{\text{грн}}$$

Розрахунок прогнозу доходів:

$$R_t = N_u * ARPU * 12_{\text{м}}$$

Де:

N_u – Кількість користувачів;

$ARPU$ – Середній дохід на користувача на місяць.

Визначення точки беззбитковості:

Дана точка вказує необхідну кількість часу задля компенсації початкових витрат.

$$T_{bk} = \frac{E_t}{D_m}$$

Де:

E_t – Загальні витрати;

D_m – Щомісячний дохід.

Оцінка рентабельності:

$$I = \frac{D_t - E_t}{E_t} * 100\%$$

D_t – загальний дохід за певний період.

Для проведення розрахунків взято наступні умови:

- За умов успішно проведеної рекламної кампанії, кількість користувачів на перший місяць: 1200;
- Оглядова кількість користувачів за рік: 5000;
- Для зручності обчислення взято лінійний ріст користувачів на протязі 12 місяців;
- Платна підписка на сервіс складатиме \$5 або ж 210 грн., а оплата починатиме зніматися з 4-го місяця.

За обраних умов приріст користувачів зіставляє:

$$\Delta K = \frac{5000 - 1200}{12} = 316.67 \text{ користувачів/місяць}$$

Річний дохід (Сума доходів за місяць):

$$D_t = 0 + 0 + 0 + 451500 + 518070 + 584430 + 651000 + 717570 + 783930 + 850500 + 917070 + 105000 = 7350000 \text{ грн.}$$

Кількість доходів на сьомий місяць становить: 122805 USD. Таким чином точка беззбитковості буде досягнута на восьмому місяці.

Рентабельність:

$$I = \frac{D_t - E_t}{E_t} * 100\% = \frac{7350000 - 5196240}{5196240} * 100\% \approx 41.39\%$$

Висновок: При комерційній реалізації додатку очікуване досягнення точки беззбитковості відбувається на восьмий місяць впровадження додатку, а відсоток рентабельності його впровадження зіставляє $\approx 41.39\%$. Таким чином даний аналіз демонструє доцільність ймовірного комерційного впровадження додатку.

3.2 Пропозиції щодо удосконалення наявного проекту.

Розроблений проект управління фінансовими даними є ефективним інструментом для організації бюджету, відстеження витрат і створення фінансових планів, що виконує поставлені перед ним задачі. Однак, необхідно враховуючи зростаючі потреби користувачів і сучасні технологічні можливості, є потенціал для подальшого вдосконалення проекту. Ці вдосконалення можуть значно підвищити функціональність, зручність використання та конкурентоспроможність системи.

Одним із перспективних напрямків варіантів покращення є впровадження функції «Тенетів» — можливість вести спільний бюджет з різних рахунків. Така функція дозволить декільком користувачам співпрацювати в рамках одного фінансового плану[40].

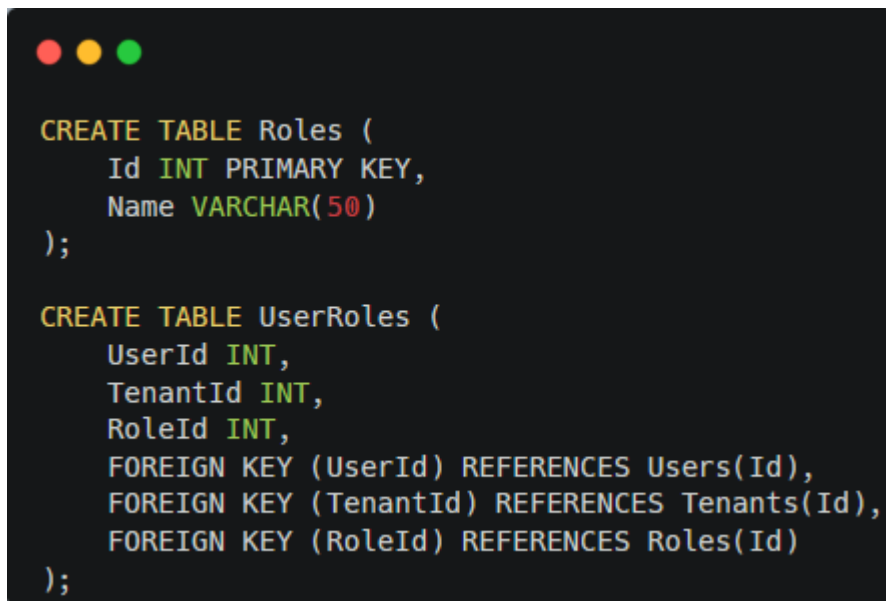
Даний багатокористувацький доступ представляє собою незалежні групи користувачів. Їм надана можливість, в просторі однієї системи спільно використовувати певні ресурси.

Реалізація даного функціоналу несе необхідність розробки архітектури бази даних, яка була б адаптована під неї. Зазвичай використовуються два основні підходи:

- Створення однієї бази даних, яка б вміщала всіх тенантів, а дані б розділялися за виданим тенату ідентифікатором. До прикладу це може бути звичайне додавання до таблиці стовпця-вказівника на відповідний тенант, такого як TenantId. Даний спосіб загалом являється найпоширенішим і часто застосовуваним.
- Реалізація окремих баз даних для кожного тената. При такому підході, дані окремих тенантів будуть зберігатися в окремих базах. Таким чином це надає максимальну ізоляцію, проте сильно збільшує складність управління.

При використанні тенетів реалізується механізм, задача якого управляти наданими ролями і відповідними доступами. Створюється поділ на ролі: адміністратор і учасник. Першому надається можливість додавати або ж видаляти користувачів, а також вносити зміни до налаштувань. Другому

надається доступ лише до перегляду та редагуванню даних, проте не до управління тенантом.



```
CREATE TABLE Roles (  
    Id INT PRIMARY KEY,  
    Name VARCHAR(50)  
);  
  
CREATE TABLE UserRoles (  
    UserId INT,  
    TenantId INT,  
    RoleId INT,  
    FOREIGN KEY (UserId) REFERENCES Users(Id),  
    FOREIGN KEY (TenantId) REFERENCES Tenants(Id),  
    FOREIGN KEY (RoleId) REFERENCES Roles(Id)  
);
```

Рис. 3.1 – Приклад реалізації ролей через використання бази даних.

Управління тенетами реалізується через розробку відповідного API, яке буде включати в собі функціонал, що зможе забезпечити:

- Створення тенантів. Генерація запису і додавання початкових даних.
- Додавання нових користувачів. Нові користувачі додаються до тенанта, а саме таблиці і їм надається відповідний TenantId.
- Реалізація внесення змін до тенанта. Редагування правил для спільних даних, а також зміна ролей, наданих користувачам.

```

public IActionResult AddUserToTenant(int tenantId, int userId)
{
    var userRole = new UserRole
    {
        UserId = userId,
        TenantId = tenantId,
        RoleId = _context.Roles.First(r => r.Name == "Participant").Id
    };
    _context.UserRoles.Add(userRole);
    _context.SaveChanges();
    return Ok("User added successfully");
}

```

Рис. 3.2 – Приклад коду реалізації додання користувача до тенанту.

Реалізація подібного покращення, значно розширить функціональність додатку, а також збільшить зручність використання для груп користувачів. Також подібний функціонал може розширити цільову аудиторію, привабивши корпоративних користувачів.

Наступним важливим етапом удосконалення є розробка сучасного та зручного інтерфейсу користувача. Під час роботи над проектом основна увага приділялася саме роботі з базою даних та бекендом. Надання ж інтуїтивно зрозумілого та зручного інтерфейсу значно покращить користувацький досвід.

Враховуючи потреби як новачків, так і досвідчених користувачів, інтерфейс повинен мати зручний, інтуїтивно зрозумілий та бути адаптивним. Для його реалізації є низка можливих варіантів фреймворків та мов програмування, використання яких буде доцільним: Agile.js, React.js або Vue.js. Вони дозволяють створити динамічний інтерфейс з мінімальною підтримкою взаємодії.

Крім того, доцільним буде інтегрувати можливість відстеження транзакцій з різних платіжних систем, таких як PayPal, Stripe, Revolut або Google Pay. Це дозволить автоматично імпортувати транзакції користувачів в систему, що значно спростить фінансовий облік.

Реалізується подібна можливість через підключення API подібних платіжних систем. Основою виступає процес обробки запитів до даних API. Кожна система надає свій перелік функцій які підтримують взаємодію із сервісом.

Загальні кроки для реалізації:

1. Обирається необхідна платіжна система, після чого відбувається реєстрація розробника і створення облікового запису для можливості доступу до необхідного API. В багатьох випадках платіжні системи надають середовище тестування для перевірки інтеграції. Після цього надається унікальний API-ключ або токен, що й використовується для запитів.
2. Обробка транзакцій за допомогою API. Для прикладу: застосовуючи API PayPal з'являється можливість отримання інформації про історію транзакцій, що включає дату, тип, суму та інше; при використанні Stripe API, отримується доступ до списку платежів, відшкодувань, а також періодичних оплат. Відбувається це шляхом надсилання запитів від сервера до відповідних функцій API, і отримання від них JSON-об'єктів з даними, з якими потім відбувається обробка та запис до бази даних.
3. Забезпечення безпеки даних користувачів. Для цього впроваджується шифрування токенів доступу, використовується HTTPS для всіх виконуваних запитів, а також відбувається регулярне оновлення ключів API. Також система має містити механізми, які будуть забезпечувати відновлення з'єднання на випадок тимчасової недоступності API.

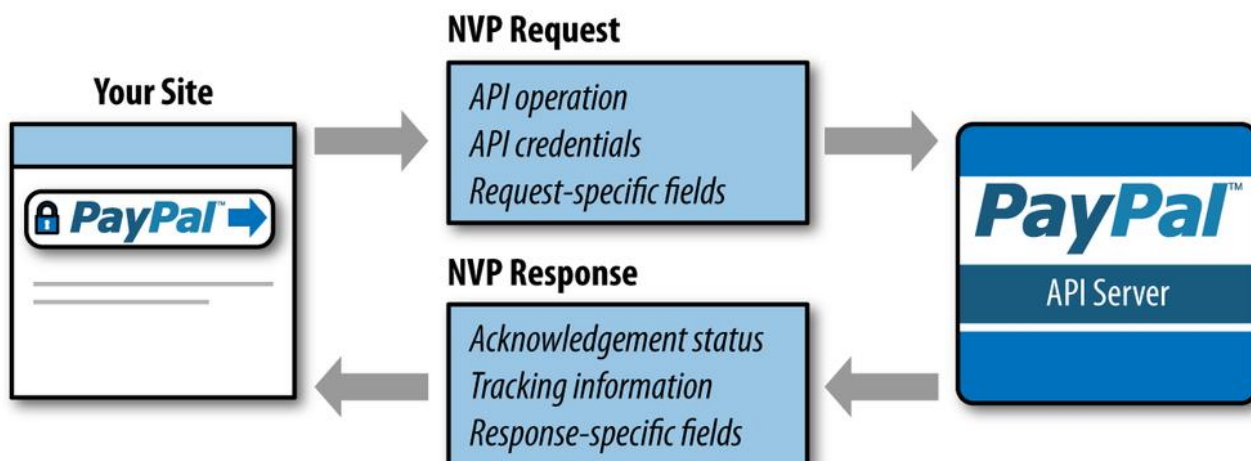


Рис. 3.3 – Загальна схема принципу роботи р PayPal API.

Така функція підвищить загальну цінність та зручність роботи з додатком і дозволить користувачам отримувати в одному місці актуальну інформацію про всі свої витрати.

Останньою пропозицією покращення є розширення можливостей за рахунок реалізації мобільної версії.

Для реалізації існує два шляхи:

1. Реалізація можливості адаптивності для мобільних пристроїв. В загальному це представляє собою реалізацію інтерфейсу, що буде коректно функціонувати й відповідно відображатися на різних типах пристроїв. Дана практика є дуже поширеною, адже значно збільшує зручність та підвищує задоволеність користувацького досвіду. І в загальному є більш простим варіантом.
2. Розробка окремого мобільного додатку. Це більш трудомісткий процес, адже хоч розробка й базується на використанні вже існуючої бази даних та реалізованому функціоналі, проте процес адаптації під мобільні пристрої все ще має обширний об'єм робіт, що включають: проектування UI/UX дизайну (інтерфйсу), розробка бекенду, інтеграція з вже існуючим функціоналом та інше. Проте це компенсується значним покращенням користувацького досвіду.

Наявність мобільного додатку надасть змогу користувачам відслідковувати свої фінанси в будь-який час і в будь-якому місці. Реалізувати

подібне рішення можна за допомогою таких інструментів, як Flutter або React Native. Їх використання дозволить скоротити зусилля, що витрачаються на розробку та підтримку. Функціональний мобільний додаток може включати перегляд транзакцій, створення нових записів, сповіщення про стан бюджету та підтримку синхронізації з веб-версією.

Загалом ці пропозиції спрямовані на розширення можливостей системи, підвищення її зручності для користувачів та забезпечення відповідності сучасним вимогам ринку. Інтеграція з платіжними системами, впровадження спільного доступу до бюджету, розробка сучасного інтерфейсу та мобільного додатку зроблять проект більш зручним у використанні та привабливим для більш широкої аудиторії користувачів.

Висновки

В результаті проведеної роботи було розроблено додаток управління фінансами, що виконує поставлені перед ним задачі, такі як організація бюджету і відстеження витрат, В ході реалізації проекту було проведено комплексний аналіз проблематики теми управління фінансами та досліджено існуючі рішення. Було розроблено архітектуру системи, реалізовано базу даних і бекенд,

Розроблений додаток забезпечує реалізацію основних процесів по контролю фінансів, включаючи відстеження витрат та доходів, можливість нотування транзакцій та планування.

Наукова новизна роботи. Полягає у реалізації ресурсу для контролю особистих та сімейних фінансів, заточеного під потреби рядового користувача, та з потенціалом, для подальшого масштабування й розвитку функціоналу.

Практичне значення. Створений проект, надає можливості контролю та моніторингу особистого та сімейного бюджету, сприяє поліпшенню фінансової обізнаності та грамотності користувачів. Його використання утворить позитивний вплив на раціональне використання фінансових ресурсів.

У ході проведення роботи, були вирішені наступні завдання:

1. Проаналізовано сучасний стан та проблематику управління особистими фінансами.
2. Проведено дослідження існуючих додатків та наявних рішень проблеми.
3. Було підібрано підходящий інструментарій та модель архітектури для розробки системи.
4. Було реалізовано працездатність основних компонентів програми.
5. Були підібрані та запропоновані рекомендації щодо вдосконалення та подальшого покращення проекту.

На основі проведеної роботи були сформульовані рекомендації щодо подальшого розвитку та покращення додатку:

- Впровадження функції «Тенетів» — можливість вести спільний бюджет з різних рахунків.

- Надання інтуїтивно зрозумілого та зручного для користувачів інтерфейсу.
- Додання можливості відстеження транзакцій з різних платіжних систем.
- Розширення можливостей за рахунок реалізації мобільної версії.

Таким чином, проведене дослідження несе за собою як практичну, так і теоретичну цінність. Закладає основу для подальшого вдосконалення додатку управління фінансами. Біло проведено фінансовий розрахунок, що демонструє відсоток рентабельності його комерційного впровадження, який зіставляє $\approx 41.39\%$ що демонструє доцільність використання та розвитку додатку.

Список використаних джерел

1. Ways to manage your personal finances. *darrenbaine.com*. URL: <https://darrenbaine.com/ways-to-manage-your-personal-finances/> (date of access: 05.01.2025).
2. Generation-specific perceptions of financial literacy and digital solutions. IEEE Xplore. URL: <https://ieeexplore.ieee.org/abstract/document/9780717> (date of access: 05.01.2025).
3. Financial literacy in the digital age. *wiley.com*. URL: <https://onlinelibrary.wiley.com/doi/full/10.1111/joca.12510> (date of access: 05.01.2025).
4. Digital financial literacy and its determinants: an empirical evidences from rural india - european scholar. Welcome to European Scholar - European Scholar. URL: <http://article.publish4promo.com/id/eprint/1107/> (date of access: 05.01.2025).
5. Mint. Budget Tracker & Planner | Free Online Money Management | Mint. URL: <https://mint.intuit.com/> (date of access: 05.01.2025).
6. Actual vs YNAB | actual budget documentation. Actual | Actual Budget Documentation. URL: <https://actualbudget.org/blog/2024-07-01-actual-vs-ynab/> (date of access: 05.01.2025).
7. Goodbudget. URL: <https://goodbudget.com/> (date of access: 05.01.2025).
8. Overview of ASP.NET core. Microsoft Learn: Build skills that open doors in your career. URL: <https://learn.microsoft.com/en-us/aspnet/core/introduction-to-aspnet-core?view=aspnetcore-9.0> (date of access: 05.01.2025).
9. Choose between ASP.NET 4.x and ASP.NET core. Microsoft Learn. URL: <https://learn.microsoft.com/en-us/aspnet/core/fundamentals/choose-aspnet-framework?view=aspnetcore-9.0> (date of access: 05.01.2025).
10. MVC - MDN web docs glossary: definitions of web-related terms | MDN. MDN Web Docs. URL: <https://developer.mozilla.org/en-US/docs/Glossary/MVC> (date of access: 05.01.2025).
11. Overview of ASP.NET core MVC. Microsoft Learn: Build skills that open doors in your career. URL: <https://learn.microsoft.com/en-us/aspnet/core/mvc/overview?view=aspnetcore-9.0> (date of access: 05.01.2025).
12. Microsoft SQL Server | WebStorm. WebStorm Help. URL: <https://www.jetbrains.com/help/webstorm/microsoft-sql-server.html> (date of access: 05.01.2025).
13. Beaulieu A. Learning SQL: generate, manipulate, and retrieve data. O'Reilly Media, Incorporated, 2020.
14. SQL в Access: основні поняття, глосарій і синтаксис - Підтримка від Microsoft. Microsoft Support. URL: <https://support.microsoft.com/uk-ua/topic/sql-в-access-основні-поняття-глосарій-і-синтаксис-444d0303-cde1-424e-9a74-e8dc3e460671> (дата звернення: 05.01.2025).
15. SQL server management studio (SSMS) - SQL server management studio (SSMS). Microsoft Learn: Build skills that open doors in your career. URL: <https://learn.microsoft.com/en-us/sql/ssms/sql-server-management-studio-ssms?view=sql-server-ver16> (date of access: 05.01.2025).

- 16.Первинний ключ (PRIMARY KEY) в SQL - aCode. aCode. URL:
<https://acode.com.ua/primary-key-sql/> (дата звернення: 05.01.2025).
- 17.Зовнішній ключ (FOREIGN KEY) в SQL -. aCode. URL:
<https://acode.com.ua/foreign-key-sql/> (дата звернення: 05.01.2025).
- 18.Entity framework documentation. ef.readthedocs.io. URL:
https://ef.readthedocs.io/_/downloads/en/staging/pdf/ (date of access: 05.01.2025).
- 19.Оптимізація роботи з базами даних за допомогою ORM (Object-Relational Mapping). it-rating.ua. URL: <https://it-rating.ua/optimizatsiya-roboti-z-bazami-danih-za-dopomogoyu-orm-object-relational-mapping> (дата звернення: 05.01.2025).
- 20.Що таке перший підхід коду в Entity Framework Core 7?. Комплексна рекомендація. URL: <https://omutom.liberty.cx.ua/shho-take-pershiy-pidkhid-kodu-v-entity-framework-core-7/> (дата звернення: 05.01.2025).
- 21.Що таке DbContext і DbSet у Entity Framework?. Експерт дня. URL:
<https://hels.aqua-svit.com.ua/shho-take-dbcontext-i-dbset-u-entity-framework/> (дата звернення: 05.01.2025).
- 22.Що таке LINQ?. DevZone. URL: <https://devzone.org.ua/post/linq-ohliad> (дата звернення: 05.01.2025).
- 23.Як створити міграцію в EF core [C#] - dijix - блог. *Dijix - Блог*. URL:
<https://dijix.com.ua/blog/uk/kak-sozdat-migracziyu-v-ef-core-c/> (дата звернення: 05.01.2025).
- 24.Entity framework core migrations. *Learn Entity Framework Core - Getting Started EF Core Tutorial*. URL: <https://www.learnentityframeworkcore.com/migrations> (date of access: 05.01.2025).
- 25.Що таке CRUD простими словами: функції, переваги та приклади. *Highload.today - медіа для розробників*. URL: <https://highload.today/uk/shho-take-crud-prostimi-slovami-funktsiyi-perevagi-ta-prikлади/> (дата звернення: 05.01.2025).
- 26.Операції з даними (CRUD) – Create. *Про Програмування – ABout IT And Programming*. URL: <https://abitap.com/1-5-operacziyi-z-danymy-crud/> (дата звернення: 05.01.2025).
- 27.What Is CRUD? Read. *CrowdStrike: We Stop Breaches with AI-native Cybersecurity*. URL: <https://www.crowdstrike.com/en-us/cybersecurity-101/observability/crud/> (date of access: 05.01.2025).
- 28.Codecademy. What is CRUD? Update. *Codecademy*. URL:
<https://www.codecademy.com/article/what-is-crud> (date of access: 05.01.2025).
- 29.CRUD - delete. *CelerData | High-Performance Data Lake Analytics Powered by StarRocks*. URL: <https://celerdta.com/glossary/create-read-update-and-delete-crud> (date of access: 05.01.2025).
- 30.HTTP методи CRUD - операцій. *JavaRush*. URL:
<https://javarush.com/ua/quests/lectures/ua.javarush.web.core.lecture.level22.lecture05> (дата звернення: 05.01.2025).
- 31.Що таке REST API? Основні принципи REST та GET, POST, PUT, PATCH, DELETE. *tseivo.com*. URL: <https://tseivo.com/b/memecode/t/jgjojmb1ar> (дата звернення: 05.01.2025).
- 32.Documents | AspNet Boilerplate. *ASP.NET Boilerplate*. URL:
<https://aspnetboilerplate.com/Pages/Documents/Introduction> (date of access: 05.01.2025).

33. IApplicationService. *ASP.NET Boilerplate*. URL: <https://aspnetboilerplate.com/Pages/Documents/Application-Services> (date of access: 05.01.2025).
34. Основи: модель (MVC). *Yii Framework*. URL: <https://www.yiiframework.com/doc/guide/1.1/uk/basics.mvc> (дата звернення: 05.01.2025).
35. ASP.NET MVC controller overview (C#). *Microsoft Learn: Build skills that open doors in your career*. URL: <https://learn.microsoft.com/en-us/aspnet/mvc/overview/older-versions-1/controllers-and-routing/aspnet-mvc-controllers-overview-cs> (date of access: 05.01.2025).
36. ASP.NET MVC views overview (C#). *Microsoft Learn: Build skills that open doors in your career*. URL: <https://learn.microsoft.com/en-us/aspnet/mvc/overview/older-versions-1/views/asp-net-mvc-views-overview-cs> (date of access: 05.01.2025).
37. Як використовувати JSON Web Tokens (JWT) для автентифікації. *DevZone*. URL: <https://devzone.org.ua/post/iak-vykorystovuvaty-json-web-tokens-jwt-dlia-avtentyfikatsiyi> (дата звернення: 05.01.2025).
38. Плюси та мінуси JWT: короткий огляд тонкощів цієї технології - highload.today. *Highload.today - media для розробників*. URL: <https://highload.today/uk/plyusy-i-minusy-jwt-kratkij-obzor/> (дата звернення: 05.01.2025).
39. Create data transfer objects (dtos). *Microsoft Learn: Build skills that open doors in your career*. URL: <https://learn.microsoft.com/en-us/aspnet/web-api/overview/data/using-web-api-with-entity-framework/part-5> (date of access: 05.01.2025).
40. Multi-tenancy in ASP.NET core 8 - tenant resolution - michael mckenna. *Home - Michael McKenna*. URL: <https://michael-mckenna.com/multi-tenant-asp-dot-net-8-tenant-resolution/> (date of access: 05.01.2025).

Реалізація контролерів обробки запитів

```

namespace YourProject.Controllers
{
    [ApiController]
    [Route("api/[controller]")]
    public class TransactionController : ControllerBase
    {
        private readonly ITransactionAppService _transactionService;

        public TransactionController(ITransactionAppService transactionService)
        {
            _transactionService = transactionService;
        }

        [HttpGet]
        public async Task<ActionResult<List<TransactionDto>>> GetAll()
        {
            var transactions = await _transactionService.GetAllAsync();
            return Ok(transactions);
        }

        [HttpGet("{id}")]
        public async Task<ActionResult<TransactionDto>> GetById(long id)
        {
            var transaction = await _transactionService.GetByIdAsync(id);
            if (transaction == null)
            {
                return NotFound($"Transaction with ID {id} not found.");
            }
            return Ok(transaction);
        }

        [HttpPost]

```

```

    public async Task<ActionResult> Create([FromBody] CreateTransactionDto
transactionDto)
    {
        await _transactionService.CreateAsync(transactionDto);
        return Ok("Transaction created successfully.");
    }
    [HttpPut("{id}")]
    public async Task<ActionResult> Update(long id, [FromBody]
UpdateTransactionDto transactionDto)
    {
        var result = await _transactionService.UpdateAsync(id, transactionDto);
        if (!result)
        {
            return NotFound($"Transaction with ID {id} not found.");
        }
        return Ok("Transaction updated successfully.");
    }
    [HttpDelete("{id}")]
    public async Task<ActionResult> Delete(long id)
    {
        var result = await _transactionService.DeleteAsync(id);
        if (!result)
        {
            return NotFound($"Transaction with ID {id} not found.");
        }
        return Ok("Transaction deleted successfully.");
    }
}
}

```

Реалізація сервісу обробки даних

```

public class TransactionAppService : ITransactionAppService
{
    private readonly IRepository<TATransaction, long> _transactionRepository;
    private readonly IRepository<Category, long> _categoryRepository;

    public TransactionAppService(
        IRepository<TATransaction, long> transactionRepository,
        IRepository<Category, long> categoryRepository)
    {
        _transactionRepository = transactionRepository;
        _categoryRepository = categoryRepository;
    }

    public async Task<List<TransactionDto>> GetAllAsync()
    {
        var transactions = await _transactionRepository.GetAllIncluding(t =>
t.Category).ToListAsync();

        return transactions.Select(t => new TransactionDto
        {
            Id = t.Id,
            Amount = t.Amount,
            Description = t.Description,
            TransactionTime = t.TransactionTime,
            IsTransferred = t.IsTransferred,
            CategoryName = t.Category?.Name
        }).ToList();
    }
}

```

```

public async Task<TransactionDto> GetByIdAsync(long id)
{
    var transaction = await _transactionRepository.GetAllIncluding(t => t.Category)
        .FirstOrDefaultAsync(t => t.Id == id);

    if (transaction == null) return null;

    return new TransactionDto
    {
        Id = transaction.Id,
        Amount = transaction.Amount,
        Description = transaction.Description,
        TransactionTime = transaction.TransactionTime,
        IsTransferred = transaction.IsTransferred,
        CategoryName = transaction.Category?.Name
    };
}

public async Task CreateAsync(CreateTransactionDto transactionDto)
{
    var transaction = new TATransaction
    {
        Amount = transactionDto.Amount,
        Balance = transactionDto.Amount,
        CategoryId = transactionDto.CategoryId,
        Description = transactionDto.Description,
        TransactionTime = transactionDto.TransactionTime,
        IsTransferred = transactionDto.IsTransferred
    };
}

```

```

        await _transactionRepository.InsertAsync(transaction);
    }

    public async Task<bool> UpdateAsync(long id, UpdateTransactionDto
transactionDto)
    {
        var transaction = await _transactionRepository.FirstOrDefaultAsync(id);
        if (transaction == null) return false;

        transaction.Amount = transactionDto.Amount;
        transaction.Description = transactionDto.Description;
        transaction.TransactionTime = transactionDto.TransactionTime;
        transaction.IsTransferred = transactionDto.IsTransferred;
        transaction.CategoryId = transactionDto.CategoryId;

        await _transactionRepository.UpdateAsync(transaction);
        return true;
    }

    public async Task<bool> DeleteAsync(long id)
    {
        var transaction = await _transactionRepository.FirstOrDefaultAsync(id);
        if (transaction == null) return false;

        await _transactionRepository.DeleteAsync(transaction);
        return true;
    }
}

```