Міністерство освіти і науки України
Харківський національний університет радіоелектроніки

Факультет Інформаційно-аналітичних технологій та менеджменту
(повна назва)
Кафедра Інформатики
(повна назва)

КВАЛІФІКАЦІЙНА РОБОТА
Пояснювальна записка

рівень вищої освіти перший (бакалаврський)

РОЗРОБКА ЗАСТОСУНКУ ДЛЯ УПРАВЛІННЯ ПЕРСОНАЛЬНИМИ ФІНАНСАМИ
(тема)

Виконав:
здобувач 4 року навчання,
групи ІТІНФ-21-2
Краснянська В. В.
(прізвище, ініціали)

Спеціальність 122 Комп'ютерні науки
(код і повна назва спеціальності)

Тип програми освітньо-професійна

Освітня програма Інформатика
(повна назва освітньої програми)

Керівник доц. Тітова О. В.
(посада, прізвище, ініціали)

Допускається до захисту

Завідувач кафедри інформатики _____ Кобилін О. А.
(підпис) (прізвище, ініціали)

2025 р.

Харківський національний університет радіоелектроніки

Факультет Інформаційно-аналітичних технологій та менеджменту

Кафедра Інформатики

Рівень вищої освіти перший (бакалаврський)

Спеціальність 122 Комп'ютерні науки
(код і повна назва)

Тип програми освітньо-професійна

Освітня програма Інформатика
(повна назва)

ЗАТВЕРДЖУЮ:
Зав. кафедри _____
(підпис)
«____»_____ 2025 р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ

здобувачеві Краснянській Валерії Валеріївні
(прізвище, ім'я, по батькові)

1. Тема роботи Розробка застосунку для управління персональними фінансами

затверджена наказом університету від 19 травня 2025 року № 381Ст
2. Термін подання здобувачем роботи до екзаменаційної комісії 25 травня 2025 р.
3. Вихідні дані до роботи науково-методична та науково-технічна література, матеріали з інтернет джерел, існуючи вебзастосунки для керування персональними фінансами.

4. Перелік питань, що потрібно опрацювати в роботі
1. Аналіз існуючих рішень та технологій для створення вебзастосунків з управління персональними фінансами.
2. Розробка архітектури вебзастосунку: моделі даних, API та бази даних.
3. Програмна реалізація онлайн-застосунку.

5. Перелік графічного матеріалу із зазначенням креслеників, схем, плакатів, комп'ютерних ілюстрацій (п.5 включається до завдання за рішенням випускової кафедри) Актуальність проблеми, постановка задачі, тестові зображення.

6. Консультанти розділів роботи (п.6 включається до завдання за наявності консультантів згідно з наказом, зазначеним у п.1)

Найменування розділу	Консультант (посада, прізвище, ім'я, по батькові)	Позначка консультанта про виконання розділу	
		підпис	дата

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів роботи	Строк / терміни виконання етапів роботи	Примітка
1	Отримання завдання на кваліфікаційну роботу	07.04.2025	
2	Аналіз завдання, підбір літератури	08.04.25-10.04.25	
3	Аналіз літератури з досліджуваної проблеми	11.04.25-14.04.25	
4	Аналіз технічних засобів	15.04.25-20.04.25	
5	Розробка методу	21.04.25-27.04.25	
6	Програмна реалізація	28.04.25-11.05.25	
7	Оформлення пояснювальної записки	12.05.25-20.05.25	
8	Перевірка на нормоконтроль	21.05.25-01.06.25	
9	Перевірка на плагіат	21.05.25-01.06.25	
10	Рецензування	21.05.25-01.06.25	
11	Підготовка презентації та доповіді	21.05.25-18.06.25	
12	Занесення роботи в електронний архів	02.06.25-18.06.25	
	Попередній захист кваліфікаційної роботи	02.06.25-18.06.25	

Дата видачі завдання 7 квітня 2025 р.

Здобувач _____
(підпис)

Керівник роботи _____ _____доц. Тітова О. В._____
(підпис) (посада, прізвище, ініціали)

РЕФЕРАТ/ABSTRACT

Пояснювальна записка до кваліфікаційної роботи: 70 с., 5 табл., 46 рис., 32 джерела.

ОБЛІК ДОХОДІВ ТА ВИТРАТ, REACT, C#, .NET, ENTITY FRAMEWORK, МОДЕЛЬ, КОНТРОЛЛЕР, БАЗИ ДАНИХ, SQL, СТАТИСТИКА ВИТРАТ, ШТУЧНИЙ ІНТЕЛЕКТ.

Об’єктом роботи є збереження та відображення фінансових даних, що дозволяють користувачам вести облік доходів та витрат за категоріями.

Метою роботи є розробка застосунку для керування персональних фінансів, що забезпечує зручність контролю персонального бюджету та відображення статистики витрат за категоріями .

У ході роботи здійснюється аналітичний огляд існуючих фінансових застосунків, вивчаються ключові характеристики архітектури онлайн-сервісів, а також проводиться аналіз засобів для створення застосунків, зокрема React, Bootstrap, Entity Framework, .NET C# та інтеграція Gemini API для надання персоналізованих фінансових порад.

У результаті роботи здійснена програмна реалізація вебзастосунку, який надає користувачам можливість вести облік власного бюджету, отримувати наочну статистику витрат чи доходів та ефективніше організовувати фінансові дані.

INCOME AND OUTCOME ACCOUNTING, REACT, C#, .NET, ENTITY FRAMEWORK, MODEL, CONTROLLER, DATABASE, SQL, EXPENSE STATISTICS, ARTIFICIAL INTELEGENCE.

The object of the work is the storage and visualization of financial data, enabling users to track income and expenses by category.

The purpose of the work is to develop an application for managing personal finances that ensures convenience in budget control and provides statistics on expenses by category.

During the study, an analytical review of existing financial applications was conducted, the key architectural features of online services and tools for application development were examined, in particular React, Bootstrap, Entity Framework, .NET C# and the integration of Gemini API for providing personalized financial advice.

As a result of this work, a web application was implemented that allows users to track their personal budget, view visual statistics of expenses or income and organize financial data efficiently.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ, СКОРОЧЕНЬ І ТЕРМІНІВ

API – Application Programming Interface (прикладний програмний інтерфейс)

IDE – Integrated Development Environment (інтегроване середовище розробки)

SDK – Software Development Kit (набір із засобів розробки)

Framework (фреймворк) – це спеціалізована бібліотека для полегшення розробки складних систем

Entity – об’єкт, що представляє таблицю в реляційній базі даних

База даних – колекція збереженної та впорядкованої інформації

Entity Framework – об’єктно-реляційне відображення для роботи з базою даних

Full-stack application – вебзастосунок, який охоплює як фронтенд, так і бекенд

Frontend (фронтенд) – частина застосунку, що відповідає за інтерфейс користувача

Backend (бекенд) – частина застосунку, що обробляє логіку, працює з базою даних та керує запитами від користувача

DOM (Document Object Model) – об’єктна модель документа, яку браузер створює на основі HTML-коду для представлення структури вебсторінки у вигляді дерева елементів

Кросплатформеність – здатність програми працювати на різних операційних системах (Windows, macOS, Linux тощо) без необхідності змінювати її код

ШІ – штучний інтелект

ВСТУП

Практично будь-яка сфера діяльності в сучасному світі тісно пов’язана з фінансовими операціями, оскільки гроші є універсальним еквівалентом обміну, виміру та накопичення вартості. У бізнесі це виявляється у розрахунках за товари та послуги, інвестиції та управління витратами. В освіті та науці фінансові операції забезпечують фінансування досліджень та програм. Навіть у повсякденному житті люди постійно стикаються з необхідністю керувати особистим бюджетом, оплачувати рахунки чи робити заощадження.

Жодна людина не може обійтись без мінімальних витрат на продукти, предмети побуту, одяг чи інші повсякденні потреби, що наголошує на важливості фінансової грамотності в наші дні. Сучасні технології, такі як онлайн-банкінг та електронні платежі, спрощують ці процеси, але одночасно вимагають від кожної людини мінімальних знань у галузі фінансів.

Враховуючи це, виникає актуальна задача розробки застосунку, який міг би надавати користувачам зручний та безпечний спосіб вести облік власного бюджету. Застосунок повинен дозволяти користувачу отримати візуалізацію витрат та доходів у вигляді діаграм та надавати поради щодо покращення фінансового стану. Це допоможе користувачам побачити, куди витрачаються гроші, визначити основні статті витрат і за необхідності внести зміни для досягнення фінансових цілей.

Ця робота спрямована на розробку та впровадження застосунку для керування персональними фінансами. Цей застосунок буде в нагоді людям, що тільки розпочинають знайомство з фінансовою грамотністю та прагнуть ефективно організувати власний бюджет.

1 ОГЛЯД ОСНОВНИХ МЕТОДІВ СТВОРЕННЯ ВЕБЗАСТОСУНКУ ДЛЯ КЕРУВАННЯ ПЕРСОНАЛЬНИМИ ФІНАНСАМИ

1.1 Розвиток інтернету та WEB

Інтернет став основою для подальшого розвитку та існування вебзастосунків. Його витоки сходять до часів Другої світової війни, а саме роботи американського вченого, що працював над виявленням підводних човнів для США. Він зміг об’єднати зусилля провідних університетів, таких як Берклі та Гарвард, а також військових сил задля підвищення фінансування наукових досліджень, що були пов’язані з потребами збройних сил Сполучених Штатів. Це співробітництво призвело до створення ARPANET – першого прототипу сучасного інтернету, який служив для обміну конфіденційної військової інформації. Проте він став доступним для масового користування лише на початку 1980-х років, коли почали розвиватися протоколи для глобальної мережі, такі як TCP/IP. Протягом 1990-х років потужна комерціалізація інтернету, завдяки появі веббраузерів, таких як Mosaic та Netscape, дозволила значно розширити доступ до інтернету, а його інтеграція у повсякденне життя почала змінювати практично всі сфери діяльності [1].

За свою коротку історію інтернет пережив чимало істотних змін, що сприяли його еволюції в потужну платформу для вебзастосунків. Спочатку інтернет був простим інструментом для обміну текстовою інформацією. Перші вебсайти були статичними, представляючи лише візитки з текстовими сторінками і зображеннями (рис. 1.1) [2].

Однак, завдяки розвитку нових мов програмування, таких як JavaScript, та створенню фреймворків, таких як Angular та React, стало можливим створювати інтерактивні сайти, які реагують на дії користувача в реальному часі. Це дало змогу створювати повноцінні вебсайти та вебзастосунки, які виконують функції, раніше доступні тільки через настільні програми.

Рисунок 1.1 – Перший вебсайт

Сучасні вебзастосунки стають дедалі адаптивнішими та функціональнішими завдяки використанню передових технологій. Анімації на основі CSS дозволяють створювати плавні переходи й ефекти, що покращують візуальний досвід користувачів. А гармонійні кольорові палітри, розроблені дизайнерами, роблять інтерфейси ще привабливішими (рис. 1.2) [3].

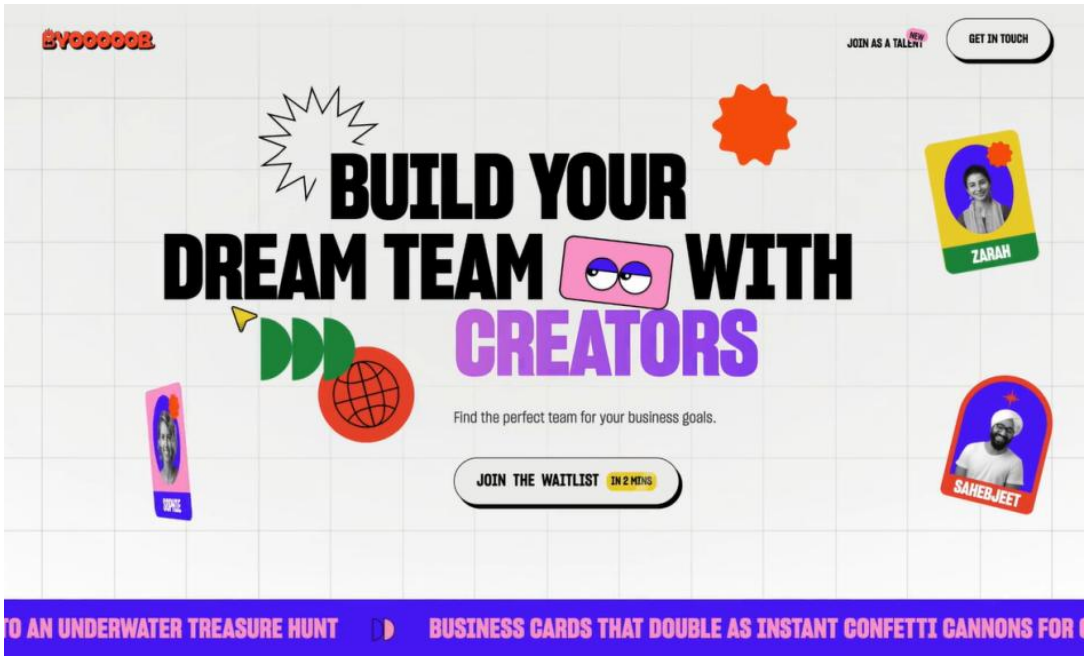

Рисунок 1.2 – Сучасний вебсайт

Використання фреймворків JavaScript, таких як React, Angular і Vue, дає змогу розробникам створювати швидкі, динамічні інтерфейси, що адаптуються до пристроїв різного розміру, а також надають можливість

інтеграції з API, базами даних та іншими зовнішніми сервісами. Зокрема, за допомогою таких інструментів як Node.js, серверна частина вебзастосунків також стала більш потужною і гнучкою, що дозволяє створювати масштабовані рішення для будь-яких завдань.

За менш ніж 20 років кількість користувачів з тисяч достигла мільярдів. А наразі інтернетом користується приблизно 4 мільярда людей, що становить майже 50 відсотків населення Землі. Це свідчить про стрімке поширення інтернету та його інтеграцію в наше повсякденне життя.

Один із ключових впливів вебзастосунків – це революція у сфері комунікацій. Месенджери, соціальні мережі та платформи для відеодзвінків дозволяють людям залишатися на зв’язку незалежно від відстані (Telegram, Instagram, WhatsApp).

У сфері фінансів вебзастосунки сприяли зростанню популярності онлайн-банкінгу, мобільних гаманців та бюджетних менеджерів. Люди можуть оплачувати рахунки, здійснювати грошові перекази, інвестувати та стежити за своїми витратами, використовуючи лише смартфон (Privat24, Monobank).

Освіта також зазнала значного впливу завдяки вебзастосункам. Онлайн-курси, інтерактивні платформи для навчання, електронні бібліотеки, сайти викладачів та доступ до величезної кількості ресурсів, таких як навчально-методичні матеріали дозволяють людям з усього світу отримувати знання у зручний для них час [4].

Не можна оминути й роль вебзастосунків у сфері здоров’я. Мобільні додатки допомагають людям стежити за фізичною активністю, раціоном харчування, режимом сну, а також записуватися на прийом до лікаря.

У 21 столітті вебзастосунки стали не просто інструментами, а фундаментальною частиною нашого повсякденного існування. Вони допомагають нам бути продуктивнішими, здоровішими, освіченішими та щасливішими, якщо ми використовуємо їх усвідомлено та відповідально. Водночас активне використання вебзастосунків має і свої виклики. Серед

них – залежність від технологій, ризики втрати конфіденційності даних та надмірна цифровізація, що може знижувати живе спілкування. Однак правильне і розумне використання вебзастосунків здатне суттєво поліпшити якість життя, забезпечуючи зручність, ефективність і нові можливості в різних сферах.

1.2 Дослідження актуальності розробки вебзастосунку

Вебзастосунок – це програмне забезпечення, що працює та відкривається у вікні браузера, забезпечуючи доступ до різноманітних функцій без необхідності встановлення додаткового програмного забезпечення на пристрій користувача. Це стало можливим завдяки тому, що браузери є кросплатформенними, що дозволяє запускати вебзастосунок на різних операційних системах, таких як Windows, macOS, Linux, Android або iOS. Користувачеві достатньо мати стабільний доступ до інтернету, що робить вебзастосунок універсальним і зручним інструментом для багатьох задач, починаючи від комунікації і до роботи з даними.

Однією з головних переваг вебзастосунків є їхня доступність. Користувач може отримати необхідний функціонал у будь-який час і з будь-якої точки світу, маючи підключення до інтернету. Вебзастосунки дозволяють автоматизувати процеси, що значно економить час і ресурси як для користувачів, так і для компаній. Крім того, сучасні вебтехнології забезпечують глибоку інтеграцію з іншими системами, такими як штучний інтелект, аналітика даних і хмарні сервіси.

Перші вебзастосунки почали з’являтися у 1990-х роках із розвитком Всесвітньої павутини. Спочатку це були прості динамічні сайти, що дозволяли взаємодію з користувачами, наприклад, через форми або електронні таблиці. У 2000-х роках поширилися вебзастосунки для електронної комерції, банкінгу та соціальних мереж, що змінили спосіб комунікації та ведення бізнесу. З

розвитком технологій, таких як AJAX та HTML5, вебзастосунки стали більш інтерактивними, наближаючись за можливостями до традиційного програмного забезпечення.

Сьогодні, у 2025 році, вебзастосунки є основою цифрового світу. Завдяки хмарним обчисленням, прогресу в безпекових технологіях та появі прогресивних вебзастосунків (PWA), вони стали ще швидшими, зручнішими та доступнішими для користувачів на будь-яких пристроях. Інновації в області штучного інтелекту та автоматизації дозволяють створювати ще розумніші рішення, що адаптуються до потреб користувачів та підвищують ефективність бізнес-процесів [5, 6].

Фінансовий вебзастосунок матиме суттєвий позитивний вплив на суспільство. Він сприятиме підвищенню фінансової грамотності, допомагаючи користувачам краще розуміти свої доходи та витрати. Автоматизовані звіти та аналітика допоможуть уникнути зайвих витрат і ефективно розподіляти кошти. Крім того, використання штучного інтелекту дозволить запропонувати індивідуальні рекомендації з управління бюджетом, що особливо корисно для молоді та осіб, які бажають покращити свою фінансову дисципліну.

Такі сервіси також можуть бути інтегровані з банківськими системами, що дозволить користувачам отримувати єдину платформу для керування фінансами, оплати рахунків та ведення бізнес-аналітики. Це сприятиме економічному зростанню та забезпеченню фінансової стабільності для широкого кола користувачів.

Таким чином підбиваючи підсумки сказаного вище, можна дійти висновку, що розробка подібного застосунку є виправданою та перспективною ініціативою, яка сприятиме економічному розвитку та фінансовій стабільності суспільства.

1.3 Аналіз застосунків для ведення обліку персональних фінансів

Під час планування власного проєкту, також необхідно зробити аналіз застосунків конкурентів. В першу чергу, оцінюючи досвід користувачів інших застосунків, можна зрозуміти, що працює добре, а що не зовсім, що дозволить створити більш зручний і адаптований до потреб користувачів інтерфейс і функціонал.

Розглянемо переваги та недоліки популярних застосунків для керування персональним бюджетом, зокрема YNAB, GoodBudget і PocketGuard. На основі цього аналізу визначимо ключові вимоги та сформулюємо постановку задачі.

1.3.1 YNAB (You Need a Budget)

YNAB – це вебзастосунок для керування персональним бюджетом, заснований на системі конвертів (рис. 1.3), розробленій однойменною приватною американською компанією. Головна ідея додатку – «Give every dollar a job» («Кожному долару – своє завдання») [7].

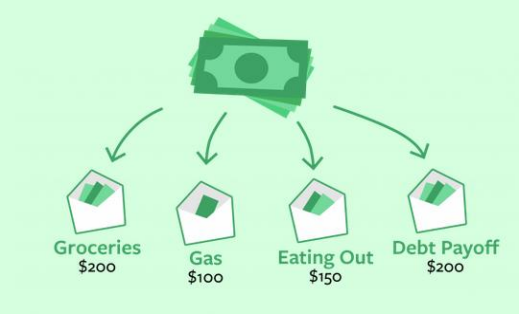

Рисунок 1.3 – Система конвертів

Розробка цього застосунку почалася у 2004 році, коли Джессі Мечам, під час навчання в університеті, зіткнувся з фінансовими труднощами. Разом зі своєю дружиною вони зрозуміли, що їм потрібно покращити управління своїм

бюджетом. Спочатку застосунок був простим електронним табличним інструментом для відстеження витрат, але з часом він еволюціонував у більш функціональний продукт. Сьогодні YNAB існує в двох версіях: вебзастосунок та мобільний застосунок. Цікавий факт, що розробники навіть зараз продовжують пряму комунікацію з користувачами, регулярно публікуючи відео туторіали для застосунку, надають людям поради для покращення свого фінансового стану та навіть мають особистий блог.

Відмінними рисами цього додатку є гнучкий підхід до введення даних: користувачі можуть імпортувати транзакції з фінансових установ (наприклад, завантажити транзакції форматах OFX, QFX, QIF, CSV з банківських додатків), вводити їх вручну або прив’язати YNAB до свого банківського рахунку та автоматизувати імпорт транзакцій. Крім того, застосунок пропонує регулярні фінансові звіти, які допомагають залишатися в курсі стану своїх фінансів.

До перерахованих зверху переваг, слід, також додати прогнозування майбутніх витрат, створення планів погашення боргів, налаштування цілей, а найголовніше – надійний захист даних, що гарантує конфіденційність і безпеку фінансової інформації.

Щодо інтерфейсу, то було вирішено зробити аналіз головної сторінки, використовуючи існуючий туторіал як основу (рис. 1.4).

З першого погляду може здатись, що інтерфейс має дуже багато всього, тому розглянемо основні частини окремо:

– у лівій частині екрану відображаються всі рахунки користувача: доступні кошти для витрат або, у випадку кредитної картки, сума заборгованості перед банком;

– категорії, тобто репрезентація витрат та збережень;

– гроші яким треба призначити ціль;

– підсумування за поточний та попередні місяці.

Хоча інтерфейс виглядає чистим і структурованим, він може здатися перевантаженим новачкам через велику кількість налаштувань та функцій.

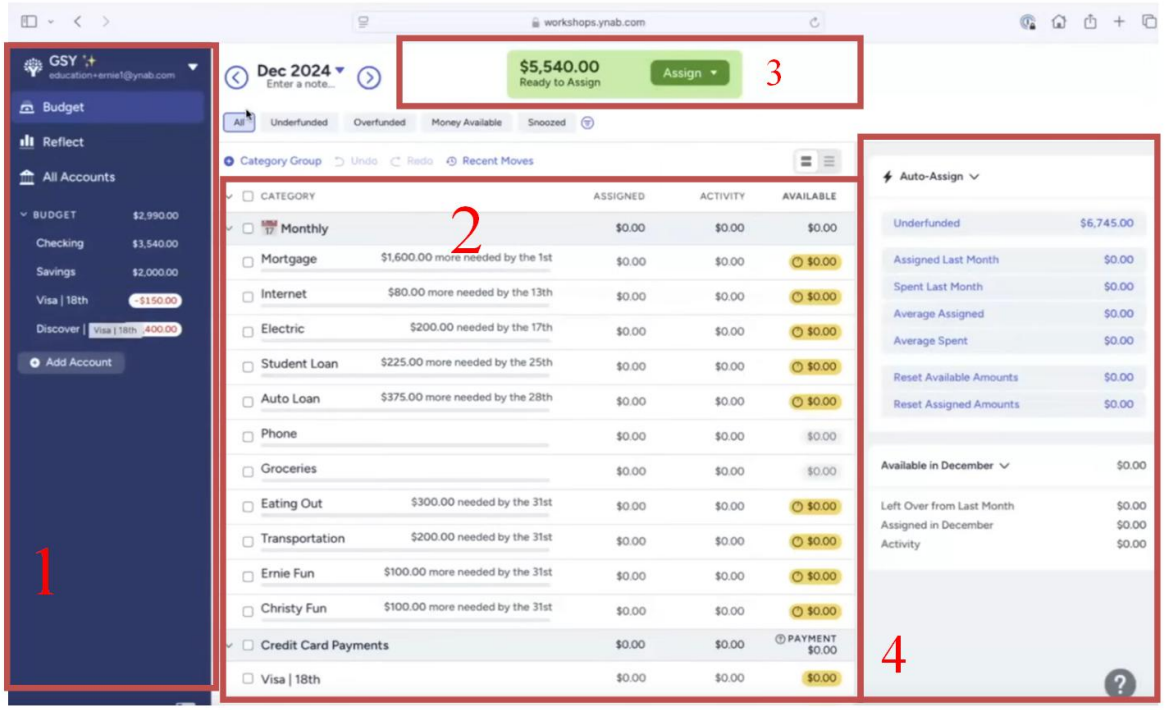

Рисунок 1.4 – Інтерфейс вебзастосунку YNAB

Попри всі переваги, YNAB має кілька недоліків, які можуть вплинути на досвід користувачів. Великий потенціал та не менш велика купа інструментів може налякати користувача, що тільки починає знайомство з фінансовою дисципліною. Не меншою перешкодою для деяких користувачів може бути і ціна платної підписки, що становить $14.99 на місяць ($98.99 на рік), також відсутність локалізації гальмує подальше розповсюдження застосунку, оскільки користувачі, які не володіють англійською мовою можуть зіткнутись з труднощами у його використанні.

YNAB має недоліки, які можуть бути критичними для певних користувачів, особливо через ціну або складність у використанні. Проте для тих, хто готовий адаптуватися до його системи, цей застосунок є потужним інструментом для управління персональним бюджетом.

1.3.2 GoodBudget

GoodBudget – це безкоштовний застосунок для ведення бюджету, що дозволяє користувачам вручну записувати витрати та доходи, створювати конверти для категорій витрат і тримати фінанси під контролем [8].

GoodBudget розпочався в 2009 році як експеримент Dayspring Technologies, фірми з розробки інтернет та мобільних пристроїв у Сан-Франциско. Зараз же користувачі мають можливість користуватись застосунком для керування фінансів на своїх мобільних пристроях та у веббраузері. Головні ідеї додатку – «There is enough» («Є достатньо») та «Giving is good» («Дарувати – це добре»).

З переваг необхідно зазначити, що маючи безкоштовну версію користувач отримує можливість створення 20 конвертів (преміум – безліміт), 1 рахунок (преміум – безліміт), підключення 2 девайсів (преміум – 5 девайсів) та збереження історії на рік (преміум – 7 років).

Є можливість завантаження нещодавньої банківської активності у форматах QFX, OFX, CSV. Також система пропонує звіти за різними критеріями, зокрема витрати за конвертами, прогрес у погашенні боргів та розподіл бюджету, що дозволяє детально аналізувати фінансовий стан і приймати обґрунтовані рішення.

У розділі навчання було знайдено додаткову інформацію щодо поліпшення фінансового стану, зокрема статті та поради про ефективне управління бюджетом, а також подкасти на актуальні теми, такі як стратегія заощаджень, планування великих покупок, зокрема придбання будинку, або рекомендації щодо ведення спільного бюджету в парі, що може бути корисним для сімейних фінансів.

Перейдемо до розгляду інтерфейсу застосунку, наступним чином виглядає стартова сторінка Goodbudget користувача, який тільки почав налаштування особистого бюджету (рис. 1.5):

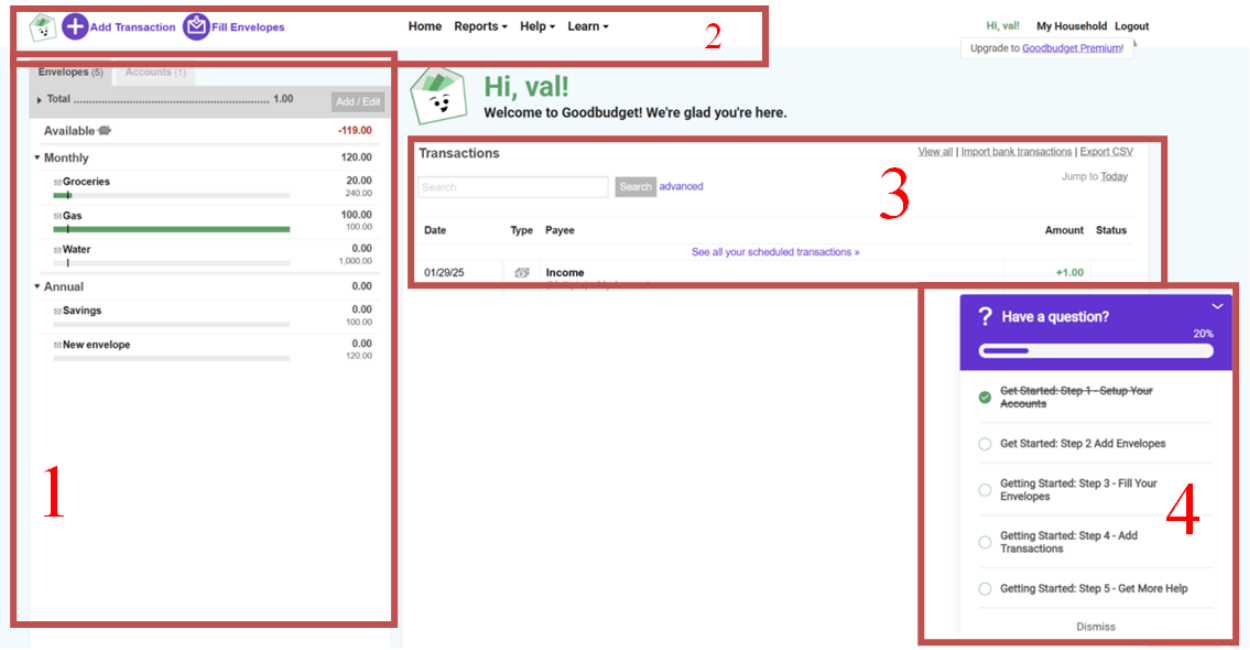

Рисунок 1.5 – Інтерфейс вебзастосунку GoodBudget

– ліва панель містить Envelopes (Конверти), список категорій бюджету, та список рахунків;

– верхня панель навігації має кнопки «Add Transaction» та «Fill Envelopes» (додати транзакцію та поповнити конверти), розділи Home, Reports, Help, Learn, посилання на оновлення до Goodbudget Premium;

– центральна частина відображає список транзакції, опції «View all» (Переглянути все), »Import bank transactions» (Імпорт банківських транзакцій), «Export CSV» (Експорт у CSV);

– права панель – це прогрес налаштування, що складається з 5 кроків налаштування: Setup Your Accounts (налаштуйте свої облікові записи,), Add Envelopes (додайте конверти), Fill Your Envelopes (заповніть свої конверти), Add Transactions (додайте транзакції).

Загалом інтерфейс є інтуїтивно зрозумілим, і навіть під час першого візиту не викликає інформаційного перевантаження, оскільки всі основні функції структуровані логічно, а підказки допомагають швидко зорієнтуватися в можливостях застосунку.

Незважаючи на простоту та зручність у використанні слід відзначити, що GoodBudget має свої недоліки. Попри сучасним тенденціям UI (User

interface) дизайн мобільного застосунку та вебверсії виглядає дещо застарілим, що може знеохочувати нових користувачів. Відсутність належної локалізації та мультивалютності може ускладнити користування додатком, оскільки не маючи знань англійської мови або використовуючи кілька валют одночасно користувачі можуть зіткнутись з труднощами у використанні застосунку. Застосунок не підтримує прогнозування бюджету на основі минулих даних та автоматичної синхронізації з банківськими рахунками.

Отже, GoodBudget найкраще підходить для користувачів, які хочуть вручну контролювати свої витрати, використовувати систему «конвертів» та планувати сімейний бюджет. Це чудовий вибір для тих, хто надає перевагу простоті та організованості без надмірної автоматизації.

1.3.3 PocketGuard

PocketGuard – це потужний застосунок для контролю особистих фінансів, доступний у вебверсії, а також у вигляді мобільного застосунку для iOS та Android. Він допомагає користувачам спростити управління бюджетом, відстежувати витрати та ухвалювати обґрунтовані фінансові рішення [9].

Застосунок був створений 1 січня 2014 року, та з того часу головною метою розробників є спрощення фінансового життя людей, допомагаючи їм контролювати свій бюджет та приймати правильні рішення для здійснення своїх мрій. За допомогою великої кількості інструментів PocketGuard автоматизує більшість процесів, зменшуючи час і зусилля, необхідні для керування фінансами.

Однією з найкорисніших функцій PocketGuard є «In My Pocket», що дозволяє визначати, скільки коштів залишається у розпорядженні користувача після врахування обов’язкових платежів, заборгованостей, заощаджень та інших фінансових зобов’язань.

PocketGuard підтримує синхронізацію з різними банківськими рахунками та кредитними картками, що дозволяє автоматично імпортувати транзакції. Застосунок автоматично розподіляє витрати за категоріями, такими як продукти, комунальні платежі, розваги, транспорт тощо.

Користувачі також можуть створювати власні категорії для детальнішого аналізу витрат відповідно до своїх індивідуальних потреб. Експорт даних може відбуватись тільки у CSV форматі, що подалі надає можливість користувачу створювати діаграми та аналізувати дані за допомогою інструментів MS Excel або Google Sheets.

Маючи преміум-підписку, користувачі отримують доступ до розширених функцій, таких як «Секторна діаграма витрат» для графічного аналізу витрат, «Цілі» для планування заощаджень, підтримка декількох грошових рахунків, функції касових операцій, а також можливість завантаження зображень чеків і керування дублікатами касових операцій.

Далі буде розглянуто інтерфейс застосунку PocketGuard (рис. 1.6):

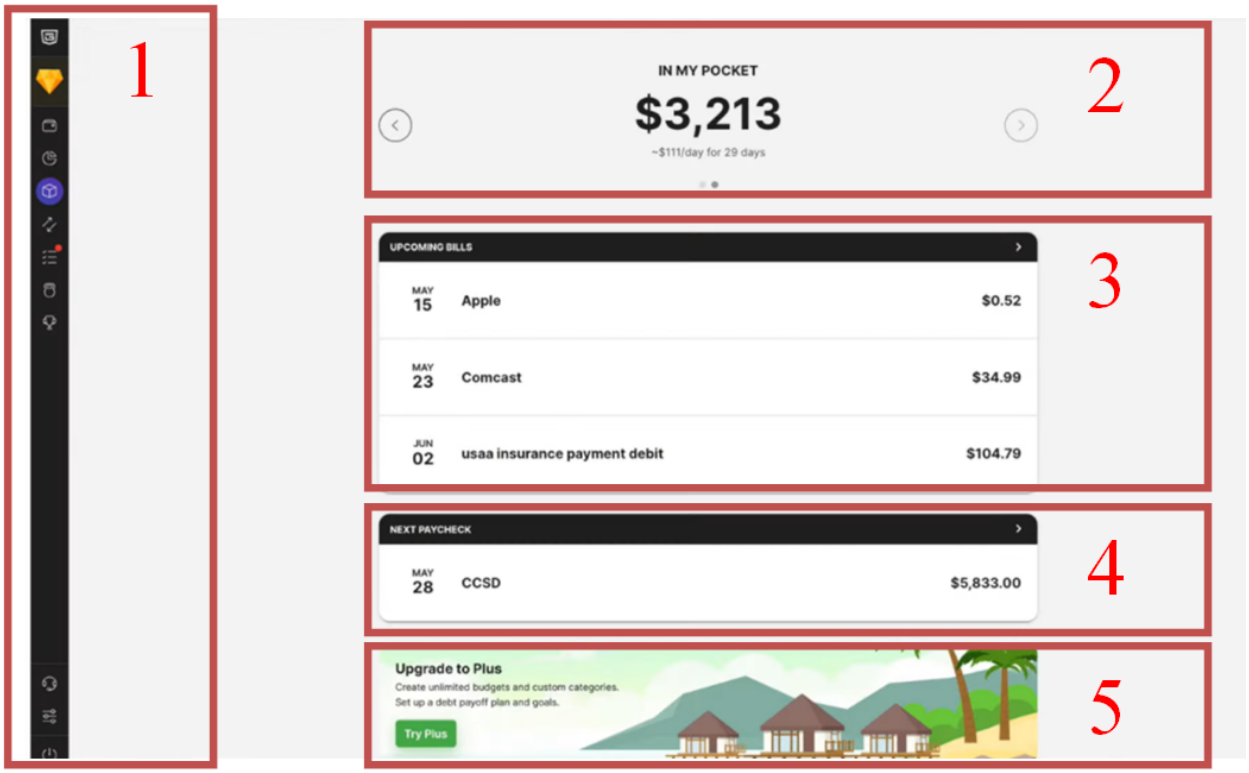

Рисунок 1.6 – Інтерфейс застосунку PocketGuard

– зліва можна знайти меню для навігації до головної сторінки, аналітиці, бюджету, налаштувань тощо;

– у верхній частині сторінки знаходиться розділ «In my pocket», що показує доступні кошти користувача після врахування всіх запланованих витрат та фінансових зобов’язань;

– секція «Upcoming Bills» (Майбутні рахунки) має перелік майбутніх витрат;

– секція «Next Paycheck» (Наступна зарплата) показує очікуваний дохід;

– внизу є банер «Upgrade to Plus», який пропонує користувачам оновитися до PocketGuard Plus.

Загалом, інтерфейс орієнтований на простоту та ефективність, він виглядає чистим та інтуїтивно зрозумілим, що робить PocketGuard зручним для користувачів, які хочуть швидко контролювати свої фінанси без зайвої складності.

Попри широкий функціонал, у PocketGuard є й певні обмеження. Наприклад, з грудня 2024 року компанія прибрала можливість безкоштовного користування сервісом та отримання довічної підписки, що робить застосунок менш доступним для тих, хто не готовий оформлювати преміум-підписку. Хоча програма містить детальний опис функцій і посібник з усунення більшості несправностей, вона не пропонує подкасти чи статті з фінансовими порадами. Додатковими недоліками є відсутність локалізації та підтримки мультивалютності, що обмежує зручність використання для користувачів з різних країн.

Незважаючи на деякі обмеження, PocketGuard залишається ефективним інструментом для тих, хто прагне покращити своє фінансове планування та організацію бюджету.

На основі аналізу фінансових застосунків можна зробити висновок, що кожен із них має свої сильні та слабкі сторони, залежно від потреб користувача. Порівняльну таблицю, де оцінено ключові характеристики кожного застосунку показано у таблиці 1.1.

Таблиця 1.1 – Результати порівнянь оглянутих застосунків

Параметр	**YNAB**	**GoodBudget**	**PocketGuard**
1	**2**	**3**	**4**
Модель роботи	Преміум- підписка ($14.99/міс або $98.99/рік)	Безкоштовна версія, преміум-підписка. ($10.00/міс або $80.00/рік)	Преміум-підписка, $6.25/міс ($74.99/рік) при щорічній оплаті або $12.99/міс ($155.88/рік) при щомісячній оплаті
Доступність	Вебверсія, iOS, Android	Вебвверсія, iOS, Android	Вебвверсія, iOS, Android
Основний функціонал	Автоматичне або ручне введення витрат, система конвертів, категоризація витрат, прогнозування бюджету, інтеграція з банками, фінансові звіти	Ручне введення витрат, система конвертів, базові звіти, імпорт транзакцій вручну	Автоматичний імпорт транзакцій, аналіз витрат за категоріями, бюджетування, функція «In My Pocket» для визначення доступних коштів після всіх зобов’язань
Автоматизація	Висока: підтримка автоматичного імпорту з банківських рахунків	Мінімальна: всі транзакції вводяться вручну	Висока: підключення до банківських рахунків та кредитних карток, розподіл витрат за категоріями

Продовження таблиці 1.1

1	2	3	4
Система конвертів	Основна концепція застосунку	Основна концепція застосунку	Відсутня, замість цього використовується система бюджетування
Фінансові звіти	Детальні аналітичні звіти, прогнозування витрат, секторні діаграми витрат	Базові звіти за категоріями витрат та доходів, діаграми витрат	Доступні аналітичні інструменти, секторні діаграми витрат (у преміум-версії), експорт даних у CSV
Інтерфейс	Функціональний, але може здатися складним для новачків	Простий та інтуїтивний, але застарілий дизайн	Мінімалістичний, простий та зручний
Локалізація	Відсутня (англійська мова)	Відсутня (англійська мова)	Відсутня (англійська мова)
Додаткові можливості	Прогнозування бюджету, навчальні матеріали, блог, підтримка планів погашення боргів	Навчальні матеріали, подкасти, просте управління сімейним бюджетом	«In My Pocket», цілі заощаджень, підтримка кількох рахунків, касові операції, завантаження чеків (у преміум-версії)

Продовження таблиці 1.1

1	2	3	4
Основні недоліки	Відсутність безкоштовної версії, відсутність локалізації та мультивалютності, складний для початківців.	Відсутність автоматизації, обмежений функціонал, застарілий дизайн.	Відсутність безкоштовної версії, немає подкастів або статей з фінансовими порадам локалізації.
Кому підходить?	Для тих, хто серйозно ставиться до фінансового планування, готовий приділити час налаштуванню бюджету.	Для користувачів, які хочуть вручну контролювати витрати без складних інструментів.	Для тих, хто хоче автоматизувати управління бюджетом і швидко аналізувати свої фінанси.
Синхронізація з банками	Підтримується	Відсутня	Підтримується
Підтримка мультивалютності	Відсутня	Відсутня	Відсутня

Аналізуючи наведені в таблиці дані, можна зробити висновок, що основні переваги сучасних рішень для керування бюджетом включають автоматизацію обліку витрат, категоризацію транзакцій, візуалізацію фінансових даних та можливість встановлення фінансових цілей. Крім того, деякі застосунки пропонують інтеграцію з банківськими рахунками, що значно спрощує моніторинг доходів і витрат.

Серед недоліків найпоширенішими є складність освоєння для нових користувачів, обмежені можливості безкоштовних версій, недостатня гнучкість у налаштуваннях та питання конфіденційності даних. Деякі застосунки змушують користувачів вручну вносити частину операцій, що може стати незручним у довгостроковій перспективі.

Таким чином, ідеальний вебзастосунок для керування фінансами має поєднувати інтуїтивно зрозумілий інтерфейс, гнучку систему налаштувань та надійний рівень безпеки. На основі цього аналізу можна сформулювати вимоги до нового рішення, яке враховуватиме недоліки існуючих застосунків і забезпечить більш зручний та ефективний спосіб управління особистими фінансами.

1.4 Постановка задачі

Таким чином, розробка застосунку для керування персональними фінансами є актуальною темою. Вебзастосунок стане корисним інструментом для тих, хто прагне контролювати свій бюджет і досягати фінансових цілей.

Об’єктом роботи є збереження та відображення фінансових даних, що дозволяють користувачам вести облік доходів та витрат за категоріями.

Метою роботи є розробка застосунку для керування персональних фінансів, що забезпечує зручність контролю персонального бюджету та відображення статистики витрат за категоріями

Для досягнення мети необхідно вирішити такі завдання:

− провести аналіз існуючих застосунків для керування персональними фінансами;

− розробити програмну архітектуру з необхідними API для зберігання та відображення транзакцій та налаштувати API для надсилання запитів до ШІ;

− реалізувати зручний та інтуїтивно зрозумілий інтерфейс для користувачів.

2 АНАЛІЗ ВИКОРИСТАНИХ ІНСТРУМЕНТІВ ТА АРХІТЕКТУРИ ЗАСТОСУНКА

2.1 Аналіз вимог

Першим та ключовим етапом проєктування системи є аналіз вимог. Його основна мета – пошук, збір, формування та аналіз вимог до майбутньої системи, що дозволяє забезпечити її відповідність потребам користувачів, бізнес-цілям та технічним обмеженням.

Визначення функціональних і нефункціональних вимог дозволяють сформувати чітке уявлення про те, що система потрібна робити, а також які обмеження необхідно враховувати при її розробці.

Таким чином, для створення застосунку для керування фінансами необхідно зібрати вимоги, які чітко визначають його функціональність [11]. Нижче наведено основні функціональні вимоги до системи управління фінансами:

– користувач повинен мати можливість додавати нові транзакції до системи, вказуючи такі дані, як тип транзакції (доходи/витрати), сума, дата, категорія та додаткові примітки;

– користувач повинен мати можливість створювати власні категорії, що дозволить персоналізувати облік під свої потреби;

– система повинна мати функцію пошуку за різними критеріями, такими як дата, сума, категорія та ключові слова в описі;

– користувач має можливість додавати додатково опис транзакції;

– система має надавати користувачеві можливість візуалізувати дані про витрати у вигляді діаграм для наочного аналізу;

– система має аналізувати фінансові дані користувача та надавати персоналізовані рекомендації щодо оптимізації витрат та досягнення фінансових цілей;

– система повинна мати інтуїтивно зрозумілий інтерфейс, який дозволяє користувачеві легко освоїти всі функції програми без додаткової інструкції.

Окрім функціональних вимог, які визначають, що система повинна робити, необхідно також сформулювати нефункціональні вимоги, які описують якісні характеристики системи. Добре продумані нефункціональні вимоги є запорукою високої якості розробки системи, що забезпечить її довготривалу експлуатацію та задоволеність користувачів. Нижче перераховані деякі з них:

– система повинна бути стійкою до збоїв та забезпечувати збереження даних користувачів;

– система повинна захищати конфіденційність фінансової інформації користувачів;

– система повинна працювати швидко та ефективно, забезпечуючи миттєву реакцію на дії користувача;

– система повинна бути спроможна обробляти великі обсяги даних та підтримувати зростання кількості користувачів;

Завдяки детальному аналізу вимог ми отримуємо чітке розуміння того, які функції повинен виконувати фінансовий застосунок, а також які якісні характеристики він повинен мати. Це допоможе створити застосунок, який відповідатиме вимогам користувачів, буде простим та зручним у користуванні та надаватиме надійний та ефективний спосіб роботи з персональним бюджетом.

2.2 Опис архітектури застосунку

Архітектура системи загалом визначає структуру її компонентів, їхню взаємодію та принципи організації. Вона впливає на продуктивність, масштабованість, безпеку та зручність розробки й підтримки програмного забезпечення.

Існує кілька основних видів архітектур, серед яких найпоширенішими є дворівнева, трирівнева та багаторівнева архітектури. Вибір конкретної архітектури залежить від кількох факторів, таких як розмір системи, рівень захищеності, складність бізнес-логіки та вимоги до масштабованості.

Для реалізації системи керування фінансами було обрано дворівневу (клієнт-серверну) архітектуру. Візуалізацію робити даної архітектури показано на рисунку 2.1.

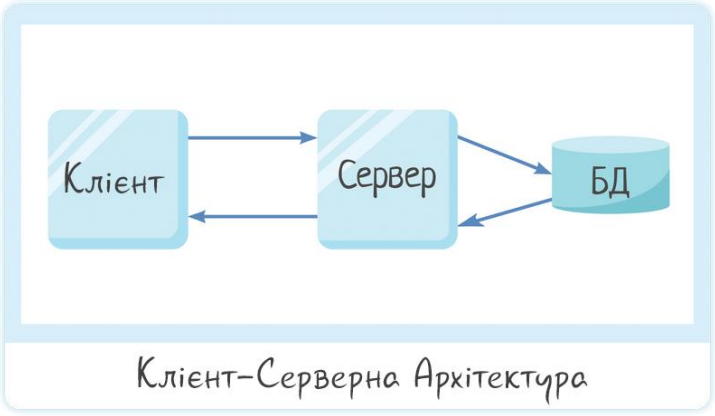

Рисунок 2.1 – Візуалізація клієнт-серверної архітектури.

Як зрозуміло з назви, клієнт-серверна архітектура складається з двох основних компонентів: клієнта та сервера. Зазвичай клієнт – це пристрій користувача (наприклад комп’ютер, смартфон або планшет), який надсилає запити до сервера через API для отримання певної інформації або для виконання певних дій. Слід зазначити, що клієнт може бути не лише фізичним пристроєм, а й автономною програмою, що взаємодіє з сервером для обробки інформації та виконання необхідних функцій.

Сервер, у свою чергу, – це більш потужний пристрій (комп’ютер, серверне обладнання тощо), який обробляє запити клієнта, зберігає та керує даними, надає доступ до ресурсів, а також виконує бізнес-логіку та реалізує функціональні вимоги системи.

Такий підхід дозволяє розподілити навантаження, зробити систему масштабованою та забезпечити безпеку даних, що є особливо важливим для фінансових застосунків.

Клієнтська частина (фронтенд) відповідає за інтерфейс користувача (UI) та його взаємодію із системою. Вона забезпечує відображення даних, отриманих із сервера, і надає інструменти для введення та обробки інформації користувачем. Серверна частина (бекенд) виконує логіку обробки запитів, управління базою даних, а також реалізує механізми автентифікації та авторизації [12–14].

2.3 Огляд використаних інструментів

Для побудови сучасного вебзастосунку слід використовувати методи та технології, що забезпечують стабільну роботу, високу продуктивність і кросплатформенність. Це означає, що система повинна коректно працювати на різних операційних системах (Windows, macOS, Linux), пристроях (настільні комп’ютери, планшети, смартфони) та веббраузерах (Chrome, Firefox, Safari, Edge тощо). Для цього слід дотримуватися принципів адаптивного дизайну, застосовувати сучасні вебфреймворки та забезпечити відповідність стандартам безпеки та доступності. Використання API для інтеграції з іншими сервісами та базами даних також сприятиме гнучкості й масштабованості системи.

2.3.1 Огляд технологій використаних у фронтенд-розробці

Для реалізації фронтенд частини було обрано бібліотеку React та платформу Node.js. React – це одна з найпопулярніших JavaScript-бібліотек для створення динамічних вебзастосунків. Вона була розроблена компанією Facebook у 2013 році, і з того часу вона стала стандартом для створення інтерфейсів користувача завдяки своїй високій ефективності та зручності. React дозволяє створювати складні інтерфейси, розбиваючи їх на невеликі

компоненти, які можна зручно повторно використовувати та тестувати. Спочатку бібліотека була розроблена для використання всередині веббраузерів, але згодом з’явилися додаткові рішення для роботи на мобільних пристроях (React Native) та серверних середовищах.

React дозволяє створювати адаптивний дизайн, що забезпечить коректне відображення інтерфейсу на різних пристроях (комп’ютери, планшети, смартфони). Завдяки використанню сучасних CSS технологій та інтеграції з фреймворками для стилізації, React дозволяє розробляти інтерфейси, що автоматично підлаштовуються під різні розміри екрану.

Ще однією особливістю цієї бібліотеки є віртуальний DOM (Document Object Model) – концепція програмування, у якій розробник вказує, в якому стані має знаходитись інтерфейс користувача. Це означає, що представлення інтерфейсу зберігається в пам’яті комп’ютера (віртуальний DOM) і синхронізується з оригінальним DOM за допомогою React. Коли відбувається зміна стану інтерфейсу, React спочатку оновлює віртуальний DOM, а потім порівнює його з попереднім станом, визначаючи, які частини реального DOM потрібно змінити [15, 16].

Для запуску та управління залежностями у проєкті використовується Node.js – це безкоштовне середовище виконання JavaScript-коду на сервері, що розширює можливості цієї мови, яка традиційно використовувалася лише на стороні клієнта. Оскільки Node.js є платформою з відкритим кодом, розробники з усього світу сприяють розвитку Node.js та створюють різноманітні бібліотеки та інструменти для роботи з цією платформою. Крім того, Node.js підтримує основні операційні системи, такі як Windows, Linux та macOS, що робить його кросплатформеним.

У цьому проєкті Node.js використовується лише для запуску застосунку та завантаження необхідних модулів. Для цього використовується пакетний менеджер npm (node package manager). Npm – це менеджер пакетів для Node.js, який дозволяє розробникам легко встановлювати, оновлювати та керувати

сторонніми бібліотеками та інструментами, необхідними для розробки [17–19].

Модулі дозволяють спростити розробку, оскільки вони забезпечують повторне використання коду, полегшують інтеграцію різних функцій і скорочують час розробки. Далі буде наведено короткий опис модулів, що було використано під час розробки застосунку для управління фінансами.

Chart.js – JavaScript-бібліотека для створення інтерактивних графіків і діаграм, що дозволяє ефективно візуалізувати фінансові чи статистичні дані у вигляді різних типів графіків.

Daterangepicker – інструмент для вибору діапазону дат. Він дозволяє користувачам вибирати початкову та кінцеву дату з календаря, що використовується для фільтрації даних за періодами часу.

jQuery – JavaScript-бібліотека, що фокусується на взаємодії HTML та JavaScript. Вона допомагає легко отримати доступ до будь-якого елемента DOM.

Bootstrap – фреймворк для створення адаптивних вебінтерфейсів. Він включає стилі та JavaScript-компоненти для створення кнопок, форм, навігаційних панелей, модальних вікон і інших елементів інтерфейсу, що значно спрощує розробку і дизайн вебзастосунків [20].

Google Generative AI – набір інструментів від Google для генерації та обробки контенту за допомогою штучного інтелекту.

Фронтенд частина застосунку взаємодіє з серверною через REST API, використовуючи HTTP-запити для отримання, додавання, редагування та видалення фінансових даних.

REST API – архітектурний стиль взаємодії між клієнтом і сервером через HTTP. Він базується на принципах REST (Representational State Transfer), ці принципи роблять API гнучким, масштабованим та легким у використанні. На відміну від деяких API, таких як SOAP або XML-RPC, які вимагають від розробників більше зусиль через складний синтаксис, суворі правила форматування та високе споживання ресурсів, REST API є більш простим,

гнучким та ефективним рішенням для взаємодії між клієнтом і сервером, оскільки розробник може імплементувати REST за допомогою будь-якої мови програмування, зокрема: JavaScript, Python, Java, C#, PHP тощо. Завдяки цьому REST API є універсальним підходом для створення вебсервісів незалежно від стеку технологій [21].

2.3.2 Ключові інструменти для розробки бекенду

Для забезпечення функціональності REST API на серверній частині проєкту було використано платформу .NET 8. Вибір цієї технології зумовлений її високою продуктивністю, масштабованістю та надійністю, що є критичними для обробки великого обсягу фінансових даних. .NET 8 забезпечує необхідні можливості для ефективного створення вебсервісів і API, що відповідають вимогам сучасних фінансових застосунків.

.NET – це кросплатформена, відкрита екосистема для розробки програмного забезпечення, створена компанією Microsoft. Вона дозволяє будувати веб, мобільні, десктопні, хмарні та AI-застосунки за допомогою різних мов програмування, таких як C#, F#, VB.NET. Сьогодні .NET .активно використовується в Unity для створення ігор, а також дозволяє створювати масштабовані вебзастосунки та API.

Для забезпечення ефективної роботи з базою даних у рамках розробки API на серверній частині проєкту було обрано використання Entity Framework. Entity Framework (EF) – це потужна ORM (Object-Relational Mapping) технологія для .NET, яка дозволяє взаємодіяти з реляційними базами даних, такими як SQL Server, за допомогою об’єктно-орієнтованих концепцій. EF спрощує процес роботи з даними, забезпечуючи автоматичне створення, оновлення та видалення записів у базі даних через об’єкти C#, що значно підвищує ефективність розробки та знижує ймовірність помилок [22–24].

Розглянемо, як налаштувати базу даних за допомогою Entity Framework у проєкті, де SQL Server виступає основною СУБД для зберігання фінансових даних.

Крок 1. Створюємо клас Transaction (рис. 2.2). Цей клас представляє транзакцію, а кожна транзакція в базі даних буде зберігатися як запис з відповідними полями (наприклад, сума, категорія, опис).

```
11 references
public class Transaction
{
    3 references
    public int Id { get; set; }
    0 references
    public required string Name { get; set; }
    0 references
    public string? Description { get; set; }
    0 references
    public decimal Amount { get; set; }
    3 references
    public int CategoryId { get; set; }
    [Column(TypeName = "datetime")]
    5 references
    public DateTime TransactionDate { get; set; } = DateTime.Now;
    [Column(TypeName = "datetime")]
    0 references
    public DateTime CreatedDate {  get; set; } = DateTime.Now;
    [Column(TypeName = "datetime")]
    0 references
    public DateTime? ModifiedDate { get; set; }
    6 references
    public bool IsDeleted { get; set; }
    8 references
    public required Category Category { get; set; }
    5 references
    public string UserId { get; set; }
    9 references
    public User User { get; set; } = null!;
}
```

Рисунок 2.2 – Створення класу «Transaction»

Крок 2. Створюємо контекст бази даних – клас MoneyCountDbContext, він визначає зв’язок між класами Transaction, Category, User та таблицями в SQL Server. За допомогою DbSet<Transaction> і DbSet<Category> визначаємо колекції, які відповідатимуть цим таблицям (рис. 2.3). Слід зазначити, що префікс Identity у батьківському класі вказує на те, що даний контекст призначений для роботи з користувачами. Він автоматично створює таблицю Users, що звільняє розробника від необхідності вручну налаштовувати її.

Крок 3. За допомогою методу OnModelCreating задаємо взаємозв’язки між сутностями Transaction, User і Category (рис. 2.3).

```
public class MoneyCountDbContext(DbContextOptions<MoneyCountDbContext> options) : IdentityDbContext<User>(options)
{
    11 references
    public required DbSet<Transaction> Transactions { get; set; }
    15 references
    public required DbSet<Category> Categories { get; set; }

    0 references
    protected override void OnModelCreating(ModelBuilder modelBuilder)
    {
        modelBuilder.Entity<Transaction>()
            .HasOne(t => t.User)
            .WithMany(u => u.Transactions)
            .HasForeignKey(t => t.UserId)
            .OnDelete(DeleteBehavior.NoAction);

        modelBuilder.Entity<Category>()
            .HasOne(c => c.User)
            .WithMany(u => u.Categories)
            .HasForeignKey(c => c.UserId)
            .OnDelete(DeleteBehavior.NoAction);

        modelBuilder.Entity<Transaction>()
            .HasOne(t => t.Category)
            .WithMany()
            .HasForeignKey(t => t.CategoryId)
            .OnDelete(DeleteBehavior.NoAction);

        base.OnModelCreating(modelBuilder);
    }
}
```

Рисунок 2.3 – Створення контексту бази даних

Крок 4. Налаштовуємо підключення до бази даних (рис. 2.4). Рядок підключення до SQL Server знаходиться у файлі конфігурації застосунку та реєструє MoneyCountDbContext у контейнері залежностей, використовуючи Entity Framework Core для роботи з базою даних через SQL Server.

```
var connectionString = builder.Configuration.GetConnectionString("SqlServerConnection");
builder.Services.AddDbContext<TransactionsDbContext>(
    options => options.UseSqlServer(connectionString));
```

Рисунок 2.4 – Підключення до бази даних

Крок 5.Після налаштування класів та контексту бази даних потрібно створити міграцію для генерування SQL-коду, який створює таблиці в базі даних. (рис. 2.5).

```
PM> Add-Migration MyFirstMigration
Build started...
Build succeeded.
PM> Update-Database
```

Рисунок 2.5 – Додавання міграції та оновлення бази даних

Після опису процесу створення та налаштування бази даних можна зробити висновок, що використання Entity Framework дозволяє реалізувати CRUD-операції без необхідності глибоких знань SQL, що автоматизує багато операцій, таких як створення та оновлення записів та дозволяє зосередитись на основній бізнес-логіці застосунку. Крім того, можливість працювати з SQL Server через об’єктно-орієнтовану модель забезпечує більшу гнучкість і зручність при обробці фінансових даних, що критично важливо для системи керування фінансами.

2.4 Призначення моделей даних у системі

Моделі даних – це структури, що визначають формат та спосіб зберігання та передачі інформації в системі. Вони визначають чітку схему для інформації, що зберігається і передається між клієнтом і сервером. Це забезпечує узгодженість у форматі переданих даних та унеможливлює помилки, пов’язані з невідповідністю структури [25].

Для прикладу доцільності використання моделей можна розглянути класи Category (рис. 2.6) та CreateCategoryModel (рис. 2.7).

Category – клас, що представляє сутність категорії у базі даних. Він містить усі основні поля, необхідні для роботи з категоріями: ідентифікатор (Id), назву (Name), піктограму (Icon), а також службові мета-дані (CreatedDate, ModifiedDate) для збереження історії змін. Додатково, модель містить поле IsDeleted, що дозволяє реалізувати м’яке видалення категорій, а також поля UserId та User для зв’язку з користувачем, якому належить ця категорія.

CreateCategoryModel – клас-модель, призначена для створення нової категорії. Вона містить лише необхідні дані: Name та Icon, що спрощує передачу інформації у запитах на створення нової категорії. Використання цієї моделі дозволяє зменшити обсяг передаваних даних і підвищити безпеку,

оскільки інші властивості сутності Category, такі як Id, CreatedDate чи UserId, не змінюються на рівні створення нової категорії.

```
public class Category
{
    8 references
    public int Id { get; set; }
    1 reference
    public string Name { get; set; }
    0 references
    public string? Icon { get; set; }
    0 references
    public DateTime CreatedDate { get; set; } = DateTime.Now;
    0 references
    public DateTime? ModifiedDate { get; set; }
    12 references
    public bool IsDeleted { get; set; }
    5 references
    public string UserId { get; set; }
    18 references
    public User User { get; set; } = null!;
}
```

Рисунок 2.6 – Додавання міграції та оновлення бази даних

```
public class CreateCategoryModel
{

    public required string Name { get; set; }

    public string? Icon { get; set; }
}
```

Рисунок 2.7 – Додавання міграції та оновлення бази даних

Основними причинами використання окремих моделей є безпека, зручність та гнучкість. Наприклад, у списках міститься лише необхідна інформація, тоді як детальні запити повертають розширені дані. Це дозволяє уникнути передачі зайвих відомостей клієнту (наприклад, поля CreatedDate та ModifiedDate не надсилаються при створенні запису).

Аналогічний підхід використовується і для інших класів-об’єктів. У проєкті використовуються різні моделі для транзакцій, категорій та користувачів, де кожна з них має своє призначення. Це дозволяє зробити API

більш структурованим, зрозумілим та ефективним. Перелік усіх використаних моделей та API наведено в розділі програмної реалізації проєкту.

2.5 Взаємодія з Gemini API

Для надання користувачам персоналізованих порад щодо покращення їхнього фінансового стану в застосунку було інтегровано можливості штучного інтелекту. Використовується Gemini API – набір ендпоінтів до сімейства моделей Gemini від корпорації Google, що забезпечує генерацію тексту, аналіз запитів та роботу з контекстом.

Для взаємодії з API застосовуються клієнтські SDK (Software Development Kit), які дозволяють легко інтегрувати можливості штучного інтелекту у веб та мобільні застосунки. За допомогою SDK можна додати функціонал для генерації тексту та коду, створити чат з ШІ або реалізувати інші інтелектуальні сервіси.

Слід зазначити, що використання Google AI Studio є безкоштовним у всіх доступних країнах. Для безкоштовного тарифу діють такі обмеження:

– 15 запитів на хвилину (RPM);

– 1 мільйон токенів на хвилину (TPM);

– 1 500 запитів на день (RPD).

Токен – це базова одиниця даних, яка використовується моделями ШІ для обробки текстової інформації. Зазвичай 100 токенів відповідають приблизно 75 словам.

Розглянемо покрокову інтеграцію Gemini API до вебзастосунку:

– реєструємо обліковий запис та отримаємо API-ключ на сайті Google AI Studio;

– встановлюємо необхідне SDK, для Node.js використано команду, показану на рисунку 2.8;

```
➜  ~ npm install @google/generative-ai
```

Рисунок 2.8 – Встановлення інструментів для роботи з Gemini API

– використовуємо API для генерації тексту (рис. 2.9). Для цього необхідно ввести згенерований раніше API-ключ, обрати тип моделі та створити її екземпляр. Останній крок – викликати функцію для генерації контенту за запитом, що є параметром функції.

```
import { GoogleGenerativeAI } from "@google/generative-ai";

const genAI = new GoogleGenerativeAI("API_KEY");

async function generateText() {
    const model = genAI.getGenerativeModel({ model: "gemini-pro" });
    const result = await model.generateContent("How can I improve my budget?");
    console.log(result.response.text());
}

generateText();
```

Рисунок 2.9 – Приклад запиту на генерацію тексту

Підбиваючи висновки, слід зазначити, що Gemini API надає широкі можливості для інтеграції штучного інтелекту без додаткових витрат на запити та налаштування моделей. Це дозволяє використовувати API як для невеликих, так і для масштабних завдань, забезпечуючи при цьому стабільну роботу в рамках безкоштовних лімітів [26].

2.6 Обґрунтування вибору середовища програмної реалізації

Для реалізації бекенд частини програмного забезпечення, написаної мовою програмування C#, було використано інтегроване середовище розробки Visual Studio 2022, а для фронтенд частини – редактор коду Visual Studio Code.

Visual Studio – потужне IDE, створене компанією Microsoft, яке надає широкий набір інструментів для розробки вебсайтів, мобільних і десктопних застосунків. Перша версія вийшла ще у 1997 році й стала однією з перших спроб об’єднати кілька мов програмування в єдиному середовищі.

З часом Visual Studio значно еволюціонувало, отримавши підтримку нових мов, фреймворків і технологій. Сьогодні воно є основним інструментом для розробки на платформі .NET, а також підтримує широкий спектр мов програмування, зокрема C#, C++, Python, JavaScript та інші.

Однією з ключових переваг середовища є інтеграція з Microsoft Azure та SQL Server, що спрощує розробку хмарних застосунків і роботу з базами даних. Крім того, Visual Studio підтримує юніт-тестування через MSTest, NUnit, xUnit, а завдяки системі керування пакетами NuGet розробники можуть легко додавати необхідні бібліотеки та модулі до проєкту.

Visual Studio забезпечує зручну командну роботу, дозволяючи розробникам спільно працювати в режимі реального часу та керувати версіями коду безпосередньо в IDE. Інтегрований Git-менеджер надає можливість створювати та перемикатися між гілками, виконувати коміти, зливати зміни та переглядати історію проєкту.

Завдяки повній підтримці .NET середовище містить широкий набір інструментів, що спрощує розробку на C#. Вбудований редактор коду з автодоповненням (IntelliSense) і рефакторингом, зрозумілий інтерфейс (рис. 2.10), швидкий доступ до інструментів, можливість гнучкого налаштування середовища та велика кількість шаблонів проєктів – все це робить Visual Studio одним із найзручніших середовищ розробки для C# [27].

Його широкий функціонал, інтеграція з .NET та підтримка сучасних технологій дозволяють створювати ефективні та масштабовані програмні рішення. Завдяки Visual Studio розробники можуть зосередитися на якості коду та швидко реалізовувати свої ідеї в різних сферах програмування.

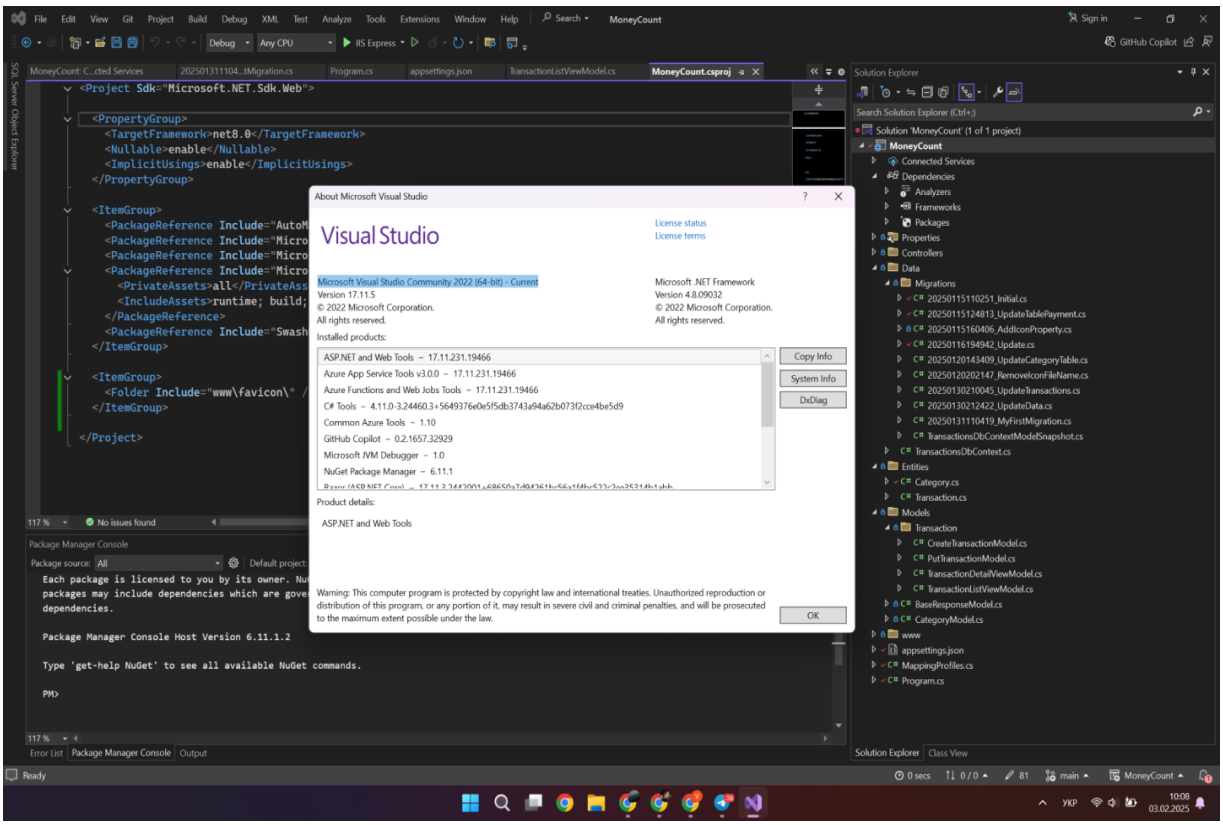

Рисунок 2.10 – Інтерфейс IDE Visual Studio

Як було зазначено вище, для фронтенд частини, що написана за допомогою JavaScript-бібліотеки React, було використано редактор коду, Visual Studio Code. Його перша версія вийшла у 2015 році, і він швидко завоював популярність серед розробників завдяки своїй швидкодії, розширюваності та кросплатформеності.

На відміну від Visual Studio, VS Code не є повноцінним IDE, бо за замовчуванням він не містить вбудованих інструментів для компіляції, налагодження та управління проєктами, які є у традиційних інтегрованих середовищах розробки. Натомість VS Code більше орієнтований на гнучкість, легкість та швидкість роботи, що ідеально підходить для фронтенд-розробки, написання скриптів та кросплатформної роботи.

Однією з ключових переваг є його розширюваність – завдяки вбудованому менеджеру розширень розробники можуть легко додавати необхідні функціональні можливості, такі як підтримка фреймворків, додаткові засоби налагодження та інтеграція з системами контролю версій.

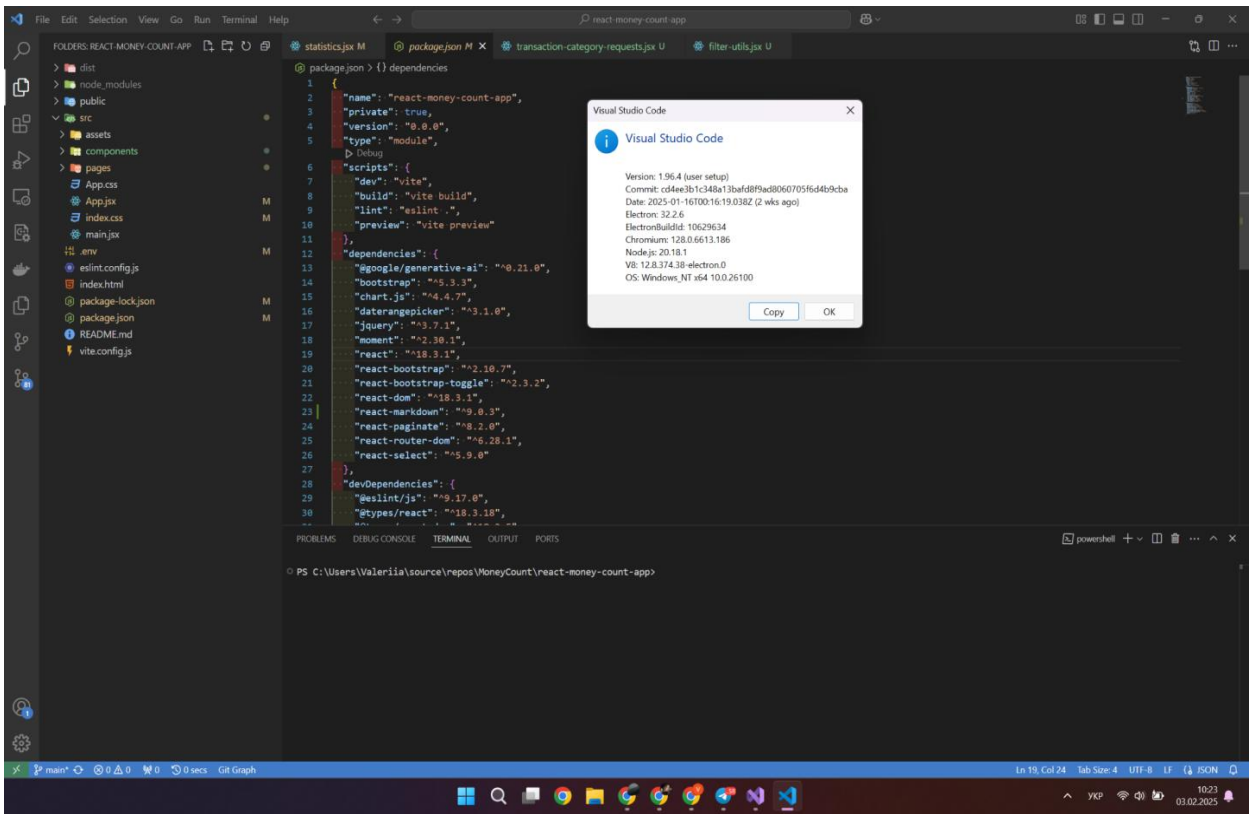

Рисунок 2.11 – Інтерфейс редактору коду VS Code

Для командної роботи VS Code підтримує Git та інші системи керування версіями, що дозволяє зручно відстежувати зміни, працювати з гілками та здійснювати спільну розробку.

Завдяки своїй гнучкості, кросплатформності та великій кількості розширень Visual Studio Code став одним із найпопулярніших редакторів коду у світі. Його швидкість, мінімалістичний інтерфейс (рис. 2.11) та розширені можливості роблять його ідеальним вибором для роботи з фронтенд-технологіями, веброзробкою та багатьма іншими сферами програмування [28–29].

3 ПРОГРАМНА РЕАЛІЗАЦІЯ ЗАСТОСУНКУ ДЛЯ КЕРУВАННЯ ФІНАНСАМИ

3.1 Програмна реалізація застосунку

Вебзастосунок призначений для управління особистими фінансами та надання рекомендацій щодо покращення фінансового стану користувача. Архітектура застосунку поділена на дві частини: бекенд, розроблений на C# із використанням бібліотек EntityFrameworkCore, і фронтенд, створений на основі JavaScript-бібліотеки React у поєднанні з Bootstrap та іншими модулями.

У цьому розділі також описано структуру створених API, їх функціональність, використані моделі даних і логіку роботи контролерів для ефективної взаємодії між клієнтом і сервером.

Слід зазначити, що розроблене API базується на архітектурі REST, що забезпечує ефективну взаємодію між клієнтською та серверною частиною застосунку, серед яких:

– клієнт-серверна архітектура;

– безстанність (кожен запит містить усю необхідну інформацію, а сервер не зберігає стан сесії між запитами);

– єдиний інтерфейс (використання стандартних HTTP-методів та форматів обміну даними, таких як JSON або XML).

Проте принципи кешованості та багаторівневої (шарової) системи в цій реалізації не використовувалися.

Для забезпечення структурованості та ефективності API використано кілька різних моделей для транзакцій, кожна з яких має своє призначення. Це дозволяє спростити роботу з даними й уникнути передачі зайвої інформації. На рисунку 3.1 можна побачити список усіх файлів, які відповідають за певну модель даних.

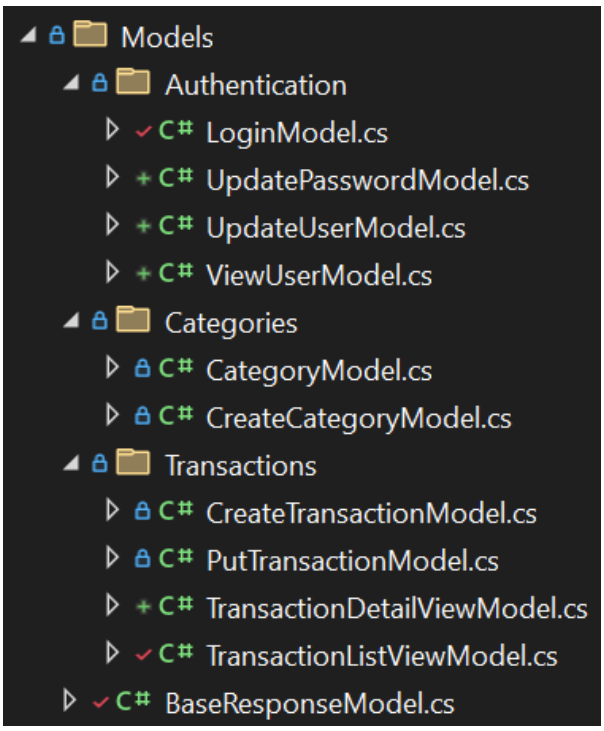

Рисунок 3.1 – Структура файлів моделей даних системи

Розглянемо коротке пояснення для кожної моделі у таблиці 3.1.

Таблиця 3.1 – Список використаних моделей

Модель	**Поля**	**Призначення**	**Використання в API**
1	**2**	**3**	**4**
Category	Id, Name, Icon, CreatedDate, ModifiedDate	Основна модель категорії, що відповідає таблиці в базі даних	Зберігання інформації про категорії
CreateCategoryModel	Name (обов’язкове), Icon (необов’язкове)	Створення нової категорії, забезпечує валідацію вхідних даних	POST /api/Categories
CategoryModel	Id, Name, Icon	Відображення інформації про категорії, розширює CreateCategoryModel	GET /api/Categories, GET /api/Categories/{id}

Продовження таблиці 3.1

1	2	3	4
Transaction	Id, Name, Amount, PaymentDate, Category, CreatedDate, ModifiedDate	Основна модель транзакції, що відображає таблицю в базі даних	Зберігання транзакцій у базі даних
CreateTransaction Model	Name, Amount, PaymentDate, CategoryId (усі з валідацією [Required])	Створення нової транзакції, забезпечує коректність введених даних	POST /api/Transactions
PutTransaction Model	Id, Name, Amount, PaymentDate, CategoryId	Оновлення існуючої транзакції, успадковує CreateTransaction Model і додає Id	PUT /api/Transactions/{id}
TransactionListView Model	Id, Name, Amount, PaymentDate, Category	Відображення списку транзакцій, містить лише необхідні для списку поля	GET /api/Transactions,GET /api/Transactions/get-all-transactions
ViewUserModel	Id, Name, Email	Відображення головної інформації про користувача, використовується для домашньої сторінки	GET /api/ account/home/ {email}
LoginModel	Email, Password, Remember	Автентифікація користувача в систему	POST /api/account/login

Продовження таблиці 3.1

1	2	3	4
TransactionDetailViewModel	Id, Name, Amount, PaymentDate, Category, Desctiption	Відображення детальної інформації про категорії, розширює TransactionListViewModel	GET /api/Transactions/{id}
BaseResponseModel	Status, Message, Data	Базова модель відповіді для API, яка містить статус, повідомлення, дані	Використовується для формування відповіді на запити API
User	Name, CreatedDate, LastLogin, ModifiedDate, IsAdmin, Transactions, Categories, Id, UserName, Email, PasswordHash	Основна модель користувача, що відображає таблицю в базі даних	Зберігання інформації про користувачів
UpdatePasswordModel	OldPassword, NewPassword	Оновлення паролю користувача	PUT /api/account/update-password

Основними перевагами використання окремих моделей є розділення відповідальності, кожна модель використовується лише для своєї конкретної задачі (запис у базі даних, створення, оновлення, відображення), списки не містять зайвої інформації, а детальні запити повертають лише необхідні поля.

Для роботи з розглянутими моделями у серверній частині були реалізовані відповідні контролери, що обробляють CRUD-запити до API. Завдяки чітко визначеній модульності API контролери забезпечують ефективне керування ресурсами. Кожен контролер відповідає за обробку

відповідних HTTP-запитів (GET, POST, PUT, DELETE), забезпечуючи взаємодію між клієнтською частиною застосунку та базою даних. Контролери містять логіку для перевірки вхідних даних, виклику сервісів для обробки бізнес-логіки та формування коректних відповідей API [30].

TransactionsController відповідає за керування транзакціями. Він реалізує функціонал для створення, перегляду, оновлення та видалення транзакцій. CategoriesController забезпечує управління категоріями витрат. Контролер дозволяє додавати нові категорії, редагувати наявні, видаляти їх, а також отримувати детальну інформацію про окрему категорію. AccountController відповідає за керування обліковими записами користувачів. Він дозволяє створювати та редагувати профілі користувачів, отримувати інформацію про обліковий запис, а також може містити функції зміни пароля, оновлення особистих даних або автентифікації користувача.

Уся взаємодія клієнта із сервером здійснюється через набір ендпоінтів, що охоплюють основні функції управління фінансами. У таблицях 3.2 – 3.4 буде наведено перелік ендпоінтів для керування даними у вебзастосунку.

Таблиця 3.2 – Список API-запитів для управління категоріями

Метод	**Ендпоінт**	**Опис**
GET	/api/categories	Отримання списку всіх категорій витрат
POST	/api/categories	Створення нової категорії з вказаними параметрами
PUT	/api/categories	Оновлення даних існуючої категорії
DELETE	/api/categories/	Видалення категорії за ідентифікатором
GET	/api/categories/{id}	Отримання детальної інформації про категорію за ідентифікатором
POST	/api/categories/upload-category-icon	Завантаження іконки для категорії

Таблиця 3.3 – Список API-запитів для управління транзакціями

Метод	**Ендпоінт**	**Опис**
1	**2**	**3**
GET	/api/transactions/get-all-transactions	Отримання всіх транзакцій без фільтрації
GET	/api/transactions	Отримання списку транзакцій з фільтрацією та сортуванням
POST	/api/transactions	Створення нової транзакції
PUT	/api/transactions	Оновлення даних існуючої транзакції
DELETE	/api/transactions	Видалення транзакції за вказаним ідентифікатором
GET	/api/transactions/{id}	Отримання детальної інформації про транзакцію за ідентифікатором

Таблиця 3.4 – Список API-запитів для управління користувачами

Метод	**Ендпоінт**	**Опис**
1	**2**	**3**
POST	/api/account/register	Реєстрація нового користувача.
POST	/api/account/login	Вхід користувача в систему.
POST	/api/account/logout	Вихід користувача з системи
GET	/api/account/admin	Перевіряє, чи є користувач адміністратором, і повертає відповідну відповідь
GET	/api/account/home/{email}	Отримує інформацію про користувача за його email
GET	/api/account/checkuser	Перевіряє, чи автентифікований користувач, і повертає його статус
PUT	/api/account/update-user	Оновлення інформації про користувача

Продовження таблиці 3.4

1	2	3
PUT	/api/account/update-password	Оновлення паролю користувача
DELETE	/api/account/delete-user/ {id}	Видалення користувача за вказаним ідентифікатором

Для тестування розроблених API-ендпоінтів у проєкті використовувався інструмент Swagger, що є одним із найпопулярніших рішень для документування та інтерактивної перевірки REST API. Він надає вебінтерфейс для перегляду доступних ендпоінтів, їх параметрів, запитів і відповідей.

Swagger дозволяє перевірити правильність переданих параметрів, форматів даних і отриманих відповідей, що допомагає виявити помилки ще на етапі розробки. Наявність зрозумілого інтерфейсу (рис. 3.2) полегшує роботу не лише розробникам, а й тестувальникам чи іншим членам команди, які можуть взаємодіяти з API без глибоких технічних знань.

Розділення логіки на окремі моделі та контролери забезпечує стабільну роботу API, спрощує підтримку коду та гарантує коректну функціональність компонентів у різних сценаріях використання.

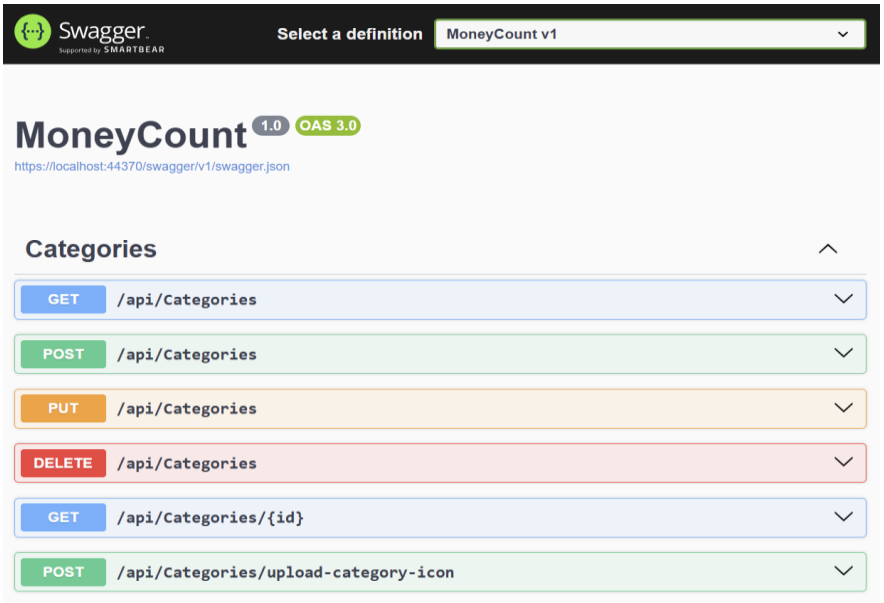

Рисунок 3.2 – Вебінтерфейс Swagger для тестування API

Для реалізації роботи з базою даних було використано вбудовану систему для менеджменту реляційними базами даних – SQL Server Object Explorer, яка дозволяє переглядати, змінювати структуру таблиць, виконувати SQL-запити та керувати базами даних безпосередньо в середовищі Visual Studio.

3.2 Опис сторінок та компонентів системи

Після створення серверної частини та API необхідно створити клієнтську частину: сторінки та компоненти, з якими буде взаємодіяти користувач.

Головною задумкою є простота використання, зрозумілий інтерфейс, висока швидкість завантаження, адаптивний дизайн для коректної роботи на різних пристроях, інтуїтивна навігація, а також забезпечення доступності для користувачів із різними потребами.

Компоненти сторінки – це окремі частини інтерфейсу користувача, які разом формують повноцінну вебсторінку. У випадку з React, компоненти – це незалежні, багаторазові блоки коду, які відповідають за відображення певної частини інтерфейсу та логіку його роботи. Файлову структуру компонентів вебзастосунку показано на рисунку 3.3 [31].

Отже, розглянемо основні компоненти застосунку.

Компонент Logout забезпечує можливість виходу з акаунта з будь-якої сторінки. Це реалізується шляхом надсилання запиту до сервера під час створення компонента.

Далі розглянемо компоненти, необхідні для коректної роботи сторінки Categories, яка дозволяє користувачеві створювати, редагувати та видаляти категорії.

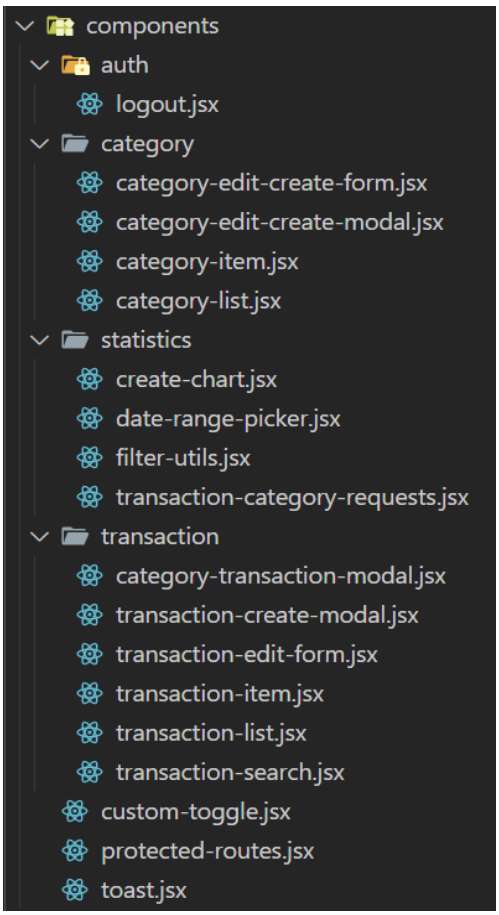

Рисунок 3.3 – Структура компонентів вебзастосунку

Category-edit-create-modal відповідає за відображення модального вікна для взаємодії з категорією. Воно відкривається при створенні нової або редагуванні наявної категорії витрат.

Category-edit-create-form – це форма, що містить набір запитів для виконання CRUD-операцій із категоріями, містить наступні поля: піктограма, де користувач може завантажити зображення для категорії та ім’я – текстове поле для введення назви категорії.

Category-list – компонент, що відображає список усіх створених користувачем категорій. Він виконує GET-запит до ендпоінту /api/Categories для отримання актуальних даних про категорії та динамічно оновлює інтерфейс.

Category-item – компонент, що представляє окрему категорію у списку. Він містить іконку та назву категорії, окрім цього, Category-item є

інтерактивним елементом: при натисканні відкривається Category-edit-create-modal, що дозволяє редагувати або видаляти категорію.

Далі розглянемо компоненти, необхідні для роботи сторінки Statistics. Create-chart містить три функції, що відповідають за побудову діаграм: аналіз витрат за певний період, перегляд діаграми доходів та порівняння доходів і витрат за останні пів року.

Daterangepicker – окремий компонент, створений для роботи з модулем daterangepicker. Він дозволяє користувачу обирати діапазон дат і приймає onDateChange() – callback-функцію, що викликається при зміні діапазону. Це необхідно для повторного рендерінгу діаграм витрат.

Filter-utils – компонент, що забезпечує фільтрацію транзакцій за певний період (getNewTransactionsListOnDateRange) та підрахунок загальної суми доходів або витрат (getTotalOnDateRange).

Transaction-category-requests – це компонент, що містить набір функцій для отримання даних про транзакції та категорії, необхідних для формування фінансової статистики, він виконує запити до наступних ендпоінтів: /api/transactions/get-all-transactions для отримання списку усіх транзакцій користувача та /api/categories для отримання переліку всіх доступних категорій витрат та доходів.

Розглянемо компоненти, що використовуються на сторінці Transactions. Category-transaction-modal – модальне вікно, що відображає список категорій, створених користувачем. При виборі категорії її назва оновлюється в модальному вікні створення транзакції.

Transaction-create-modal – містить модальне вікно для створення нової транзакції. Вікно включає заголовок, форму введення даних та кнопки для підтвердження або скасування дії.

Transaction-edit-form – форма для створення та редагування транзакції, що містить такі поля: сума, ім’я, опис, категорія, дата транзакції. Вона забезпечує валідацію введених даних і передає їх на сервер для подальшого збереження.

Transaction-list – список усіх транзакцій, згрупованих за датою створення. Кожен запис містить ключову інформацію про транзакцію, а саме ім’я, піктограму категорії, дату та суму транзакції, завдяки чому можна робити фільтрацію транзакцій у списку. Також Transaction-list містить ReactPaginate – компонент для пагінації сторінок, що дозволяє користувачеві зручно переміщатися між списками транзакцій. Він автоматично розбиває довгий список на менші частини, покращуючи навігацію та швидкість завантаження.

Transaction-search – компонент для пошуку транзакцій за ім’ям, датою та сумою. Він забезпечує швидку фільтрацію списку транзакцій, що дозволяє користувачеві знаходити потрібні операції без необхідності перегортати сторінки.

Окрім основних компонентів, застосунок також використовує:

Custom-toggle – компонент для створення кнопки-перемикача, що приймає параметри кольору та значення, які будуть на ній відображені. Використовує Toggle із модуля react-bootstrap-toggle.

Toast – компонент для відображення спливаючих повідомлень. Використовує Toast із react-bootstrap та інформує користувача про помилки чи інші події.

ProtectedRoute перевіряє, чи автентифікований користувач перед наданням доступу до сторінок, які вимагають авторизації, таких як сторінка транзакцій або домашня сторінка. Для цього маршрути, що потребують автентифікації, необхідно обгорнути в захищений компонент, як показано на рисунку 3.4. У разі відсутності авторизації користувача буде перенаправлено на сторінку входу.

Описані вище компоненти відіграють ключову роль у структурі вебзастосунку, забезпечуючи взаємодію користувача з системою, обробку даних та захист маршрутів.

```
<Route element={<ProtectedRoutes />}>
    <Route path="/" element={<Home />} />
    <Route path="/admin" element={<Admin />} />
    <Route path="/transactions" element={<Transactions />} />
    <Route path="/categories" element={<Categories />} />
    <Route path="/statistics" element={<Statistics />} />
    <Route path="/edit/:Transactionid" element={<EditTransaction />} />
    <Route path="/logout" element={<LogoutPage />} />
</Route>
```

Рисунок 3.4 – Використання компоненту Protected-routes

Вони відповідають за відображення даних, обробку запитів і взаємодію з API, а також сприяють перевикористанню коду. Тепер розглянемо, як ці компоненти інтегруються у структуру основних сторінок вебзастосунку. Файлова структура сторінок вебзастосунку показана на рисунку 3.5.

Далі буде наведено короткий опис функціональності кожної зі сторінок вебзастосунку.

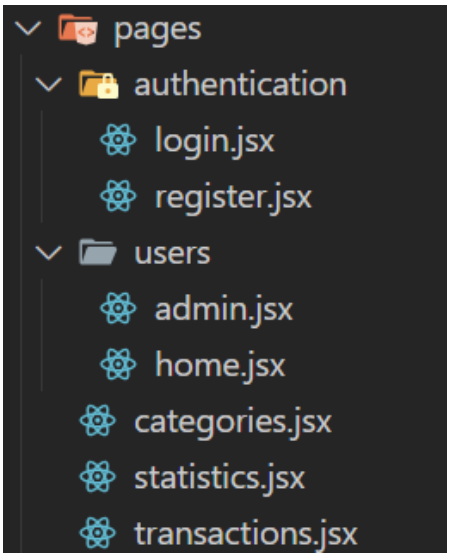

Рисунок 3.5 – Використання компоненту Protected-routes

Home – це коренева сторінка для автентифікованого користувача, де він може переглядати та редагувати свої особисті дані, зокрема ім’я, пароль та email. Для цього передбачено можливість надсилання запитів: PUT

/api/account/update-user, /api/account/update-password, для оновлення інформації та GET /api/account/home/{email} для її отримання.

Admin – сторінка для привілейованих користувачів, таких як адміністратори. Вона містить таблицю з переліком усіх зареєстрованих користувачів, дозволяючи переглядати їхні дані.

Categories – сторінка, що відображає список категорій, раніше описаних у відповідних компонентах. Також тут доступна функція створення нових категорій через модальне вікно.

Statistics – сторінка, що надає аналітичні інструменти для покращення фінансового стану користувача. Вона містить інтерактивні діаграми, реалізовані в компоненті create-chart, а також DateRangePicker для вибору періоду аналізу фінансових даних. Крім того, сторінка підтримує інтеграцію з Gemini API, що дозволяє отримувати персоналізовані фінансові поради.

Transactions – сторінка для управління фінансовими операціями. Тут можна переглядати список транзакцій, згрупованих за датами, фільтрувати за різними параметрами, а також створювати нові записи через модальне вікно.

У результаті реалізації клієнтської частини вебзастосунку було створено зручний та інтерактивний інтерфейс, що забезпечує користувачам можливість ефективно керувати фінансами. Реалізовані сторінки для авторизації, управління категоріями, перегляду статистики та управління транзакціями дозволяють швидко отримувати доступ до необхідної інформації. Завдяки використанню компонентного підходу код залишається гнучким, масштабованим та придатним до перевикористання.

3.3 Сценарії роботи користувача з системою

Для ефективного розуміння процесу взаємодії користувача із застосунком доцільно описати user flow – це графічне або текстове представлення маршруту, яким користувач проходить у застосунку для

досягнення певної цілі. Це допоможе чітко визначити ключові етапи роботи, перевірити їхню логічну послідовність та забезпечити зручний досвід користувача. Діаграма на рисунку 3.6 показує, як відбувається типовий сценарій роботи користувача у системі керування фінансами.

Рисунок 3.6 – Користувацький потік

Спочатку користувач переходить до вебзастосунку. Якщо він не має облікового запису, система пропонує зареєструватися, інакше – увійти. Після введення даних для авторизації користувач отримує повідомлення про успішний вхід і автоматично переходить на домашню сторінку.

На головній сторінці можна переглянути введені під час реєстрації дані. Якщо щось було вказано некоректно, користувач має можливість змінити ім’я, email або пароль.

Далі система перевіряє, чи створені категорії витрат. Якщо вони вже є, користувач може одразу перейти до створення транзакції. Якщо категорій немає, необхідно спочатку створити хоча б одну, а потім додати фінансову операцію. Для створення транзакції користувач відкриває сторінку Transactions, натискає «Додати транзакцію», вводить суму, категорію, дату та опис, а потім зберігає запис.

Щоб отримати аналітику щодо витрат і рекомендації від системи, потрібно перейти на сторінку Statistics, вибрати період витрат і натиснути кнопку «Ask AI». Завершивши роботу, користувач може вийти із системи, натиснувши кнопку «Logout» у меню вебзастосунку.

Описаний user flow демонструє логічну послідовність дій користувача в системі керування фінансами. Така структура забезпечує зручну та інтуїтивно зрозумілу навігацію, дозволяючи користувачеві швидко виконувати необхідні операції, редагувати дані та аналізувати фінансовий стан.

3.4 Інструкція користувача для використання застосунку

У цьому розділі надається інструкція щодо використання вебзастосунку для управління особистими фінансами. Це особливо важливо для нових користувачів, які не мають досвіду в роботі з подібними інтерфейсами чи фінансовими інструментами.

Першим кроком є запуск сервера у Visual Studio. Після успішного запуску у виводі консолі відобразиться інформація про порт, на якому працює сервер (рис. 3.7). Ці дані знадобляться для подальшого підключення та взаємодії з серверною частиною застосунку.

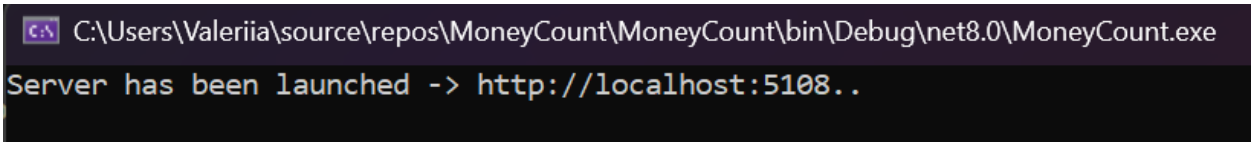

Рисунок 3.7 – Запуск сервера

Після запуску серверної частини можна перейти до клієнтської. Далі слід запустити клієнтську частину у VS Code. Це можна зробити за допомогою команди «npm run dev», після виконання якої React-застосунок буде запущено і у консолі з’явиться повідомлення із посиланням на локальний сервер (рис. 3.8). Натиснувши на це посилання, користувач відкриває застосунок у браузері.

```
PS C:\Users\Valeriia\source\repos\MoneyCount\react-money-count-app> npm run dev

> react-money-count-app@0.0.0 dev
> vite

Port 5173 is in use, trying another one...

  VITE v6.0.11  ready in 317 ms

  ➜  Local:   http://localhost:5174/
  ➜  Network: use --host to expose
  ➜  press h + enter to show help
```

Рисунок 3.8 – Запуск клієнта

Після запуску клієнтської частини користувач зможе увійти до свого облікового запису або створити новий. Якщо користувач ще не автентифікований, система автоматично перенаправить його на сторінку Login. Тут доступні два варіанти: реєстрація – для нових користувачів, які ще не мають облікового запису та вхід – для тих, хто вже зареєстрований. На рисунку 3.9 зображено сторінка авторізації користувача.

Щоб створити новий обліковий запис, необхідно перейти на сторінку реєстрації (Register) (рис. 3.10).

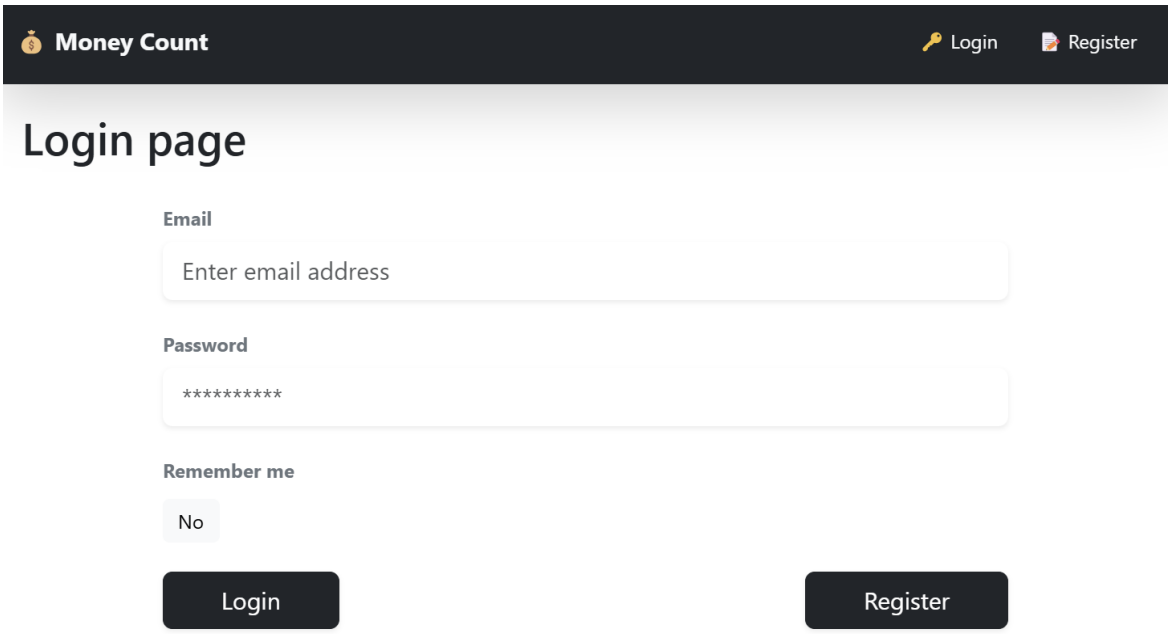

Рисунок 3.9 – Сторінка входу (Login) для неавтентифікованого користувача

Під час заповнення форми можна перевірити роботу валідації, спробувавши ввести некоректні дані, наприклад: email без домену, пароль, що складається лише з маленьких літер, цифр або без спеціальних символів, пароль, який не збігається з підтвердженням.

Якщо введені дані не відповідають вимогам, система відобразить відповідні повідомлення про помилки (рис. 3.11).

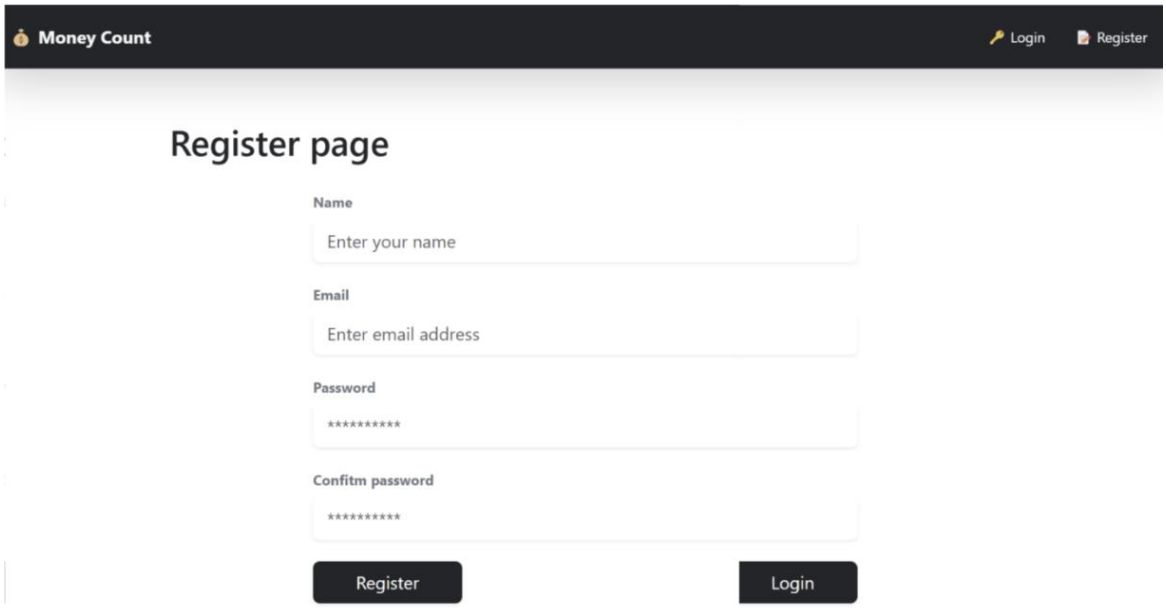

Рисунок 3.10 – Сторінка реєстрацій (Register) для користувача

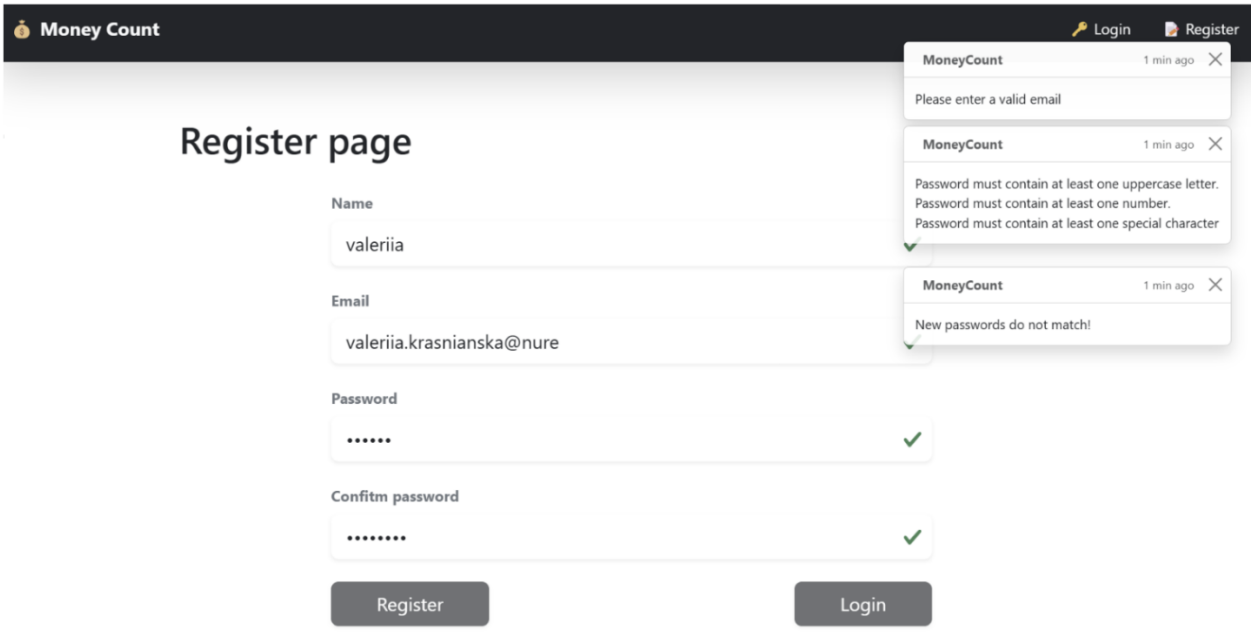

Рисунок 3.11 – Перевірка роботи верифікації при реєстрації

Після вдалої спроби створити акаунт користувач побачить повідомлення, що показано на рисунку 3.12.

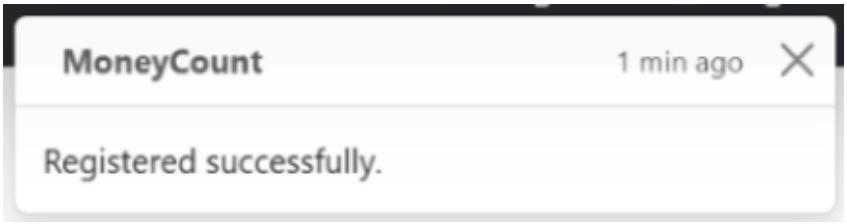

Рисунок 3.12 – Повідомлення про успішну реєстрацію

Після створення облікового запису користувач може одразу увійти в систему, використовуючи свої облікові дані. Якщо під час авторизації введені некоректні дані, система повідомить про це, вивівши повідомлення «Перевірте свої облікові дані» (рис. 3.13).

Як тільки користувач вводить правильні дані особистого облікового запису, система повідомляє про вдалу спробу авторизації (рис. 3.14).

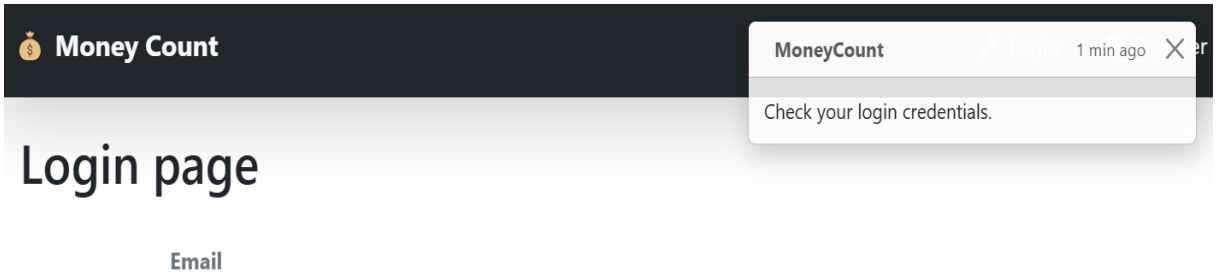

Рисунок 3.13 – Попередження про введення некоректних даних при авторизації

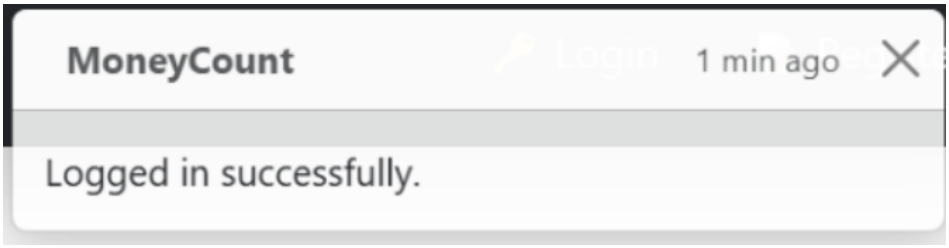

Рисунок 3.14 – Повідомлення про успішну авторизацію

Після входу в систему користувач потрапляє на свою домашню сторінку. За потреби він може змінити особисті дані, натиснувши на кнопку зі своїм ім’ям або email (рис. 3.15).

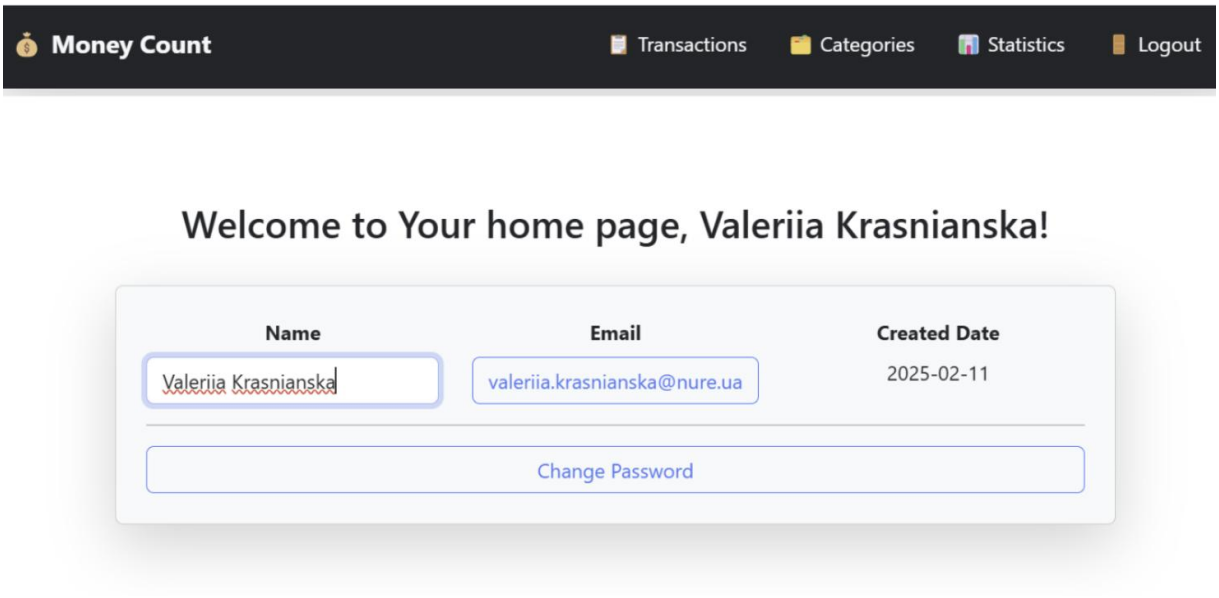

Рисунок 3.15 – Зміна особистих даних (поле «Ім’я»)

Після внесення змін система відобразить відповідне повідомлення (рис. 3.16).

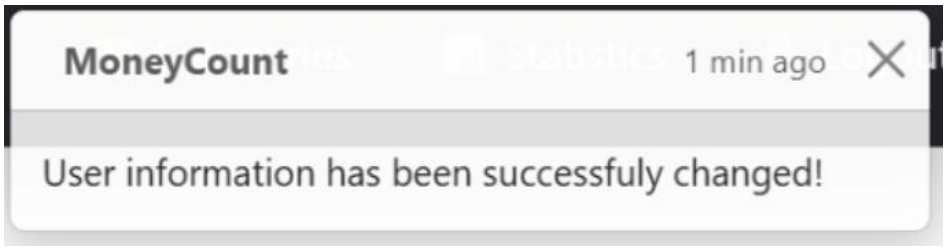

Рисунок 3.16 – Повідомлення про успішну зміну даних

Щоб змінити пароль, користувач має відкрити форму зміни паролю, ввести поточний і новий пароль, який відповідає вимогам безпеки. Під час оновлення пароля працює вбудована валідація, яка перевіряє правильність введених даних. Якщо дані не відповідають вимогам, з’явиться повідомлення про помилку (рис. 3.17).

У разі успішної зміни пароля користувач побачить підтвердження про зміну (рис. 3.18).

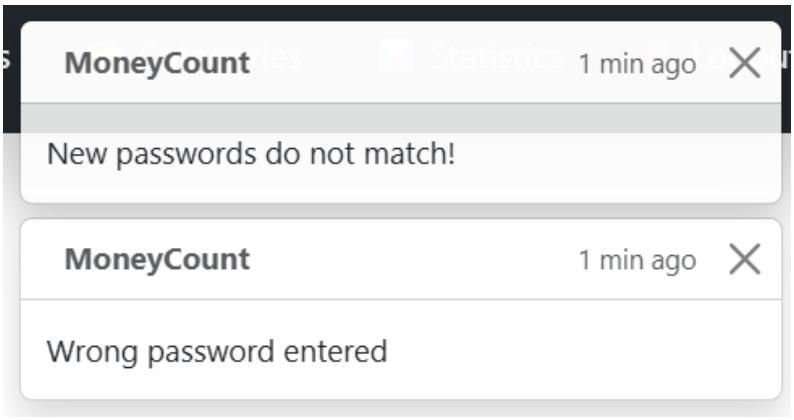

Рисунок 3.17 – Попередження про невдалу зміну паролю користувача

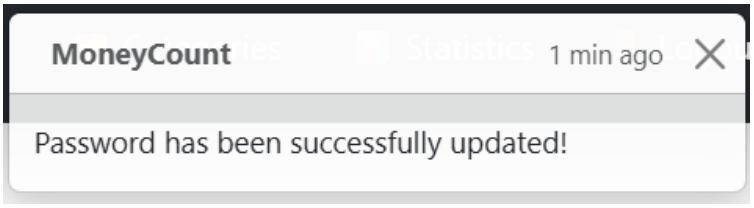

Рисунок 3.18 – Повідомлення після успішної зміни паролю

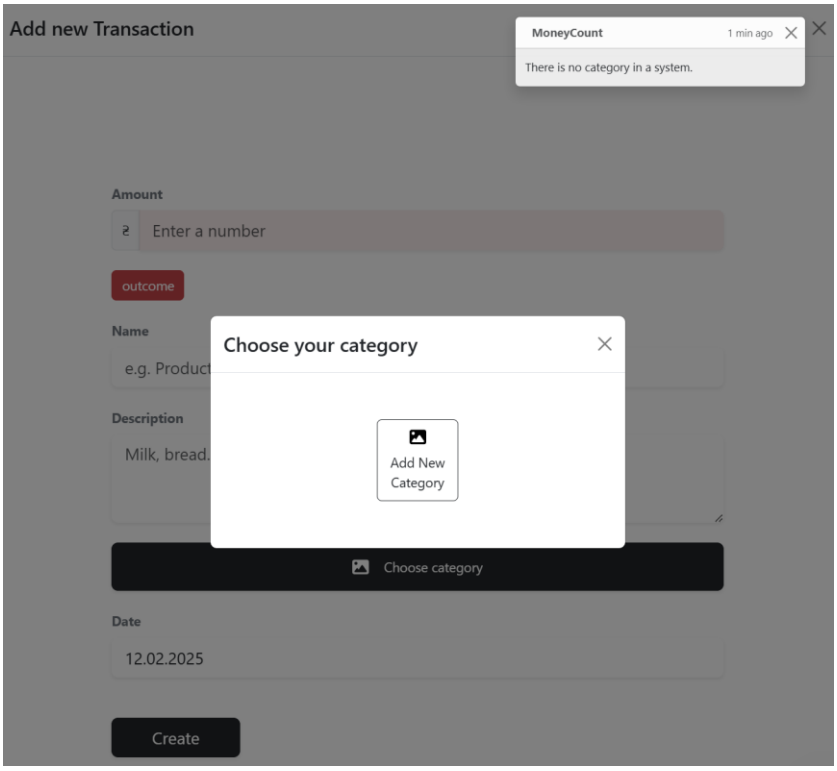

Рисунок 3.19 – Модальне вікно вибору категорії для нового користувача

Перед створенням нової фінансової транзакції необхідно додати хоча б одну категорію. Якщо новий користувач намагається додати транзакцію без створених категорій, система відобразить повідомлення: «В системі немає категорій» та запропонує створити нову (рис. 3.19).

Для зручного управління фінансами варто розподіляти витрати за категоріями. Це допоможе впорядкувати фінансові дані та полегшить аналіз доходів і витрат. Для створення нової категорії користувачеві потрібно обрати іконку у форматі .svg (опціонально) та ввести унікальну назву (рис. 3.20).

Якщо під час валідації сталась помилка, система відобразить відповідне повідомлення (рис. 3.21).

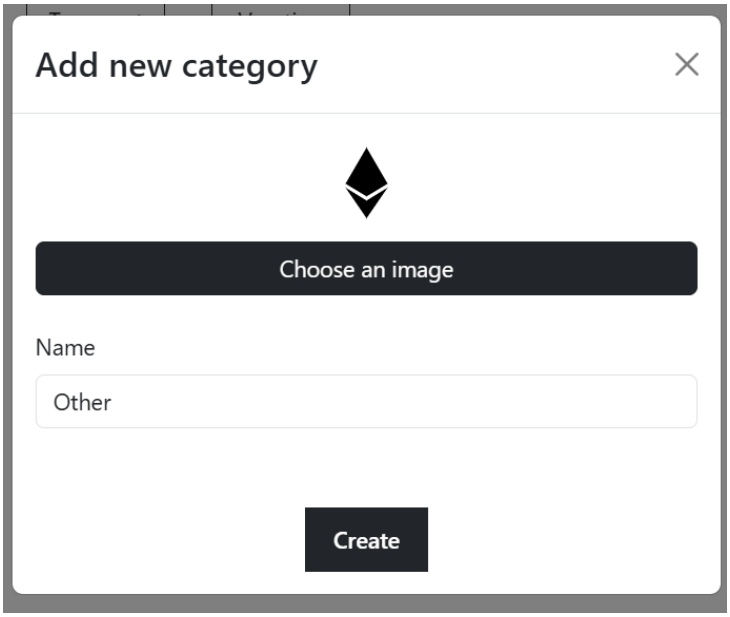

Рисунок 3.20 – Приклад створення категорії

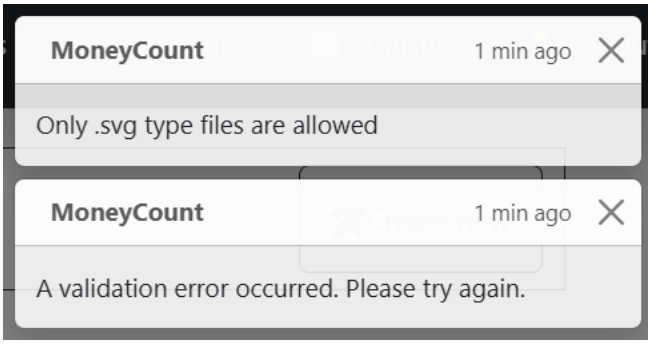

Рисунок 3.21 – Валідація під час створення категорії

Створимо основні категорії, які допоможуть краще відстежувати бюджет: продукти, хобі, їжа та напої, транспорт, відпустка та інше (рис. 3.22). Після додавання категорій до системи, вони автоматично будуть доступні користувачеві під час створення транзакції. (рис. 3.23)

Після створення категорій можна переходити до додавання фінансових транзакцій. Транзакція – це запис про витрату або надходження коштів, який можна категоризувати за однією категорією.

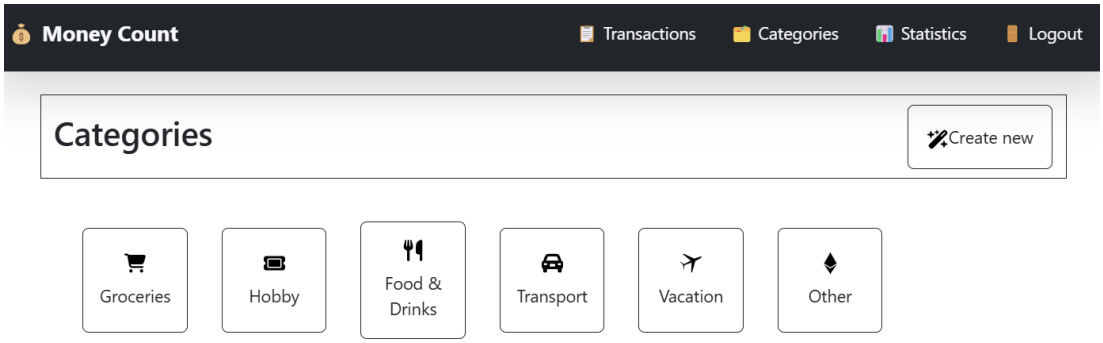

Рисунок 3.22 – Перелік створених категорій

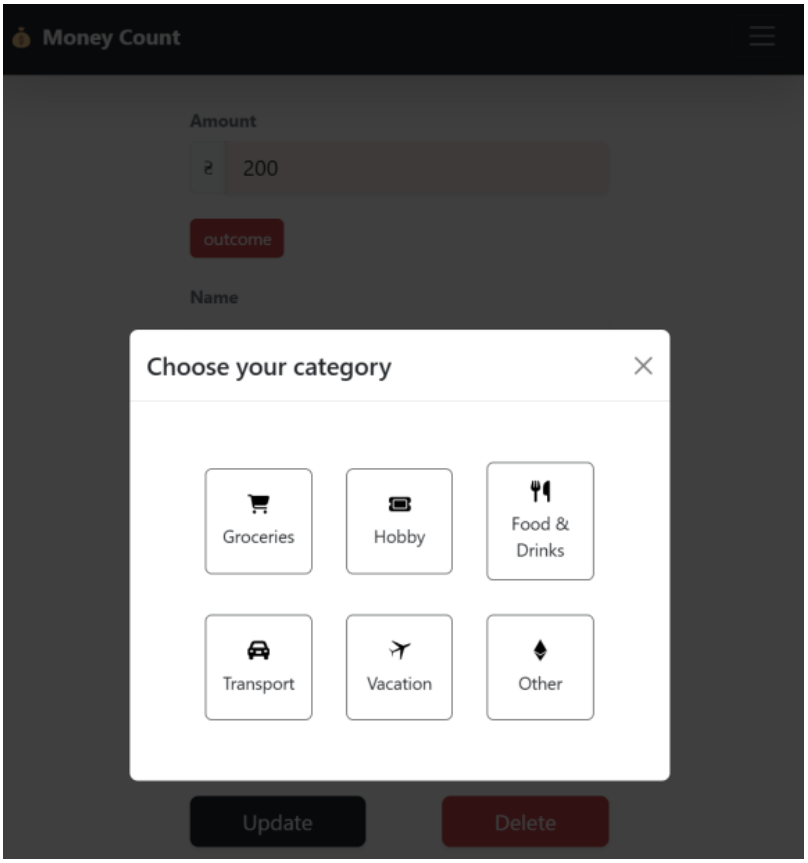

Рисунок 3.23 – Створені категорії у вікні додавання транзакції

Щоб створити нову транзакцію, користувач має: відкрити форму створення нової транзакції, вказати суму, обрати категорію, додати короткий опис (за бажанням), обрати дату (за замовчуванням сьогодні) (рис. 3.24). Якщо хоча б одне з обов’язкових полів не заповнено, система повідомить про це відповідним повідомленням (рис. 3.25).

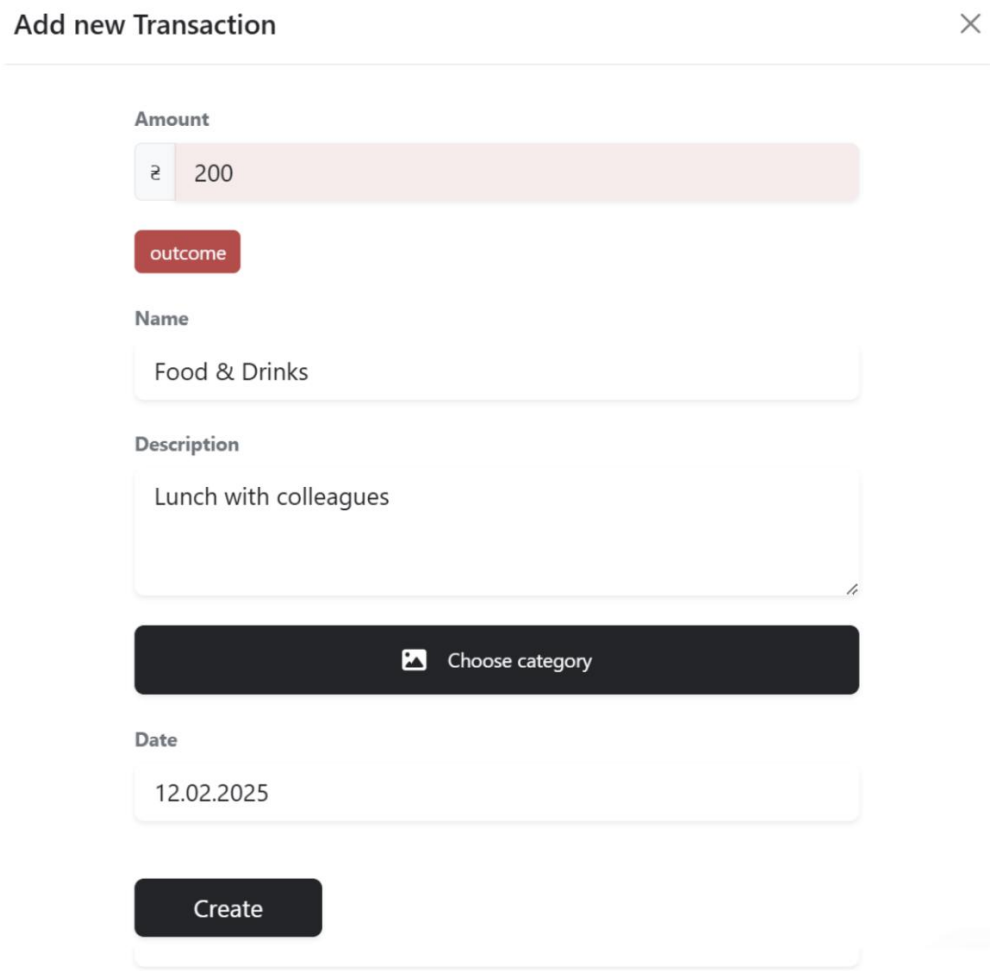

Рисунок 3.24 – Приклад створення транзакції (витрата)

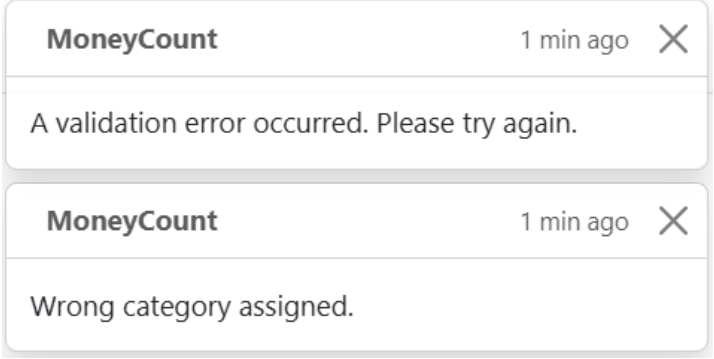

Рисунок 3.25 – Валідація під час невдалої спроби створення транзакції

Після створення транзакції вона з’являється у списку всіх транзакції, як зображено на рисунку 3.26.

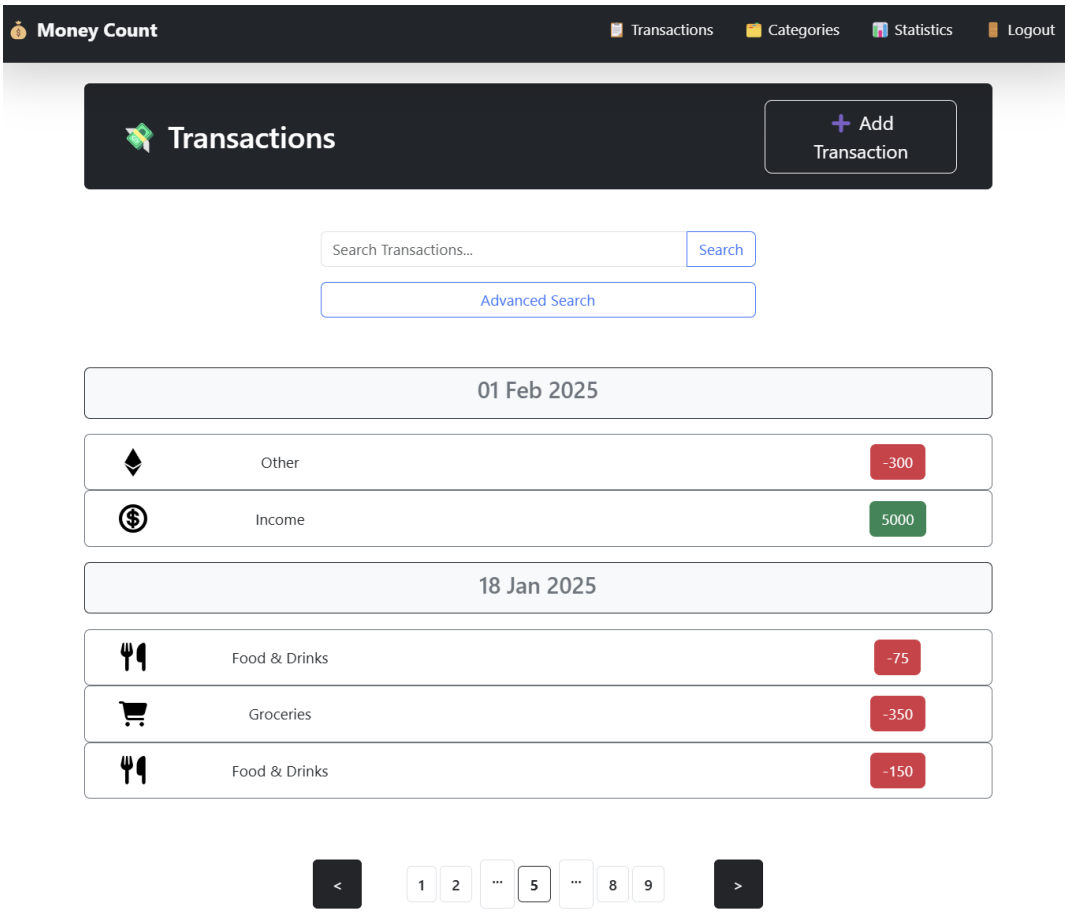

Рисунок 3.26 – Перелік створених транзакцій

У вебзастосунку передбачена сторінка Statistics, яка дозволяє користувачам детально аналізувати свої фінанси. Основними можливостями сторінки є:

– аналіз витрат – обравши період, за який необхідно переглянути витрати, система надасть детальну інформацію про фінансові операції проведені у цей проміжок часу;

– персональні рекомендації – після вибору періоду витрат штучний інтелект проаналізує дані та запропонує три короткі поради щодо покращення фінансового стану користувача;

– графічна аналітика – користувачу доступні три діаграми для подальшого аналізу особистого фінансового стану: діаграма витрат за обраний

період (рис. 3.27), динаміка доходів за останні 6 місяців (рис. 3.28), порівняння доходів і витрат за останні 6 місяців (рис. 3.29).

Рисунок 3.27 – Діаграма витрат користувача

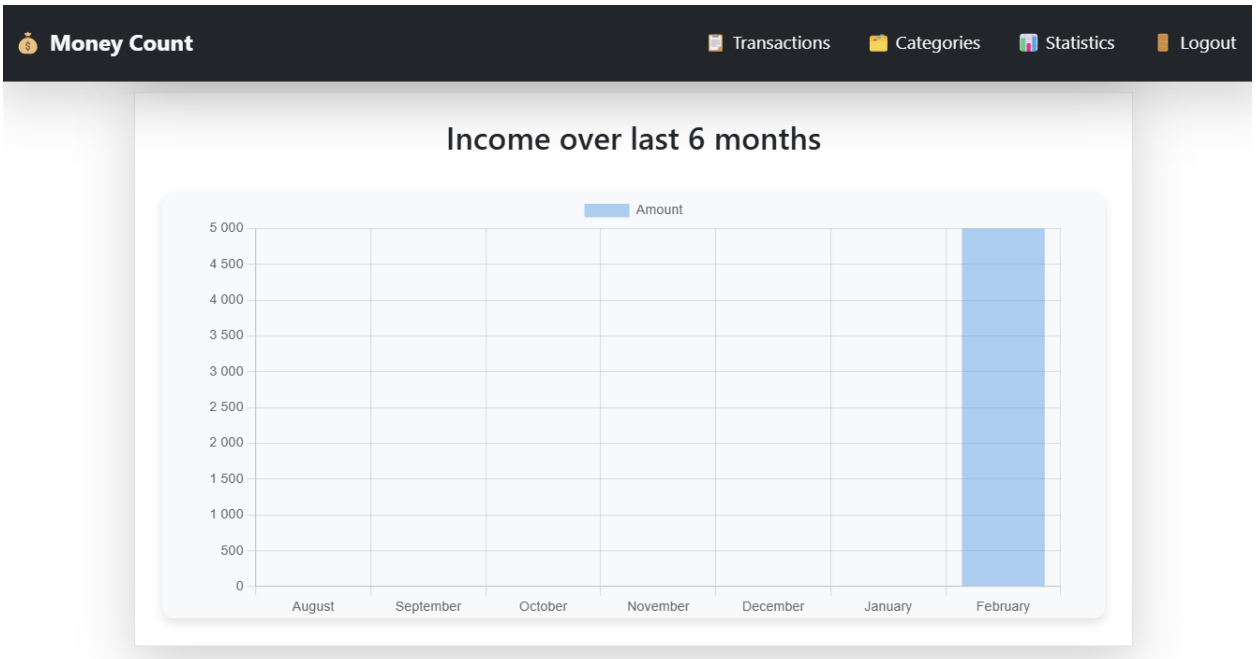

Рисунок 3.28 – Діаграма доходів за останні 6 місяців

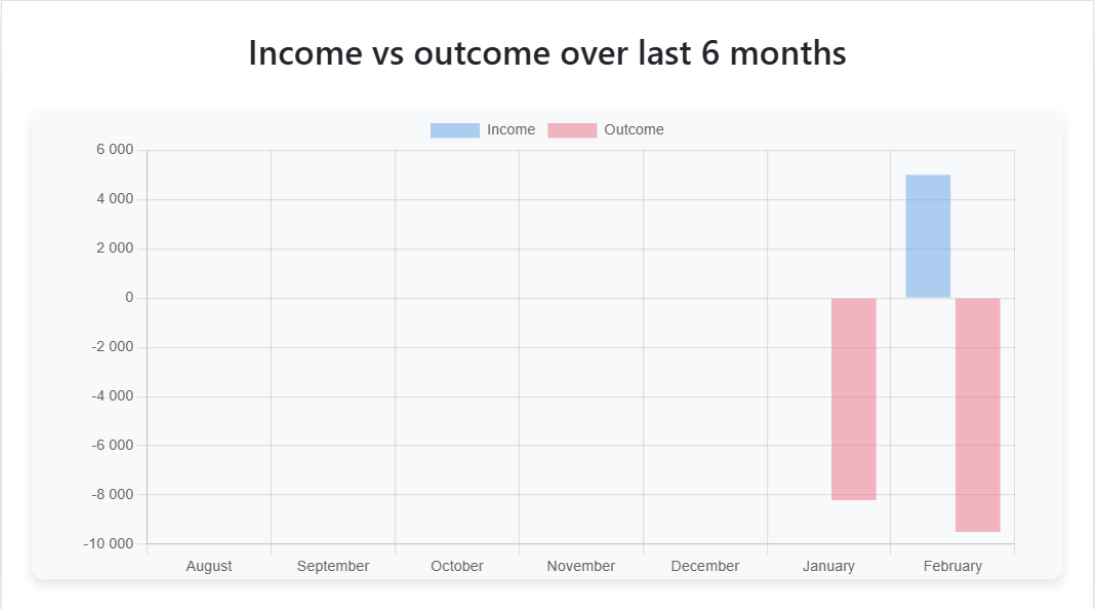

Рисунок 3.29 – Діаграма порівняння доходів і витрат за останні 6 місяців

Отже, можна зробити висновки з вищенаведеного тестування вебзастосунку, що система коректно обробляє помилки, такі як введення некоректних даних під час реєстрації чи створення категорій без назви. Інтерфейс інтуїтивно зрозумілий, що полегшує взаємодію з застосунком для нових користувачів.

Загалом, застосунок успішно виконує свою функцію, забезпечуючи зручний і ефективний інструмент для управління фінансами. Подальше вдосконалення може включати розширення можливостей аналітики, автоматизація імпортування даних з банківських рахунків, розширення локалізації, додання мультивалютності та покращення персоналізованих рекомендацій для користувачів.

ВИСНОВКИ

У рамках кваліфікаційної роботи було розроблено вебзастосунок для керування особистими фінансами. Було проведено аналіз існуючих методів та застосунків для ведення обліку фінансів та візуалізації витрат, що дозволило визначити ключові вимоги до функціональності застосунку.

Метою кваліфікаційної роботи було створення інтуїтивно зрозумілого та функціонального вебзастосунку, який допомагав би користувачам аналізувати свої фінанси, виявляти основні статті витрат та ефективніше планувати бюджет.

Під час розробки були використані сучасні технології: React і Node.js для клієнтської частини та C#, .NET для серверної частини. Це забезпечило гнучкість, масштабованість та високу продуктивність застосунку. Для роботи з даними була реалізована система категоризації витрат і доходів, що дозволяє користувачам швидко орієнтуватися у своїх фінансах та приймати зважені фінансові рішення.

Застосунок може бути використаний широким колом користувачів, які бажають ефективно керувати своїми фінансами. Завдяки інтерактивному інтерфейсу та аналітичним інструментам він допомагає оптимізувати витрати та досягати фінансових цілей.

У результаті, створений вебзастосунок відповідає поставленим вимогам та може бути основою для подальшого розширення функціональності, зокрема інтеграції з банківськими сервісами або додавання нових інструментів для фінансового планування.

Результати роботи апробовано у вигляді тези доповіді під час Міжнародного молодіжного форуму «РАДІОЕЛЕКТРОНІКА І МОЛОДЬ У XXI СТОЛІТТІ» [32].

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Regan, G. (2021). The internet revolution. In A brief history of computing (pp. 237-255). Cham: Springer International Publishing.

2. Home of the first website. URL: https://info.cern.ch (дата звернення 25.01.2025).

3. Best Web Design Trends. URL: https://www.awwwards.com (дата звернення 25.01.2025).

4. Тітов С. В., & Тітова О. В. (2014). Інформаційно-освітнє середовище навчального закладу: розвиток засобів і способів комунікаційної й інформаційної взаємодії. Вісник Харківської державної академії культури, (43), 144-150.

5. The Rising Era of Web Application Development With PWAs. URL: https://dzone.com/articles/the-rising-era-of-web-application-development-with (дата звернення 25.01.2025).

6. Rao, P. R., Chaurasia, A. K., & Singh, S. P. (2023). Modern web design: Utilizing HTML5, CSS3, and responsive techniques. The International Journal of Research and Innovation in Dynamics of Engineering, 1(8), a1-a18.

7. YNAB. URL: https://www.ynab.com (дата звернення 26.01.2025).

8. GoodBudget. URL: https://goodbudget.com (дата звернення 26.01.2025).

9. PocketGuard: Budgeting App & Finance Planner. URL: https://pocketguard.com (дата звернення 02.02.2025).

10. Стадії циклу розробки ПЗ. URL: https://qalight.ua/baza-znaniy/stadiyi-tsiklu-rozrobki-pz (дата звернення 01.02.2025).

11. Супруненко, О. О. (2012). Аналіз вимог до програмного забезпечення: навч. посібник. *Черкаси*: *Черкаський національний університет ім. Богдана Хмельницького*, 24 с.

12. Ali, S., Alauldeen, R., & Ruaa, A. (2020). What is Client-Server System: Architecture, Issues and Challenge of Client-Server System. HBRP Publication, 2(1), 1-6.

13. Nyabuto, M. G. M., Mony, M. V., & Mbugua, S. (2023). Architectural Review of Client-Server Models.

14. Розуміння Клієнт-Серверної Архітектури на прикладах. URL: https://dou.ua/forums/topic/44636 (дата звернення 14.01.2025).

15. 5. Banks, A., & Porcello, E. (2020). *Learning React: modern patterns for developing React apps.* O'Reilly Media.

16. Gackenheimer, C. (2015). Introduction to React. Apress.

17. Learn React. URL: https://react.dev/learn (дата звернення 21.01.2025).

18. Kinnunen, J. (2023). Designing a Node. js full stack web application.

19. Node.js NPM. URL: https://www.geeksforgeeks.org/node-js-npm-node-package-manager (дата звернення 20.01.2025).

20. Get started with Bootstrap. URL: https://getbootstrap.com/docs (дата звернення 15.01.2025).

21. Gowda, P., & Gowda, A. N. (2024). Best Practices in REST API Design for Enhanced Scalability and Security. *Journal of Artificial Intelligence, Machine Learning and Data Science*, *2*(1), 827-830.

22. What's .NET. URL: https://dotnet.microsoft.com/en-us/learn/dotnet/what-is-dotnet (дата звернення 06.02.2025).

23. What's new in .NET 8. URL: https://learn.microsoft.com/en-us/dotnet/core/whats-new/dotnet-8/overview (дата звернення 06.02.2025).

24. Myllyaho Forsberg, M. (2022). An evaluation of. NET Object-Relational Mappers in relational databases: Entity Framework core and Dapper.

25. Data Models. URL: https://dssc.eu/space/bv15e/766067670/Data+Models (дата звернення 07.02.2025).

26. Gemini Developer API. URL: https://ai.google.dev/gemini-api/docs (дата звернення 07.02.2025).

27. Visual Studio Build and Key Features. URL: https://www.csharp.com/article/visual-studio-builds-and-key-features (дата звернення 09.02.2025).

28. Kahlert, T., & Giza, K. (2016). Visual Studio Code Tips & Tricks Vol. 1. Microsoft Deutschland GmbH.

29. Your Ultimate Guide To Visual Studio vs Visual Studio Code. URL: https://www.turing.com/kb/ultimate-guide-visual-studio-vs-visual-studio-code (дата звернення 09.02.2025).

30. Selvi, K., Rekha, M., & Karthiga, R. (2024). CRUD Application Using ReactJS Hooks. EAI Endorsed Transactions on Internet of Things, 10.

31. Component. URL: https://react.dev/reference/react/Component (дата звернення 15.01.2025).

32. Краснянська, В. (2025) Аналіз застосунків для ведення обліку персональних фінансів. Радіоелектроніка і молодь у XXI столітті: тези доповідей 29-го Міжнародного молодіжного форуму (Харків, 16–19 квітня 2025 р.). Харків: ХНУРЕ, 2025. Т.7. С. 68-.69.