МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ДЕРЖАВНИЙ УНІВЕРСИТЕТ ЕКОНОМІКИ І ТЕХНОЛОГІЙ

ННІ	економіки та бізнес-освіти
Кафедра	економіки та цифрового бізнесу
Спеціальність	122 Комп'ютерні науки
Форма навчання	Денна

КВАЛІФІКАЦІЙНА РОБОТА БАКАЛАВРА

Коліогло Катерини Вячеславівни
(прізвище, ім'я, по батькові здобувача)

на тему **Розробка веб-застосунку для управління фінансами**
(повна назва теми)

за матеріалами _____

(повна назва бази дослідження)

науковий керівник _____ _____ Шокотько Л.М.
(наук. ступінь, вчене звання) *(підпис)* *(прізвище, ініціали)*

Робота допущена до захисту в ЕК
Протокол засідання кафедри
від 09.06.2025р. № 12
Завідувач кафедри _____
(підпис)
к.е.н., доцент Радько В.М.
наук. ступень, вчене звання прізвище, ініціали

Кривий Ріг – 2025

ЗАТВЕРДЖЕНО
Наказ Міністерства освіти і науки, молоді та спорту України
29 березня 2012 року № 384

Форма № Н-9.01

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ДЕРЖАВНИЙ УНІВЕРСИТЕТ ЕКОНОМІКИ І ТЕХНОЛОГІЙ
НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ ЕКОНОМІКИ ТА БІЗНЕС-ОСВІТИ
(повне найменування вищого навчального закладу)

Кафедра економіки та цифрового бізнесу
Освітній ступінь бакалавр
Спеціальність Комп'ютерні науки

ЗАТВЕРДЖУЮ
Завідувач кафедри _____В.М. Радько

"07" квітня 2025 року

З А В Д А Н Н Я
НА КВАЛІФІКАЦІЙНУ РОБОТУ БАКАЛАВРА ЗДОБУВАЧУ
Коліогло Катерині Вячеславівні

1. Тема роботи Розробка веб-застосунку для управління фінансами

науковий керівник роботи Шокотько Людмила Миколаївна
затверджені наказом вищого навчального закладу від «04» квітня 2025 р. № 224-ст

2. Строк подання здобувачем роботи 31.05.2025р.

3. Зміст кваліфікаційної роботи бакалавра, об'єкт, предмет та мета дослідження:

Розділ 1 ТЕОРЕТИЧНІ АСПЕКТИ РОЗРОБКИ ВЕБ-ДОДАТКІВ ФІНАНСОВОГО ОБЛІКУ
Розділ 2 ПРОЄКТУВАННЯ СТРУКТУРИ ТА БАЗИ ДАНИХ ВЕБ-СИСТЕМИ RIKONOMY
Розділ 3 РОЗРОБКА ВЕБ-СИСТЕМИ ОБЛІКУ ОСОБИСТИХ ФІНАНСІВ RIKONOMY

Об'єкт дослідження – процеси проєктування та розробки веб-застосунку для обліку та управління особистими фінансами

Предмет дослідження – архітектурні рішення, інформаційні потоки та засоби реалізації веб-застосунків для обліку фінансів на прикладі системи Rikonomy.

Мета кваліфікаційної роботи бакалавра – розробити веб-застосунок для обліку особистих фінансів із сучасною архітектурою, інтерактивним інтерфейсом, багатомовністю та підвищеним рівнем безпеки.

4. Дата видачі завдання 04.04.2025р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів кваліфікаційної роботи бакалавра	Строк виконання етапів роботи	Відмітка керівника про виконання етапів (дата, підпис)
1	Підготовка розділу 1	до 28.04.2025р.	25.04.2025
2	Підготовка розділу 2	до 16.05.2025р.	15.05.2025
3	Підготовка розділу 3	до 30.05.2025р.	29.05.2025
4	Реєстрація завершеної дипломної роботи	до 31.05.2025р.	30.05.2025
5	Отримання відгуку від наукового керівника	03-04.06.2025р.	04.06.2025
6	Отримання зовнішньої рецензії	05-06.06.2025р.	06.06.2025
7	Перевірка кваліфікаційної роботи на плагіат	02-09.06.2025р.	04.06.2025
8	Попередній захист кваліфікаційної роботи на кафедрі	03.06.2025р.	03.06.2025
9	Допуск кафедрою кваліфікаційної роботи до захисту	09.06.2025р.	09.06.2025
10	Підготовка студента до захисту в ЕК	до 17.06.2025р.	17.06.2025

Завдання підготував науковий керівник Шокотько Л.М.

Завдання одержав здобувач _____ Коліогло К.В.
(підпис) (прізвище та ініціали)

РЕФЕРАТ

Кваліфікаційна робота бакалавра: 54 сторінки, 19 рисунків, 8 таблиць, 17 джерел, 3 додатки.

Об’єкт дослідження: процеси проєктування та розробки веб-застосунку для обліку та управління особистими фінансами.

Мета: створення клієнт-серверного веб-додатку для зручного, захищеного та гнучкого ведення фінансового обліку, з інтерактивним інтерфейсом, багатомовністю та підтримкою різних валют.

У результаті дослідження було проаналізовано проблеми, що виникають при обліку особистих фінансів, досліджено сучасні технології розробки веб-систем, проєктування баз даних і підходи до побудови клієнт-серверної архітектури. Розроблено структуру бази даних MySQL, реалізовано серверну логіку авторизації на Node.js/Express, створено інтерфейс модуля авторизації на HTML/CSS/JavaScript із перевіркою складності пароля й адаптивною верстkою. Особливу увагу приділено безпеці: застосовано хешування паролів, захист персональних даних і сучасні стандарти автентифікації. Здійснено тестування модуля авторизації, сформовано рекомендації для подальшого розвитку системи.

Результати роботи можуть бути використані для створення персональних та корпоративних систем обліку фінансів, впровадження авторизаційних модулів у веб-додатки, а також як основа для подальшого розширення функціоналу у фінансових застосунках.

Ключові слова: авторизація, адаптивний інтерфейс, база даних, веб-застосунок, інформаційна безпека, клієнт-серверна архітектура, облік фінансів, хешування паролів, JavaScript, Express, MySQL, Node.js, REST API

ЗМІСТ

ПЕРЕЛІК УМОВНИХ СКОРОЧЕНЬ

API – Application Programming Interface.

СКБД – Система Керування Базами Даних

JS – JavaScript.

JSON – JavaScript Object Notation.

JWT – JSON Web Token.

SQL – Structured Query Language.

OODBMS – Object-Oriented Database Management System

RDBMS – Relational Database Management System

REST – Representational State Transfer.

HTML – HyperText Markup Language.

CSS – Cascading Style Sheets.

UI – User Interface.

UX – User Experience.

ВСТУП

Інтенсивна діджиталізація фінансового сектору зумовила появу великої кількості веб-застосунків для управління особистими коштами. Проте більшість із них зберігають дані ізольовано, опираються на різні стандарти безпеки та пропонують обмежений функціонал локалізації й конвертації валют. Провідні дослідницькі центри фінтех-галузі (NFC, FinTechLab, European Payments Council) концентруються нині на розробці відкритих API, багатовалютних сервісів і механізмів захисту даних, що відповідають вимогам GDPR та PSD2. Попри помітні результати, питання інтеграції розрізнених джерел транзакцій, забезпечення багатомовної підтримки і збереження високої продуктивності залишаються актуальними.

Актуальність роботи обумовлена потребою створення універсальної клієнт-серверної платформи, здатної об’єднати дані з різних банківських та платіжних систем, забезпечити їх надійне зберігання, підтримати автоматичний перерахунок валют і надати користувачеві локалізований інтерфейс. Світові тенденції демонструють перехід до open-banking-підходу, використання AI-алгоритмів для прогнозу витрат і підсилення захисту даних за рахунок багаторівневих протоколів автентифікації.

Мета дослідження полягає у розробленні та експериментальній перевірці клієнт-серверної веб-системи Rikonomy, що поєднує багатомовний інтерфейс, тематичне оформлення та мультивалютну підтримку з транзакційною обробкою даних у реальному часі і відповідає сучасним вимогам безпеки й масштабованості.

Об’єктом дослідження є веб-застосунки для обліку особистих фінансів, а предметом – методи та архітектурні рішення, що забезпечують їхню безпечну, продуктивну й адаптивну роботу.

Для досягнення поставленої мети визначено такі завдання:

– проаналізувати сучасний стан ринку персональних фінансових сервісів та їхніх технологічних обмежень;

- сформулювати вимоги користувачів до функціоналу, безпеки й локалізації системи;
- обґрунтувати вибір архітектурної моделі, СКБД та стеку технологій;
- спроєктувати структуру бази даних і RESTful-API;
- реалізувати прототип системи з підтримкою багатомовності, тем оформлення і валютної конверсії;
- провести навантажувальне й безпекове тестування, оцінити продуктивність і масштабованість;
- проаналізувати результати й визначити напрями подальшого розвитку.

У дослідженні використано методи системного та порівняльного аналізу, UML-моделювання, проєктування баз даних, експериментальне навантажувальне тестування, а також методи статистичного аналізу для обробки отриманих результатів. Інформаційну базу становлять наукові публікації IEEE, ACM, матеріали конференцій IFIN, офіційна документація PSD2, GDPR та специфікації відкритих API провідних банків і платіжних провайдерів.

Запропоноване рішення має практичну цінність завдяки можливості розгортання як автономного сервісу або інтеграції окремих модулів (наприклад, валютної конвертації чи системи бюджетування) у наявні фінтех-проєкти. Наукова новизна полягає в комплексній інтеграції механізмів локалізації, персоналізації й мультивалютності з транзакційною обробкою фінансових даних за клієнт-серверної архітектури, що забезпечує дотримання сучасних стандартів безпеки та можливість горизонтального масштабування.

РОЗДІЛ 1
АНАЛІЗ СУЧАСНИХ ІНСТРУМЕНТІВ ДЛЯ СТВОРЕННЯ ВЕБ-ДОДАТКІВ ФІНАНСОВОГО ОБЛІКУ

1.1 Огляд сучасних технологій веб-розробки

Сучасна веб-розробка – це одна з найбільш швидкозмінних галузей у світі інформаційних технологій. Вимоги користувачів, які тільки зростають, до функціоналу, швидкості та безпечності веб-додатків стимулюють появу нових інструментів, технік та підходів до їх розробки. У цьому ключі важливо зосередитися на кількох основних аспектах, які визначають напрямок розвитку веб-розробки.

Однією з основних тенденцій є активне використання фреймворків і бібліотек для спрощення розробки клієнтської частини додатків. Популярні бібліотеки, такі як React.js, Angular і Vue.js, забезпечують створення динамічних інтерфейсів з використанням компонентного підходу. Це дозволяє розробникам повторно використовувати код, що підвищує ефективність роботи та якість кінцевого продукту. Для серверної частини поширеними є Node.js, Django та Laravel, які надають широкий спектр інструментів для роботи з базами даних, API і безпекою[1].

Розвиток хмарних технологій став ще однією важливою складовою веб-розробки. Завдяки сервісам, таким як Amazon Web Services (AWS), Google Cloud Platform і Microsoft Azure, розробники мають можливість створювати масштабовані додатки, які автоматично адаптуються до навантаження. Хмарні платформи також забезпечують надійність та безпеку, що є критично важливим для додатків, які працюють з конфіденційними даними.

Ще однією значною тенденцією є впровадження Progressive Web Apps (PWA) [2]. Ці веб-додатки поєднують функціональність веб-сайтів і мобільних додатків, забезпечуючи користувачам доступ до сервісу навіть без постійного

інтернет-з’єднання. Використання PWA дозволяє суттєво покращити швидкість завантаження, а також підвищити зручність і інтерактивність додатків.

У нинішніх реаліях однією з головних вимог до веб-застосунків стала їхня стійкість до несанкціонованого втручання. Передавання даних у зашифрованому вигляді через HTTPS поверх протоколу TLS розглядається як базовий стандарт безпеки[3].Доповненням до цього виступають заходи протидії найпоширенішим атакам, зокрема XSS, SQL-ін’єкціям і CSRF: застосовують заголовок Content Security Policy, параметризовані запити до бази та одноразові анти-CSRF токени. На всіх етапах життєвого циклу ПЗ код регулярно сканують автоматизовані засоби, серед яких OWASP ZAP і Burp Suite; вони дають змогу оперативно виявляти вразливі ділянки й усувати їх до виходу продукту у продакшн[4].

Окремо слід відзначити розвиток методів тестування веб-додатків. Інструменти автоматизованого тестування, такі як Selenium, Cypress і Jest, дозволяють знизити ризик появи помилок, що сприяє покращенню якості продукту. Важливою є і практика Continuous Integration/Continuous Deployment (CI/CD), яка автоматизує процеси тестування та розгортання додатків[5].

Для кращого розуміння сучасних технологій веб-розробки нижче наведено таблицю 1.1, що узагальнює популярні інструменти та їх основні характеристики.

Таблиця 1.1

Популярні технології веб-розробки та їх основні характеристики

Технологія	Тип	Основні переваги	Недоліки
1	2	3	4
HTML5	Мова розмітки	Простота використання, широке підтримання всіма сучасними браузерами, адаптивність для різних платформ, підтримка мультимедіа без додаткових плагінів.	Обмеження функціоналу без поєднання з CSS та JavaScript.

Продовження табл. 1.1.

1	2	3	4
CSS3	Мова стилізації	Потужні інструменти для створення адаптивних та естетичних інтерфейсів, підтримка анімації та медіа-запитів, сумісність із HTML5.	Складність організації стилів для великих проєктів без препроцесорів (наприклад, SASS чи LESS).
JavaScript	Мова програмування	Інтерактивність, багатофункціональність, підтримка обробки подій, можливість роботи як на клієнтській, так і на серверній стороні (через Node.js).	Можливість створення некоректного коду через динамічну типізацію, складність при роботі з великими проєктами.
React.js	Фреймворк JS	Висока продуктивність завдяки віртуальному DOM, можливість повторного використання компонентів, підтримка спільнотою, зручність при створенні SPA (односторінкових додатків).	Потребує базових знань JavaScript, складність початкового налаштування для новачків.
Angular	Фреймворк JS	Потужний функціонал для створення складних додатків, підтримка Google, структура «все в одному» для роботи з компонентами, сервісами та шаблонами.	Великий розмір бібліотеки, крута крива навчання, можлива надмірна складність для невеликих проєктів.
Vue.js	Фреймворк JS	Легка у засвоєнні, гнучка у використанні, зручна інтеграція у наявні проєкти, підтримка компонентного підходу та реактивності.	Менша популярність у порівнянні з React чи Angular, можлива нестабільність при використанні зовнішніх плагінів.

Продовження табл. 1.1.

1	2	3	4
Node.js	Платформа JS	Можливість серверної розробки на JavaScript, асинхронна обробка запитів, масштабованість, активна підтримка npm-пакетів.	Складність у відлагодженні через асинхронність, залежність від продуктивності JavaScript.
Python (Django)	Фреймворк Python	Висока швидкість розробки, потужний інструмент для роботи з базами даних, безпечність, модульність, можливість швидкої інтеграції сторонніх бібліотек.	Великий розмір фреймворку, менша популярність для фронтенд-розробки.
PHP (Laravel)	Фреймворк PHP	Інтуїтивна структура, зручний інструментарій для роботи з базами даних, підтримка MVC-архітектури, активна спільнота.	Обмежена продуктивність порівняно з іншими технологіями, потреба у конфігурації для оптимізації продуктивності.
MySQL	Система баз даних	Популярність, гнучкість у використанні, відкрите ПЗ, потужні інструменти для управління великими обсягами даних.	Менш підходить для горизонтального масштабування у порівнянні з деякими іншими СУБД.
PostgreSQL	Система баз даних	Надійність, підтримка складних запитів, висока масштабованість, активна спільнота, зручність роботи з JSON-даними.	Вища складність конфігурації у порівнянні з MySQL.

(Розроблено автором)

Таким чином, сучасні технології веб-розробки створюють широкі можливості для розробників, дозволяючи реалізувати ефективні, безпечні та масштабовані рішення. Вибір інструментів залежить від потреб конкретного проєкту, проте загальним трендом залишається інтеграція інноваційних методів у всі етапи розробки.

1.2 Аналіз проблем та викликів в обліку особистих фінансів

Особисті фінанси займають ключове місце у повсякденному житті кожної людини, оскільки раціональне управління грошовими ресурсами дозволяє забезпечувати стабільність та досягати фінансових цілей. Проте ефективний облік особистих витрат залишається складним завданням через низку факторів, які впливають на точність, своєчасність та зручність ведення такого обліку.

Однією з головних проблем є недостатня прозорість фінансових потоків. Сучасний спосіб життя передбачає використання кількох інструментів для здійснення платежів – готівка, банківські картки, мобільні платіжні сервіси, електронні гаманці. Така багатоканальність значно ускладнює контроль за витратами, оскільки користувачу необхідно відстежувати транзакції з різних джерел. Дослідження вказують, що використання безготівкових платежів може сприяти втраті відчуття контролю над витратами, що створює додаткові труднощі в особистому обліку [6].

Часто виникає проблема нестачі часу для ведення фінансового обліку. Щоб фіксувати кожну операцію вручну, потрібно докласти чимало зусиль і робити це регулярно, а це далеко не завжди зручно. Через це у записах можуть з’являтися пропуски або неточності. Навіть якщо користувач звертається до автоматизованих сервісів, для їхнього налаштування все одно потрібні певні знання і час, адже кожна система потребує індивідуального підходу.

Складність аналізу накопичених фінансових даних також залишається актуальною проблемою. Відсутність гнучких інструментів візуалізації витрат та формування звітності обмежує можливості користувачів у прийнятті обґрунтованих рішень щодо розподілу бюджету та планування витрат. За результатами досліджень, інтеграція гейміфікаційних і візуальних елементів може підвищити ефективність сприйняття фінансової інформації, однак така функціональність все ще не є поширеною серед масових продуктів [7].

Окрему увагу слід приділити питанню конфіденційності даних. Багато користувачів побоюються використовувати сторонні сервіси через ризик витоку

або несанкціонованого використання фінансової інформації. Поширеною практикою є зберігання персональних даних у хмарних середовищах, що не завжди відповідає вимогам щодо захисту чутливої інформації. За даними фінансових регуляторів, навіть великі додатки не завжди дотримуються стандартів шифрування на етапі передачі або зберігання даних [8].

Останнім часом дедалі більше користувачів звертають увагу на веб-додатки, які допомагають стежити за особистими фінансами. Такі інструменти дають змогу зручно вносити витрати, отримувати звіти й налаштовувати категорії за власними потребами. Автоматизовані сервіси також дозволяють надсилати нагадування, що спрощує регулярний облік і контроль за рухом коштів. Але навіть сучасні застосунки не позбавлені певних недоліків. Наприклад, часто потрібне постійне підключення до інтернету, інтерфейс не завжди легко підлаштувати під свої потреби, а підтримка різних мов чи регіональних налаштувань буває обмеженою.

Серед найвідоміших міжнародних застосунків для обліку особистих фінансів слід відзначити Mint (Intuit), YNAB (You Need A Budget), Money Lover, PocketGuard та Spendee. Застосунок Mint пропонує автоматичне підключення до банківських рахунків, зручну категоризацію витрат, нагадування про платежі, а також інтеграцію з кредитними рейтингами. Однак, користувачі з інших країн, окрім США та Канади, часто стикаються з обмеженнями доступу до локальних банків та підтримки національних валют .

YNAB акцентує увагу на плануванні бюджету, надаючи користувачам можливість ставити фінансові цілі та контролювати прогрес за допомогою детальних звітів. Проте, для багатьох новачків цей інтерфейс здається складним, а основний функціонал є платним після 34-денної безкоштовної версії.

Money Lover і Spendee виділяються підтримкою мультивалютності та широкою географією користувачів, а також можливістю створювати загальні бюджети для сімей або груп. Водночас, вільний функціонал у цих застосунках часто обмежений, і користувачам доводиться купувати преміум-плани для повного доступу до аналітики або автоматичного імпорту даних.

PocketGuard популярний у США, пропонує простий у використанні інтерфейс, автоматичне відстеження підписок і облік залишку заощаджених коштів, однак має англомовний інтерфейс і не підтримує багато національних банків поза США.

Незважаючи на широкі можливості, більшість згаданих рішень мають спільні недоліки: обмеження у підтримці локальних банків, мовна адаптація, відсутність інтеграції з державними чи локальними сервісами, а також ризики, пов’язані із конфіденційністю фінансової інформації.

У таблиці 1.2 наведено основні проблеми, які зустрічаються під час ведення обліку особистих витрат, і можливі наслідки таких складнощів.

Таблиця 1.2.

Основні проблеми в обліку особистих витрат та їх наслідки

Проблема	Причина виникнення	Потенційні наслідки
Багатоканальність фінансових потоків	Використання кількох платіжних засобів (готівка, картки, е-гаманці)	Ускладнене узагальнення витрат, зниження точності обліку
Обмеженість часу	Високий темп життя, відсутність мотивації до систематичного обліку	Нерегулярність ведення даних, пропущені транзакції
Складність аналізу	Відсутність зручних звітів і візуалізацій у додатках	Зменшення ефективності планування бюджету
Ризики конфіденційності	Збереження персональних даних у хмарних сервісах без належного захисту	Недовіра до сервісів, відмова від використання
Низька адаптація до локальних умов	Відсутність підтримки місцевих валют, мов, категорій витрат	Зниження зручності використання, обмеження функціональності
Відсутність інтеграції	Неможливість синхронізації з банками, бухгалтерськими системами	Ручне введення даних, підвищення ризику помилок

(Розроблено автором)

Таким чином, аналіз існуючих рішень у сфері персонального фінансового обліку свідчить про те, що вони не завжди відповідають запитам користувачів. Недостатній захист конфіденційних даних, відсутність належної адаптації до регіональних умов, складність інтеграції з іншими платформами – усе це формує потребу у створенні нових або вдосконаленні наявних інструментів.

Отже, ключовими проблемами у сфері обліку особистих фінансів залишаються: складність організації та централізації даних, часові витрати на введення та перевірку інформації, недостатній рівень конфіденційності, а також обмежена функціональність наявних веб-додатків. Подолання цих викликів можливе шляхом впровадження інноваційних рішень, що дозволяють користувачам ефективно керувати своїми фінансами в безпечному та адаптивному цифровому середовищі.

1.3 Аналіз інформаційних потоків та постановка задачі

У сучасних клієнт-серверних веб-системах критично важливо правильно налаштувати потік інформації між основними компонентами. Від того, як саме відбувається обмін даними між клієнтською та серверною частинами залежить швидкість роботи додатку, стабільність усього сервісу та комфорт користувача. Для систем, що мають справу з фінансовими чи персональними даними, особливо важливо дотримуватись чітких правил передачі даних і стандартів безпеки.

Аналізуючи типові інформаційні потоки у таких системах, можна виділити кілька ключових сценаріїв. Запит від користувача – наприклад, авторизація, додавання витрати, перегляд статистики чи зміна налаштувань – завжди проходить від клієнта до сервера через захищений канал, а потім повертається відповідь із результатом операції. На кожному з цих етапів важливо дотримуватися послідовності дій: перевірка автентичності, обробка даних, формування відповіді, логування операцій.

Якщо розглядати майбутній додаток для обліку фінансів, інформаційний потік виглядатиме наступним чином. Перший крок – користувач заходить на сайт через браузер, відбувається завантаження інтерфейсу і відправка запиту на авторизацію. Сервер перевіряє введені дані, визначає права доступу та повертає підтвердження. Далі користувач може додавати нові витрати, створювати або змінювати категорії, переглядати аналітику чи формувати звіти. Кожна така дія – це окремий запит до серверної частини, де дані перевіряються й потрапляють у базу. Для виведення статистики чи аналізу бюджету сервер обробляє запит, виконує розрахунки та повертає готовий результат клієнту.

Варто враховувати й додаткові інформаційні потоки. У майбутньому, наприклад, може виникнути потреба інтеграції із зовнішніми API для автоматичного підвантаження курсів валют або синхронізації з банківськими сервісами. Такі операції також мають бути чітко прописані у логіці взаємодії: дані зберігаються централізовано, зовнішній запит проходить перевірку, результат валідується й додається до системи.

Особлива увага при моделюванні інформаційних потоків у фінансових системах приділяється безпеці та захисту персональних даних. Канали обміну повинні бути зашифрованими, а усі операції із записом або зміною інформації – підтверджуватись перевіркою прав доступу. Також важливо організувати резервне копіювання даних та журналювання дій користувача для подальшого аудиту.

У підсумку, для створення веб-системи фінансового обліку постає низка задач:

- чітко структурувати основні та допоміжні інформаційні потоки між клієнтською і серверною частинами;
- визначити сценарії взаємодії для типових операцій користувача (реєстрація, авторизація, облік витрат, формування звітів, зміна налаштувань);
- передбачити можливість підключення зовнішніх сервісів і розширення функціоналу у майбутньому;

– закласти основи для захисту та контролю доступу на кожному етапі обміну даними.

Усе це має забезпечити стабільну роботу системи, захист даних та простоту користування навіть при зростанні кількості користувачів чи обсягу інформації. Саме такі принципи і стають основою для подальшого проєктування архітектури та розробки майбутнього застосунку.

Висновки до розділу 1

У цьому розділі було розглянуто основні проблеми, які виникають при обліку особистих фінансів. Виявлено, що користувачі стикаються з труднощами через багатоканальність платежів, нестачу часу для регулярного обліку, незручність аналізу даних та побоювання щодо конфіденційності. Багато сучасних додатків не завжди відповідають індивідуальним потребам, мають обмежену підтримку мов, не всі інтегруються із зовнішніми сервісами.

Також у розділі розглянуто сучасні технології веб-розробки, які дають змогу створювати зручні та безпечні рішення. Описано основні інструменти для роботи з клієнтською і серверною частиною, відзначено переваги використання хмарних сервісів, сучасних підходів до розгортання додатків і засобів для тестування.

Отримані результати дають чітке розуміння стану галузі й окреслюють напрями для подальшої розробки власного рішення. Зібрана у першому розділі інформація дозволила визначити ключові вимоги для подальших етапів проєкту.

РОЗДІЛ 2
ПРОЄКТУВАННЯ СТРУКТУРИ ТА БАЗИ ДАНИХ ВЕБ-СИСТЕМИ RIKONOMY

2.1 Вибір архітектури клієнт-серверної взаємодії

У повсякденній практиці розробки програмних продуктів для обліку фінансів часто виникає необхідність уважно продумати архітектурну основу майбутньої системи. Не можна сказати, що це просте рішення, адже воно впливає на багато аспектів – від того, як зручно користувачам буде працювати з додатком, і аж до питань подальшої підтримки чи, навіть, безпеки персональних даних. У випадку створення Rikonomy це питання стало ключовим ще на етапі перших ескізів: як зробити сервіс водночас зручним, здатним до розширення і таким, щоб не виникали проблеми із збереженням чи обробкою інформації. Важливим аспектом є забезпечення можливості розширення функціоналу та інтеграції з іншими сервісами, зокрема модулями обміну валют або зовнішніми аналітичними інструментами.

Сучасна програмна інженерія виділяє декілька підходів до побудови архітектури застосунків. Протягом тривалого часу найбільш поширеним був монолітний підхід, коли уся логіка розташовувалася в одному програмному модулі. Проте з часом зростали вимоги до гнучкості та масштабування, що призвело до появи багатошарових архітектур, а згодом – до домінування клієнт-серверної моделі. Вона передбачає чіткий поділ програмного забезпечення на дві частини: клієнтську, яка відповідає за інтерфейс і взаємодію з користувачем, та серверну, що реалізує обробку запитів, зберігання і захист даних, а також бізнес-логіку. Саме таке розділення надає можливість незалежно вдосконалювати кожен компонент, спрощує масштабування, підвищує безпеку і забезпечує більш просту інтеграцію з зовнішніми API.

Визначальним фактором при виборі архітектури для Rikonomy стала потреба у постійній обробці і збереженні фінансових даних користувачів, що

потребує як високої швидкості реагування системи, так і надійного механізму контролю доступу. У класичній клієнт-серверній моделі клієнтська частина розміщується у браузері користувача, що дозволяє створити максимально інтуїтивний інтерфейс із сучасною візуалізацією, адаптивністю та додатковими опціями (темна/світла тема, багатомовність, аналітика). Водночас серверна частина перебуває на окремому захищеному сервері, на якому і відбувається перевірка даних, зберігання паролів, розрахунок статистики, облік витрат, бюджету та категорій.

У процесі аналізу було досліджено й альтернативні архітектурні рішення, зокрема однорівневі та мікросервісні моделі. Проте для завдань персонального фінансового обліку, де кожен користувач має свій ізольований набір даних, а навантаження зазвичай розподілене рівномірно, оптимальним виявилася саме класична схема “клієнт–сервер” із чітким RESTful API. Цей підхід дозволяє розмежувати права доступу, забезпечити захист через токени, а також у разі потреби інтегруватися з мобільними застосунками чи сторонніми сервісами без значної перебудови коду.

У межах означеної архітектури коректне спрямування інформаційних потоків є фундаментом стабільної й передбачуваної роботи застосунку. У Rikonomy кожна дія користувача – чи то додавання, чи перегляд витрат – породжує послідовність «запит → відповідь» між фронтендом і сервером. Клієнт формує HTTP-запит, сервер приймає його, виконує авторизацію, звертається до бази даних і повертає результат, після чого інтерфейс негайно оновлюється. Така схема водночас гарантує швидке реагування та зберігає конфіденційність персональних і фінансових даних.

Щоб система залишалася стійкою під час зростання кількості користувачів, між клієнтом і сервером можуть діяти проміжні вузли – балансувальники навантаження або проксі-сервери. Вони рівномірно розподіляють трафік, запобігають локальним перенавантаженням і підтримують безперервний сервіс.

Поряд із вибором архітектурної моделі, постає питання щодо підбору системи керування базами даних. Основні типи СКБД, які розглядаються у сучасному програмуванні – це реляційні, об’єктно-орієнтовані та NoSQL бази даних. Для реалізації Rikonomy аналізувалися всі ці категорії. Реляційні системи, такі як MySQL, PostgreSQL чи MS SQL Server, побудовані на основі табличної структури, де кожен запис у таблиці чітко визначається ключем і може бути однозначно пов’язаний з іншими таблицями через зовнішні ключі. Це забезпечує цілісність даних, можливість здійснення складних вибірок, фільтрацій, групувань, що є особливо актуальним у фінансових системах. Важливою перевагою реляційних СКБД є наявність розвинених механізмів транзакцій, гарантій цілісності даних, систем контролю доступу і резервного копіювання. Разом із тим, недоліками реляційної моделі можуть бути потреба в більш жорсткому плануванні структури бази та складність масштабування при обробці дуже великих обсягів неструктурованих даних.

Об’єктно-орієнтовані бази даних дають змогу зберігати складні структури без необхідності їх приведення до табличного вигляду. Вони є природними для деяких високоспеціалізованих задач, але поступаються реляційним системам за популярністю, зрозумілістю та кількістю готових інструментів для аналітики. У свою чергу, NoSQL системи (документоорієнтовані, графові, ключ-значення тощо) найкраще підходять для застосунків із непередбачуваною або часто змінюваною структурою даних, а також для великих, розподілених за різними дата-центрами систем. Для Rikonomy, яка має структуровану предметну область із чіткими зв’язками (користувач – витрати – категорії – бюджет), доцільнішим є використання саме реляційної бази даних.

MySQL була обрана на основі порівняння як основна СКБД. Вона відрізняється високою швидкістю виконання запитів, підтримкою стандарту SQL, доступністю інструментів резервного копіювання та відновлення, широкою підтримкою з боку спільноти, що важливо для подальшого супроводу й модернізації системи. Водночас MySQL є відкритою і безкоштовною, що знижує загальні витрати на розгортання та обслуговування системи. Слід зазначити, що

сучасні альтернативи, як PostgreSQL, також мають широкий спектр можливостей, але MySQL для задач фінансового обліку достатня і перевірена на практиці у багатьох подібних розробках.

Відповідно до вибраної архітектури, у Rikonomy клієнтська частина реалізована з використанням HTML, CSS і JavaScript. Це дозволяє забезпечити високу інтерактивність, гнучкість у налаштуванні інтерфейсу та можливість створення якісної аналітики у вигляді діаграм і графіків. Серверна частина розгорнута на Node.js із застосуванням Express як основного фреймворку для обробки HTTP-запитів. Такий вибір пояснюється простотою налаштування, асинхронністю роботи, а також наявністю великої кількості додаткових бібліотек для роботи з авторизацією, обробкою SQL-запитів, підключенням до зовнішніх сервісів тощо. Для обміну даними між клієнтом і сервером використовується формат JSON, що забезпечує уніфіковану структуру запитів і відповідей, а також зменшує навантаження на мережу.

Усі запити, які стосуються персональних даних, бюджету або фінансових операцій, захищені механізмом авторизації через JWT. Це дає змогу ідентифікувати користувача без зберігання паролів у браузері та гарантує, що кожна дія виконується саме тим користувачем, який має на це право. Додатково впроваджено хешування паролів через bcrypt, а також параметризовані SQL-запити, що знижує ризик SQL-ін’єкцій. Використання такого підходу сприяє відповідності вимогам сучасної кібербезпеки у фінансових застосунках.

Схема, подана на рис. 2.1, ілюструє логічну будову клієнт-серверної системи Rikonomy. У верхній частині зображено користувача, далі – браузер, що виступає транспортом для фронтенда, клієнтську частину з логікою інтерфейсу, серверний блок із бізнес-процесами, а також дві кінцеві точки обміну даними: базу та зовнішній API-шлюз. Суцільні стрілки позначають основний вектор руху запитів і відповідей; додаткові лінії вказують на можливість повторних звернень клієнта без участі сервера (наприклад, для кешованого контенту) або на асинхронні виклики зовнішніх сервісів. Таке розділення ролей дає змогу

централізувати бізнес-логіку, понизити навантаження на базу, а також спростити горизонтальне масштабування кожного рівня окремо.

Рис. 2.1. Архітектура клієнт-серверної системи для веб-додатку Rikonomy

(Розроблено автором)

Як видно з діаграми, кожна взаємодія зароджується на боці користувача: у браузері формуються HTTP-запити з маркером сеансу або JWT-токеном, після чого пакети проходять через клієнтський скрипт, який додає параметри інтерфейсу (наприклад, мовні налаштування чи штамп часової зони). Далі запит надходить на серверний рівень, де система автентифікації звіряє права доступу, а шар бізнес-логіки перевіряє коректність даних і приймає рішення, чи звертатися до сховища або до стороннього сервісу. Якщо потрібна операція з базою, сервер надсилає SQL-команду до кластеру, чекає на відповідь і перетворює її на компактний JSON для клієнта. У випадку інтеграції з зовнішнім API, наприклад для конвертації валют, виклик здійснюється асинхронно, аби не

блокувати основний потік, і лише після повернення результату формується остаточна відповідь.

Така трирівнева модель дає змогу розподіляти навантаження оптимально: фронтенд забезпечує миттєве відтворення інтерфейсу завдяки локальному кешу та можливостям Service Worker, середній рівень концентрує всі критичні обчислення та перевірки, а база даних і сторонні API масштабуються горизонтально залежно від зростання аудиторії. У підсумку система зберігає високу швидкодію для користувачів, гнучко підлаштовується під нові функції й залишається надійною навіть під час пікових навантажень. За потреби кожен шар можна модернізувати автономно, мінімізуючи ризики для всього застосунку та скорочуючи час розгортання оновлень.

Варто також наголосити, що побудова системи на основі клієнт-серверної архітектури створює фундамент для майбутньої масштабованості: до API можна підключати нові модулі, мобільні додатки або навіть надбудови для бізнес-аналітики без істотних змін у ядрі системи. Водночас розподіл логіки між фронтендом і бекендом забезпечує зручний процес супроводу, спрощує тестування і локалізацію функціоналу для різних мов чи регіонів. Це особливо важливо у застосунках, орієнтованих на масового користувача.

Таким чином, у процесі розробки Rikonomy враховано актуальні тенденції у побудові веб-систем, а вибір архітектури клієнт-сервер із використанням MySQL як реляційної бази даних, Node.js/Express для бекенду та класичного JavaScript-стека для фронтенду повністю відповідає поставленим завданням щодо надійності, масштабованості, безпеки та зручності у користуванні.

2.2 Проєктування структури бази даних для обліку витрат і користувачів

Бази даних є одним із ключових компонентів сучасних веб-додатків, особливо у сфері фінансового обліку, де точність, надійність та безпека даних є критично важливими. Використання баз даних для збереження фінансових даних

забезпечує низку переваг, які підвищують ефективність управління інформацією та сприяють зростанню користувацького досвіду.

По-перше, бази даних надають централізоване сховище для організації фінансових даних. Це дозволяє зберігати великі обсяги інформації у структурованому вигляді, що полегшує доступ до потрібних даних і їхнє управління. Наприклад, відомості про транзакції, категорії витрат, баланси рахунків та іншу важливу інформацію можна організувати у вигляді таблиць зі зв’язками, що значно спрощує їх обробку.

По-друге, бази даних забезпечують високу точність і надійність обробки фінансових даних. Використання реляційних баз даних, таких як MySQL чи PostgreSQL, дозволяє гарантувати цілісність даних завдяки підтримці транзакцій та використанню механізмів перевірки консистентності. Це особливо важливо у фінансових додатках, де помилка може призвести до значних матеріальних збитків.

Ще однією важливою перевагою є можливість забезпечення багатокористувацького доступу без втрати продуктивності. Сучасні системи управління базами даних (СУБД) підтримують одночасне обслуговування численних запитів, що робить їх ідеальними для використання у веб-додатках, таких як інструменти для обліку особистих фінансів. Наприклад, фінансова платформа може підтримувати синхронний доступ багатьох користувачів до їхніх даних без перешкод чи конфліктів[9].

Безпека є ще одним критично важливим аспектом. Бази даних забезпечують механізми автентифікації та авторизації користувачів, шифрування даних, а також можливість регулярного резервного копіювання. Це дозволяє мінімізувати ризик втрати чи компрометації фінансових даних. Зокрема, сучасні СУБД, такі як PostgreSQL, надають функціонал для захисту чутливих даних, наприклад через шифрування поля або ролі доступу.

Крім того, використання баз даних відкриває широкі можливості для аналітики. Дані, які зберігаються у структурованому вигляді, легко піддаються аналізу за допомогою вбудованих інструментів СУБД або зовнішніх аналітичних

платформ. Це дозволяє користувачам створювати звіти про витрати, виявляти фінансові тенденції та оптимізувати власні ресурси.

Гнучкість баз даних у роботі з великими обсягами інформації та можливість інтеграції з іншими системами є ще однією вагомою перевагою. Наприклад, бази даних NoSQL, такі як MongoDB, дозволяють працювати з неструктурованими даними, що є зручним у випадках зберігання текстових нотаток чи метаданих транзакцій.

Таким чином, використання баз даних у веб-додатках для збереження фінансових даних не лише підвищує точність і надійність, а й забезпечує високий рівень безпеки, гнучкості та аналітичних можливостей. Це робить бази даних незамінним інструментом для створення ефективних систем обліку фінансів.

Процес розробки сучасної інформаційної системи для обліку особистих фінансів, такої як Rikonomy, передбачає створення надійної та гнучкої структури бази даних, яка задовольнятиме вимоги до зберігання даних, їхньої цілісності, масштабованості та безпеки. Після визначення оптимальної архітектури, відповідно до якої використовується реляційна база даних, наступним етапом є побудова логічної структури таблиць. Кожна таблиця відображає конкретну сутність предметної області та має набір полів із визначеними типами даних та обмеженнями.

Щоб отримати загальне уявлення про структуру бази даних Rikonomy, спершу варто навести список основних таблиць, які використовуються в системі, та коротко описати їхню роль (табл.2.1).

Таблиця 2.1

Перелік таблиць бази даних Rikonomy

Назва	Опис
users	Дані про користувачів системи
expenses	Відомості про фінансові операції (витрати)
budgets	Загальні місячні бюджети користувачів
budget_categories	Ліміти по категоріях витрат у межах бюджету

(Розроблено автором)

Кожна із перелічених таблиць містить власні поля із визначеними типами даних. У сучасних реляційних СКБД (зокрема, MySQL, яку використано у Rikonomy) основними типами є:

– INT, цілі числа, що використовуються для ідентифікаторів, кількісних значень;
– DECIMAL, числа із фіксованою точністю для сум і бюджетів;
– VARCHAR, текстові поля для імен, описів, категорій, логінів тощо;
– DATE, DATETIME, для фіксації дат і часу;
– BLOB, для можливості зберігання зображень або файлів (у даній версії Rikonomy не використовується задля оптимізації навантаження на сервер).

Далі подано детальний опис основних таблиць, що формують структуру бази даних. У цьому розгляді акцент зроблено на тому, які саме поля є обов’язковими для зберігання інформації, а також на потенційних можливостях подальшого розширення цієї структури.

Таблиця users є центральною, оскільки через неї здійснюється зв’язок із усіма іншими сутностями. У цій таблиці зберігаються дані, необхідні для ідентифікації користувача, здійснення автентифікації, а також деякі атрибути, які визначають вигляд та індивідуальні налаштування облікового запису. Опис полів наведено у таблиці 2.2.

Таблиця 2.2

Опис сутності «users»

Назва	Тип даних	Обов’язковість	Короткий опис
id	INT	PRIMARY KEY; NOT NULL	Унікальний ідентифікатор
username	VARCHAR(50)	NOT NULL; UNIQUE	Унікальне ім’я користувача
password	VARCHAR(255)	NOT NULL	Хешований пароль
profile_picture_url	VARCHAR(255)		Посилання на фото профілю

(Розроблено автором)

Завдяки обов’язковості для ключових полів (id, username, password) забезпечується унікальність кожного запису та відповідність вимогам безпеки. Довжина поля для паролів дозволяє зберігати хеші сучасних алгоритмів захисту. Поле для фото профілю дозволяє реалізувати персоналізацію облікового запису.

Основна мета таблиці expenses – зберігання інформації про всі витрати користувача. Вона містить як мінімально необхідні атрибути (сума, категорія, дата), так і додаткові (опис, валюта). Кожна витрата обов’язково прив’язана до певного користувача через зовнішній ключ user_id. Деталізований опис полів подано у таблиці 2.3.

Таблиця 2.3

Опис сутності «expenses»

Назва	Тип даних	Обов’язковість	Короткий опис
1	2	3	4
id	INT	PRIMARY KEY; NOT NULL	Унікальний ідентифікатор витрати
amount	DECIMAL(10,2)	NOT NULL	Сума витрати
category	VARCHAR(50)	NOT NULL	Категорія витрати
description	VARCHAR(255)		Опис призначення
date	DATE	NOT NULL	Дата здійснення витрати
currency	VARCHAR(10)	NOT NULL	Валюта витрати
user_id	INT	NOT NULL	Зовнішній ключ на користувача

(Розроблено автором)

Поле amount забезпечує точне збереження суми витрати у вибраній валюті. Категорія та опис дають змогу додатково характеризувати кожну операцію, що спрощує подальший аналіз і пошук. Обов’язковий user_id дозволяє однозначно прив’язати витрату до конкретного користувача, що гарантує цілісність даних при зберіганні та обробці.

Таблиця budgets використовується для фіксації даних про бюджети кожного користувача окремо по місяцях. Саме вона дає змогу не лише встановлювати планові суми витрат, а й порівнювати їх із фактичними витратами у межах конкретного календарного місяця. Структуру цієї таблиці детально подано у таблиці 2.4.

Таблиця 2.4

Опис сутності «budgets»

Назва	Тип даних	Обов'язковість	Короткий опис
id	INT	PRIMARY KEY; NOT NULL	Унікальний ідентифікатор
user_id	INT	NOT NULL	Зовнішній ключ на користувача
month	VARCHAR(7)	NOT NULL	Місяць (формат YYYY-MM)
amount	DECIMAL(10,2)	NOT NULL	Сума бюджету
currency	VARCHAR(10)	NOT NULL	Валюта бюджету

(Розроблено автором)

Кожен запис у таблиці бюджету однозначно належить одному користувачеві та одному місяцю, що спрощує фільтрацію й аналіз у розрізі періодів. Додаткова підтримка різних валют дозволяє використовувати систему користувачам із різних країн.

Окремо у Rikonomy реалізовано таблицю budget_categories, яка дає змогу фіксувати ліміти по категоріях витрат у межах загального бюджету користувача. Це дозволяє реалізувати більш гнучке управління фінансами та дає змогу користувачам контролювати окремі аспекти витрат. Опис структури подано у таблиці 2.5.

Таблиця 2.5

Опис сутності «budget_categories»

Назва	Тип даних	Обов'язковість	Короткий опис
id	INT	PRIMARY KEY; NOT NULL	Унікальний ідентифікатор
user_id	INT	NOT NULL	Зовнішній ключ на користувача
month	VARCHAR(7)	NOT NULL	Місяць (формат YYYY-MM)
category	VARCHAR(32)	NOT NULL	Категорія витрат
amount	DECIMAL(12,2)	NOT NULL	Ліміт витрат по категорії
currency	VARCHAR(8)	NOT NULL	Валюта категорії

(Розроблено автором)

Така структура дозволяє користувачу встановити обмеження на окремі категорії (наприклад, «транспорт», «харчування», «розваги») в межах кожного місяця. Всі записи також прив’язані до конкретного користувача, що забезпечує індивідуалізацію бюджету.

Завдяки використанню зовнішніх ключів між таблицями підтримується референційна цілісність – при видаленні користувача видаляються пов’язані з ним бюджети й витрати. Всі фінансові значення зберігаються у типі DECIMAL із достатньою точністю для уникнення похибок під час розрахунків.

В описаній структурі передбачена можливість масштабування: у разі необхідності система може бути доповнена новими таблицями – наприклад, для обліку доходів, регулярних платежів, історії курсів валют або обліку спільних рахунків для сім’ї чи бізнесу. Крім того, організація даних у такому вигляді дозволяє легко інтегрувати додатковий функціонал для аналітики, експорт даних чи автоматизації фінансових звітів.

Назви полів та обрані типи даних у структурі бази були підібрані з урахуванням як потреб самої системи, так і досвіду тих, хто працює з подібними рішеннями. Такі підходи суттєво спрощують роботу із базою у повсякденних завданнях: розробнику не доводиться витрачати зайвий час на пошук потрібного поля чи розшифровку його призначення. Крім того, система спроектована так, щоб у разі появи нових вимог не довелося перебудовувати вже існуючі таблиці чи суттєво змінювати логіку програми. Це, своєю чергою, дозволяє не лише зберігати дані, але й без зайвих труднощів знаходити потрібну інформацію та аналізувати витрати під різними кутами: наприклад, шукати операції по категоріях, за певний місяць, чи будувати загальні звіти для користувача.

Усе це дозволяє досягти балансу між гнучкістю налаштувань, надійністю роботи системи та можливістю поступового розвитку Rikonomy. Обраний підхід відповідає сучасним вимогам до безпеки та зручності персонального фінансового обліку. Крім того, така структура дає змогу легко додавати нові функції чи розширювати можливості додатку в майбутньому, коли зміняться потреби користувачів або з’являться нові ідеї для вдосконалення сервісу.

2.3 Моделювання сутностей, зв’язків і запитів у базі даних

Після того, як визначено основні елементи структури бази даних і детально описано всі ключові таблиці, надзвичайно важливо перейти до етапу моделювання сутностей, їхніх взаємозв’язків і характерних запитів, які будуть виконуватися у системі. Саме на цьому етапі стає зрозуміло, як окремі об’єкти предметної області співвідносяться між собою, чому виникає потреба у використанні зовнішніх ключів, і як це впливає на подальшу логіку роботи застосунку. Якісно продумана модель бази даних дає не лише впевненість у коректності зберігання та обробки інформації, а й створює міцну основу для побудови складних вибірок, реалізації аналітичних функцій та подальшої інтеграції з іншими інформаційними системами.

Структура бази даних веб–системи Rikonomy ілюструє комплексний підхід до організації даних для персонального фінансового обліку. У центрі моделі розміщено сутність користувача, яка виступає основою для формування всіх операцій та взаємозв’язків у системі. Кожен користувач має унікальний ідентифікатор, на який спираються решта сутностей: витрати, бюджети, категорії. Зв’язок між користувачем і витратами реалізовано через зовнішній ключ у таблиці expenses, що дозволяє зберігати і опрацьовувати інформацію по кожному акаунту індивідуально. Це важливо для конфіденційності даних, а також для впровадження багатокористувацького середовища, в якому кожен запис про витрату належить виключно одному користувачу.

Структура категорій витрат має особливе значення при побудові моделі бази даних. Кожна фінансова операція обов’язково пов’язується з певною категорією, яка відображає її призначення або тип, наприклад, «Їжа», «Транспорт» чи «Освіта». Це дозволяє будувати багатовимірний аналіз фінансової поведінки користувачів, автоматизувати процес формування звітів, а також персоналізувати поради щодо планування бюджету. Таблиця categories містить перелік доступних категорій, а у таблиці expenses для кожного запису зберігається відповідний ідентифікатор категорії. Завдяки цьому всі операції над

витратами можуть бути агреговані, відсортовані та візуалізовані за потрібними параметрами.

В моделі важливим є те, що для кожного користувача окремо зберігається інформація про бюджети. Система дає можливість не просто встановлювати загальний ліміт на витрати, а й задавати окремі обмеження для різних категорій у кожному місяці. Такий підхід дозволяє кожному користувачу гнучко планувати свої витрати й реально стежити за тим, наскільки він дотримується намічених фінансових цілей – як у повсякденному житті, так і з розрахунком на більш тривалий період. Поле user_id у таблиці budgets дозволяє пов’язати бюджетний запис із конкретним користувачем, а поле month фіксує період дії бюджету, що спрощує аналіз динаміки змін за різні часові відрізки.

Детальна схема взаємозв’язків між сутностями наведена на рисунку 2.2.

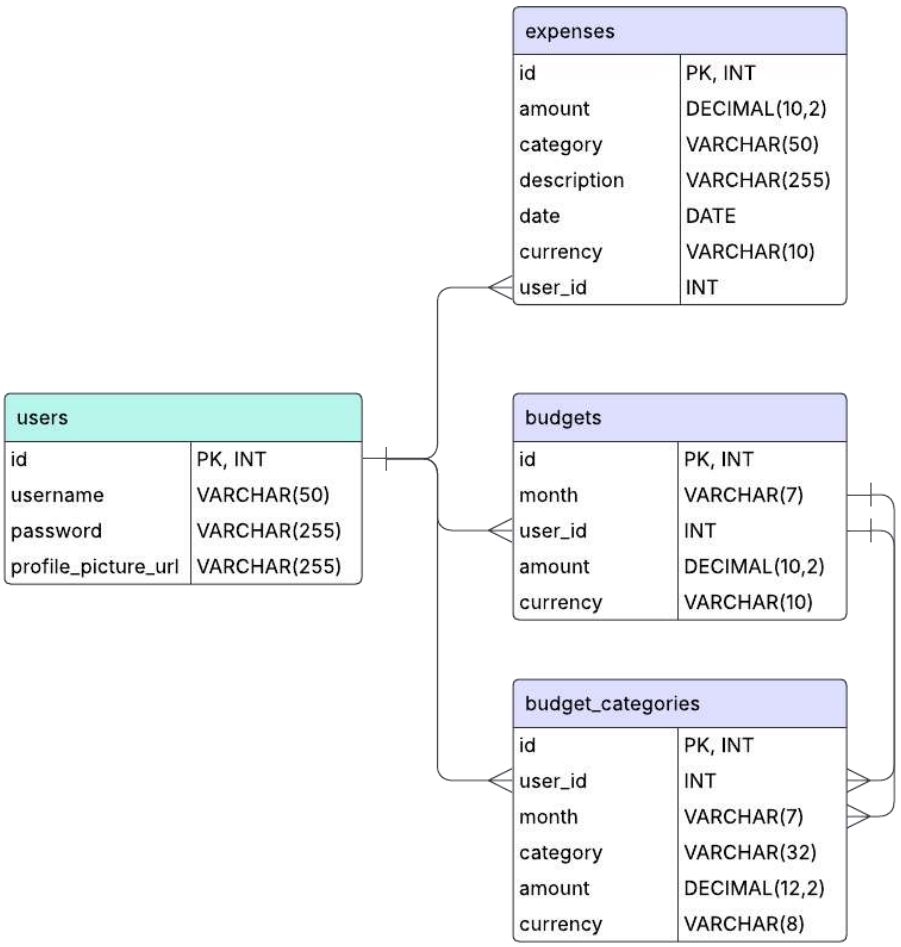

Рис. 2.2. Діаграма сутностей і зв’язків бази даних Rikonomy

(Розроблено автором)

На діаграмі чітко видно, що кожен користувач може мати довільну кількість витрат і бюджетів, а кожна витрата завжди належить до однієї категорії та одного користувача. Це класичні відношення «один–до–багатьох» та «багато–до–одного», які забезпечують структурованість, уникнення дублювання й логічну узгодженість інформації.

Аналізуючи модель, можна виділити основні логічні зв’язки, що забезпечують ефективність роботи з даними. Зокрема, у таблиці expenses зовнішні ключі user_id та category дають змогу пов’язати кожну витрату з конкретним користувачем та певною категорією. У свою чергу, у таблиці budgets зв’язок із users через user_id дозволяє встановлювати персональні ліміти, а також робити порівняння запланованих і фактичних витрат. Таблиця categories має зв’язок лише із expenses, що дозволяє централізовано адмініструвати перелік категорій і спрощує розширення функціоналу.

Реалізація зовнішніх ключів дає ще одну важливу перевагу – автоматичне забезпечення цілісності даних на рівні СКБД. Це означає, що жодна витрата не може бути додана без наявності користувача та відповідної категорії, а у випадку видалення користувача або категорії система може виконувати каскадне видалення всіх пов’язаних записів, тим самим уникнувши появи «висячих» даних. Подібна організація дозволяє не тільки підтримувати порядок у таблицях, але й значно спрощує аудит та резервне копіювання, а також полегшує майбутні міграції чи розширення функціоналу системи.

Побудова ефективної структури відображається на реалізації типових та складних запитів, необхідних для функціонування Rikonomy. Наприклад, для відображення всіх витрат користувача за певний місяць використовується вибірка із застосуванням фільтрів за user_id та датою. Для аналітики витрат по категоріях застосовується агрегація сум по полю category із подальшим групуванням результатів, що дозволяє будувати статистику у вигляді графіків. Запити для визначення залишку бюджету чи перевірки виконання лімітів включають об’єднання таблиць budgets та expenses за user_id і порівняння фактичних сум із запланованими. Такий підхід забезпечує високу швидкість

доступу навіть при великих обсягах інформації, дозволяє масштабувати систему та адаптувати її під зростаючі навантаження.

Під час створення структури бази даних питання безпеки й розмежування доступу стояли на одному з перших місць. Система організована таким чином, що користувач може працювати лише зі своїми особистими записами – жодні дані інших облікових записів не стають йому доступними ані для перегляду, ані для редагування. Для всіх дій, пов’язаних із критично важливою інформацією, обов’язково застосовується авторизація, і лише після підтвердження особи відкривається доступ до відповідних функцій. На додачу, на рівні самої структури даних передбачене використання зовнішніх ключів, які дозволяють уникнути появи «висячих» записів та забезпечують цілісність зв’язків між таблицями. Такий підхід не лише мінімізує ймовірність технічних помилок при додаванні або зміні даних, а й значно підвищує довіру до системи, оскільки будь-яке несанкціоноване втручання у чужу інформацію фактично виключене.

Важливо наголосити, що така модель дозволяє у майбутньому легко розширювати функціонал: додавати нові типи сутностей (наприклад, доходи, кредити, інвестиції), розширювати довідники категорій чи реалізовувати спільні бюджети для груп користувачів. Структура бази даних спроєктована так, щоб міграції, оновлення та резервне копіювання можна було виконувати без втрати цілісності даних, а підтримка індексів на основних полях дозволяє підтримувати високу продуктивність навіть при значних обсягах інформації.

Отже, виконане моделювання сутностей і зв’язків у базі даних Rikonomy забезпечує надійну основу для організації багатокористувацької системи обліку фінансів, підтримує цілісність, масштабованість і безпеку даних. Логічна структура і гнучка модель вибірок та агрегацій дозволяють легко впроваджувати нові можливості, не порушуючи узгодженості вже збереженої інформації, що є одним із визначальних чинників довгострокового розвитку й конкурентоспроможності веб-додатку.

Висновки до розділу 2

У цьому розділі було проаналізовано архітектурні рішення та ключові підходи до проєктування структури бази даних для веб-системи обліку особистих фінансів Rikonomy. Дослідження різних варіантів архітектури та сучасних СКБД дозволило обґрунтовано обрати класичну клієнт-серверну модель із використанням реляційної бази даних MySQL, що забезпечує необхідну гнучкість, безпеку та можливість масштабування системи в майбутньому.

Створена структура бази даних охоплює всі головні потреби багатокористувацької фінансової системи: від зберігання інформації про користувачів і ведення обліку витрат до індивідуального планування бюджету й деталізації лімітів для кожної категорії. В процесі проєктування особливу увагу приділяли цілісності даних, захисту від несанкціонованого доступу та тому, щоб система могла легко пристосовуватись до нових завдань або змін у логіці роботи.

Усі таблиці в базі пов’язані між собою через зовнішні ключі. Це допомагає підтримувати порядок у даних та стежити за тим, щоб жоден запис не залишався без прив’язки до основної інформації. Така структура спрощує роботу з даними: можна без зайвих труднощів додавати нові можливості, вносити зміни чи розширювати функціонал системи, без страху порушити цілісність моделі.

Ретельно опрацьована модель сутностей та зв’язків дає змогу ефективно реалізувати типові та складні запити, необхідні для побудови звітів, аналізу витрат і контролю за дотриманням бюджетних лімітів. Закладена логіка роботи із даними, механізми контролю доступу та модульність структури створюють фундамент для стабільної, безпечної й гнучкої системи, яка відповідає сучасним вимогам до автоматизованих інструментів фінансового обліку.

Усі взаємозв’язки між таблицями побудовані на використанні зовнішніх ключів. Такий підхід дає змогу не лише підтримувати референційну цілісність інформації, а й значно спрощує процес організації та супроводу даних у системі. Це забезпечує запобігання появі некоректних чи дубльованих записів, а також

дає змогу без труднощів реалізовувати додаткові функції чи змінювати модель відповідно до нових сценаріїв використання системи, що виникають під час її подальшого розвитку.

РОЗДІЛ 3
РОЗРОБКА ВЕБ-СИСТЕМИ ОБЛІКУ ОСОБИСТИХ ФІНАНСІВ RIKONOMY

3.1 Обгрунтування вибору інструментів для створення веб-застосунку

Вибір інструментів для створення веб-застосунку Rikonomy, присвяченого обліку особистих фінансів, на першому етапі розробки був визначальним для подальшої долі всієї системи. Ще на стадії формування вимог до програмного продукту постала потреба забезпечити сучасний рівень швидкодії, безпеки, гнучкості, а також зручності у використанні – як для пересічного користувача, так і для розробників, котрі будуть підтримувати та розвивати проект у майбутньому.

Перед початком активної розробки було опрацьовано низку технічних рішень, які стосуються вибору архітектури, мови програмування, системи керування базами даних, а також підходів до організації клієнтської й серверної частин. Для кожного елемента системи оцінювалися альтернативи: враховувалися як технічні, так і організаційні фактори – доступність кваліфікованих кадрів, відкритість інструменту, підтримка спільноти, наявність актуальної документації, стабільність роботи у production-середовищі.

Багато уваги було приділено саме моделі розділення на frontend і backend, що давно стало стандартом для сучасних web-додатків. Після порівняльного аналізу різних стеків вибір для серверної логіки зупинився на середовищі Node.js у поєднанні з фреймворком Express. Серед конкурентів розглядалися такі варіанти, як Python із Django/Flask, PHP (Laravel), Ruby on Rails, Java (Spring), але Node.js забезпечив оптимальний баланс між простотою розгортання, гнучкістю та широким вибором готових бібліотек для роботи з HTTP-запитами, авторизацією, захистом, а також з базами даних. Node.js дозволяє організувати неблокуючу обробку запитів – важливий нюанс для проєкту, що орієнтований на

роботу з багатьма паралельними підключеннями, наприклад, при оновленні інформації про фінансові операції або масовій реєстрації користувачів.

Express обрано як базу для розробки API, оскільки цей фреймворк відомий своєю простотою та гнучкістю. На практиці його застосовують у багатьох комерційних і відкритих проєктах, адже він дозволяє зручно впорядковувати код, дотримуватися зрозумілої архітектури та впроваджувати проміжні ланки (middleware) для авторизації, логування чи додаткових перевірок. Ще однією перевагою Express є відсутність жорстких вимог до структури проєкту – це відкриває можливість будувати маршрути і компоненти саме так, як зручно для конкретного бізнес-завдання, та підлаштовувати систему під будь-які специфічні потреби.

Окремої уваги заслуговує питання зберігання даних. Для Rikonomy, як для класичного фінансового застосунку, потрібна надійна система керування базою даних із підтримкою транзакцій, контролем цілісності, гнучким механізмом зв’язків і швидкою обробкою запитів. Реляційна СКБД MySQL стала основою зберігання: вона має багаторічний досвід промислового використання, гарантовано працює зі складними зв’язками між таблицями, дозволяє організовувати вибірки для звітів, діаграм і статистичних аналізів. Вибір саме MySQL, а не NoSQL-рішень (як-от MongoDB чи Cassandra), обумовлений необхідністю суворої структурованості даних, можливістю гнучко описати взаємозв’язки (наприклад, витрати прив’язані до конкретного користувача й категорії, а бюджети – до певного місяця й категорії), а також високим рівнем підтримки з боку сучасних фреймворків для JavaScript. У разі потреби масштабування MySQL добре інтегрується із засобами для резервного копіювання, кластеризації та автоматичного відновлення після збоїв.

Особливу увагу у процесі вибору інструментів приділяли питанням кібербезпеки та захисту особистих даних користувачів. У світі сучасного фінансового програмного забезпечення це ключова вимога, яка прямо впливає на довіру до системи. У Rikonomy використовуються сучасні практики безпечного зберігання й передачі паролів: дані паролів хешуються з

використанням бібліотеки bcrypt – одного з промислових стандартів для такого завдання. Це унеможливлює зворотне отримання пароля навіть у разі компрометації бази даних. Для обробки авторизації, реєстрації сесій і передачі даних між клієнтом і сервером застосовано механізм JWT (JSON Web Token): кожному користувачу після входу видається унікальний токен, який використовується для автентифікації у подальших запитах до API. Токени зберігаються лише на клієнтській стороні, що зменшує ймовірність їх викрадення на сервері. На додаток, middleware cors суворо обмежує список допустимих доменів для запитів до API, а body-parser забезпечує коректне зчитування та перевірку вхідних даних.

Під час впровадження Rikonomy був врахований ще один критичний момент – гнучкість і масштабованість розробки. Обраний стек дає змогу у будь-який момент розширити функціонал (наприклад, додати нові види звітів, інтегрувати сторонні API, підключити мобільний застосунок) без глобального переписування ядра або ризику втратити цілісність даних. Наприклад, підключення модуля для автоматичного оновлення курсів валют (на основі відкритих API) або впровадження багатомовної підтримки не потребує значної перебудови архітектури. Завдяки Node.js/Express будь-яку додаткову бізнес-логіку чи сторонній модуль можна впроваджувати поступово, розширюючи існуючі маршрути або додаючи нові, що не порушує роботу вже впровадженого функціоналу.

Ще на етапі проєктування фронтенду важливим було створення максимально дружнього й інтуїтивно зрозумілого інтерфейсу. Саме тому обрані стандартні технології: HTML5 (структура та наповнення сторінок), CSS3 (стилізація, адаптивний дизайн, підтримка темної/світлої теми, card-based design), JavaScript (клієнтська логіка, робота з REST API, обробка подій та анімацій). Цей вибір пояснюється універсальністю, легкістю підтримки, великим вибором бібліотек, а головне – тим, що практично кожен сучасний браузер без проблем працює із цими технологіями. Для візуалізації фінансових даних

інтегровано Chart.js: цей інструмент дозволяє відображати структуру витрат, зміни бюджету, порівнювати дані по місяцях або категоріях на наочних графіках.

Всі ці рішення спрямовані на те, щоб користувач міг у кілька кліків отримати повну аналітику своїх фінансів, не витрачаючи час на налаштування чи вивчення складних інтерфейсів. У фронтенді також реалізовано багатомовність (інтерфейс легко перекладається), підтримується пошук і фільтрація по категоріях, датах, валюті, а завдяки card-based дизайну можна швидко знаходити потрібну інформацію. Окремо організовано систему повідомлень і валідації введених даних, що знижує ризик помилок і підвищує зручність користування навіть для новачків.

Ще однією перевагою такого технологічного вибору є можливість оперативного резервного копіювання даних: MySQL дозволяє налаштувати регулярне створення резервних копій, що знижує ризики втрати інформації у разі технічних збоїв або аварій. Водночас завдяки відкритій структурі серверної частини (Node.js/Express) та чіткій схемі бази даних, у майбутньому можна реалізувати перенесення на більш потужні сервери чи у хмарні сервіси (AWS, Google Cloud, Azure) без суттєвих змін у коді.

У процесі вибору інструментів враховувалися також і питання сумісності з мобільними пристроями: якщо згодом виникне потреба створити нативний або кросплатформний мобільний додаток, RESTful API на Node.js/Express дозволить реалізувати це швидко й безболісно – мобільний застосунок зможе взаємодіяти з тим же сервером, що й браузерна версія.

Особливу роль відіграє активна підтримка вибраних інструментів спільнотою розробників. Це знижує ризики виникнення «мертвих» технологій, пришвидшує пошук рішень для типових проблем і дає змогу швидко оновлювати компоненти для захисту від нових вразливостей. Усі бібліотеки, які застосовуються, мають відкритий код, що дозволяє перевірити їхню надійність, а також не обмежують розвиток Rikonomy ліцензійними обмеженнями.

Підсумовуючи, усі технологічні рішення, що використовуються у Rikonomy, ретельно добиралися на основі досвіду провідних галузевих проєктів

і реальних потреб системи. Кожен інструмент обрано так, щоб забезпечити і захист даних, і можливість поступового розширення функціоналу без перебудови всієї структури. Завдяки такому підходу система не лише виконує поточні завдання, але й залишається відкритою до нових ідей, інтеграції зі сторонніми сервісами та впровадження змін, які можуть виникнути з розвитком IT-сфери чи появою нових запитів від користувачів.

3.2 Розробка серверної частини із захистом доступу та обробкою запитів

Розробка серверної частини Rikonomy ґрунтувалася на принципах структурованості, безпеки й простоти масштабування. Саме серверний код визначає, як обробляються всі запити користувачів, як виконується бізнес-логіка, які дані потрапляють у базу й повертаються у відповідь на запити клієнтів.

З самого початку була побудована чітка структура, де ядро серверної частини – файл server.js, у якому зосереджена основна логіка взаємодії з базою даних, а також маршрутизація та обробка різних запитів API. Використання середовища Node.js разом із Express забезпечило неблокуючу, асинхронну обробку запитів, що важливо для швидкодії навіть при великій кількості активних користувачів.

Перший крок – налаштування середовища та підключення необхідних модулів. У проекті використовуються такі бібліотеки, як mysql2/promise для роботи з базою даних, jsonwebtoken для управління токенами авторизації, bcrypt для хешування паролів, а також cors для контролю доступу до API із зовнішніх джерел.

Використання dotenv дозволяє винести конфіденційні налаштування (паролі до бази, секретні ключі, порти) у окремий файл .env, не включаючи їх до відкритого коду. Це один із стандартних підходів до забезпечення безпеки на рівні інфраструктури.

Для підключення до бази даних використовується пул з’єднань, де всі параметри конфігуруються через змінні середовища. Такий підхід ілюстровано на рисунку 3.1.

```
const db = mysql.createConnection({
    host: process.env.DB_HOST,
    user: process.env.DB_USER,
    password: process.env.DB_PASSWORD,
    database: process.env.DB_NAME,
    waitForConnections: true,
    connectionLimit: 10,
    queueLimit: 0
});
```

Рис. 3.1. Підключення до бази даних через пул з’єднань у Rikonomy

(Розроблено автором)

Усі налаштування підтягуються з .env, що дозволяє швидко змінювати параметри розгортання. Дії, які стосуються авторизації користувача й захисту доступу до фінансових операцій, реалізовані із застосуванням JWT (JSON Web Token). Процес перевірки токенів наведено на рисунку 3.2.

```
function authenticateToken(req, res, next) {
    const authHeader = req.headers['authorization'];
    const token = authHeader && authHeader.split(' ')[1];
    if (!token) return res.status(401).send('Access denied. Token missing.');

    jwt.verify(token, SECRET_KEY, (err, user) => {
        if (err) return res.status(403).send('Invalid or expired token.');
        req.user = user;
        next();
    });
}
```

Рис. 3.2. Фрагмент коду з middleware для перевірки JWT-токену

(Розроблено автором)

Цей middleware перевіряє, чи додає користувач до кожного запиту дійсний токен, і лише після цього дозволяє виконання дії. Таким чином, навіть якщо хтось спробує зробити запит до API, не маючи доступу – відповідь буде 401 (Unauthorized) або 403 (Forbidden).

Реєстрація користувачів супроводжується хешуванням паролів через bcrypt. Це означає, що у базі ніколи не зберігаються відкриті паролі. Фрагмент коду, який це реалізує, наведено на рисунку 3.3.

```
app.post('/register', async (req, res) => {
    const { username, password } = req.body;

    if (!username || !password) {
        return res.status(400).send('Username and password are required.');
    }

    const checkUserQuery = 'SELECT * FROM users WHERE username = ?';
    db.query(checkUserQuery, [username], async (err, results) => {
        if (err) {
            console.error('Error checking user:', err);
            return res.status(500).send('Server error.');
        }

        if (results.length > 0) {
            return res.status(400).send('Username already exists.');
        }

        const hashedPassword = await bcrypt.hash(password, 10);
```

Рис. 3.3. Хешування пароля під час створення користувача

(Розроблено автором)

Якщо пароль навіть потрапить у руки зловмисника, декодувати його буде практично неможливо. Перевірка автентичності користувача здійснюється шляхом порівняння хешу пароля із введеним значенням. Алгоритм авторизації зображено на рисунку 3.4.

```
app.post('/login', async (req, res) => {
    const { username, password } = req.body;
```

Рис. 3.4. Перевірка пароля та видача JWT при авторизації

(Розроблено автором)

У відповідь користувач отримує JWT-токен, який буде використовувати для всіх подальших запитів.

Обробка фінансових операцій завжди прив’язана до конкретного користувача: ID користувача витягується з токена, тому жоден запит не зможе

внести зміни у чужі дані. Приклад отримання витрат для авторизованого користувача проілюстровано на рисунку 3.5.

```
app.get('/expenses', authenticateToken, (req, res) => {
    const userId = req.user.id; // ID авторизованого користувача
    const { categories, startDate, endDate } = req.query;
    let query = 'SELECT * FROM expenses WHERE user_id = ?';
    const params = [userId];

    if (categories) {
        const categoryList = categories.split(',');
        query += ` AND category IN (${categoryList.map(() => '?').join(',')})`;
        params.push(...categoryList);
    }
    if (startDate) {
        query += ' AND date >= ?';
        params.push(startDate);
    }
    if (endDate) {
        query += ' AND date <= ?';
        params.push(endDate);
    }

    db.query(query, params, (err, results) => {
        if (err) {
            console.error('Error fetching expenses:', err);
            res.status(500).send('Error fetching expenses');
        } else {
            res.json(results);
        }
    });
});
```

Рис. 3.5. Вибірка витрат користувача із бази даних

(Розроблено автором)

Усі інші дії з витратами (додавання, видалення, редагування) працюють за аналогічним принципом – жоден сторонній користувач не може змінити чи побачити чужі дані. Щоб запобігти SQL-ін’єкціям, усі запити до бази параметризуються. Приклад такої реалізації подано на рисунку 3.6.

```
const query = 'INSERT INTO expenses (amount, currency, category, description, date, user_id) VALUES (?, ?, ?, ?, ?, ?)';
db.query(query, [numericAmount, currency, category, description, date, req.user.id], (err) => {
```

Рис. 3.6. Використання підготовлених інструкцій у запитах до бази

(Розроблено автором)

Параметризація дозволяє уникнути впливу шкідливих даних у запитах, навіть якщо користувач намагається підставити некоректний SQL-код. Додатковим захистом від несанкціонованого доступу виступає обмеження запитів лише до дозволених доменів, що налаштовується через CORS, як на рисунку 3.7.

```
app.use(cors({
    origin: process.env.ALLOWED_ORIGINS?.split(','),
    credentials: true
}));
```

Рис. 3.7. Обмеження доступу до API за допомогою CORS

(Розроблено автором)

Обробка помилок – ще одна важлива частина серверної логіки. У випадку виникнення помилок сервер повертає коротке повідомлення для користувача, а технічні деталі журналює. Реалізація такої логіки наведена на рисунку 3.8.

```
console.error('Login error:', err);
res.status(500).send("Server error.");
```

Рис. 3.8. Обробка нештатних ситуацій на сервері Rikonomy

(Розроблено автором)

Це унеможливлює розкриття технічних деталей зловмисникам та спрощує діагностику розробникам. Сервер Rikonomy також підтримує обробку різних фільтрів (за датою, категорією, сумою), роботу з бюджетами, категоріями, динамічне оновлення курсу валют тощо. Вся бізнес-логіка побудована так, щоб забезпечити ізоляцію даних, уникнути надмірного дублювання коду та полегшити підтримку нових сценаріїв використання.

Регулярне резервне копіювання бази даних організовано окремими процесами (не у цьому коді, але інтегрується за потреби), а структура таблиць дозволяє виконувати міграції без втрати існуючих даних.

У підсумку, серверна частина Rikonomy поєднує сучасні підходи до безпеки (JWT, bcrypt, CORS, SQL-параметризація), зручну організацію коду та гнучку логіку, завдяки чому система лишається стійкою до атак, адаптивною до нових вимог і зручною для подальшого розвитку. Жоден фрагмент не є зайвим чи надто формальним – кожен шматок коду має своє місце й обґрунтування.

3.3 Реалізація клієнтської частини з інтерактивним інтерфейсом

Клієнтська частина веб-додатку Rikonomy розроблена як односторінковий застосунок із динамічним оновленням вмісту, що забезпечує інтуїтивно зрозумілу взаємодію для користувача. Інтерфейс структуровано навколо декількох основних функціональних розділів: автентифікація, управління профілем, робота з витратами, планування бюджету, статистика та допомога.

Після запуску додатку користувачу відображається компактна форма для введення облікових даних, що дає змогу виконати вхід до системи або зареєструвати новий обліковий запис (рис. 3.9). Дана форма містить поля для введення імені користувача та пароля, а також інтерактивні кнопки для переходу до потрібної дії. Валідація введених даних здійснюється без перезавантаження сторінки, а повідомлення про помилки виводяться поруч із полями форми.

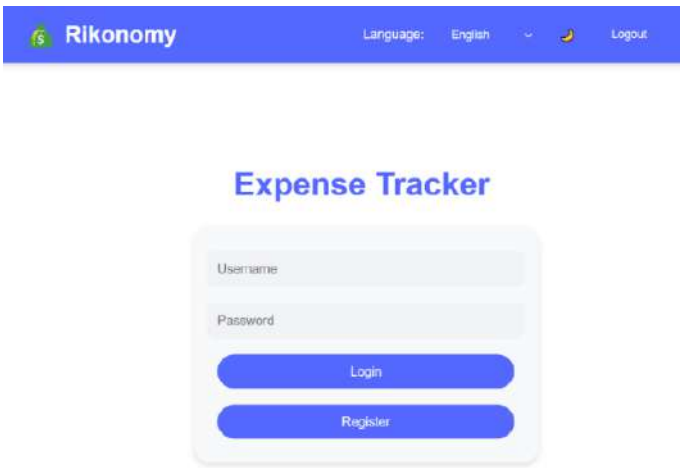

Рис. 3.9. Форма авторизації користувача у Rikonomy

(Розроблено автором)

Після успішної автентифікації інтерфейс змінюється – на екрані відображається ліве меню навігації з піктограмами для швидкого доступу до основних розділів: профілю, витрат, бюджету, статистики й розділу допомоги. Над меню знаходиться логотип системи та перемикачі мови й теми (світла/темна), які діють для всього застосунку. На рисунку 3.10 показано зовнішній вигляд розділу профілю користувача, де відображаються основна інформація про користувача, можливість зміни аватара, вибір поточного місяця і валюти для перегляду бюджету та залишку.

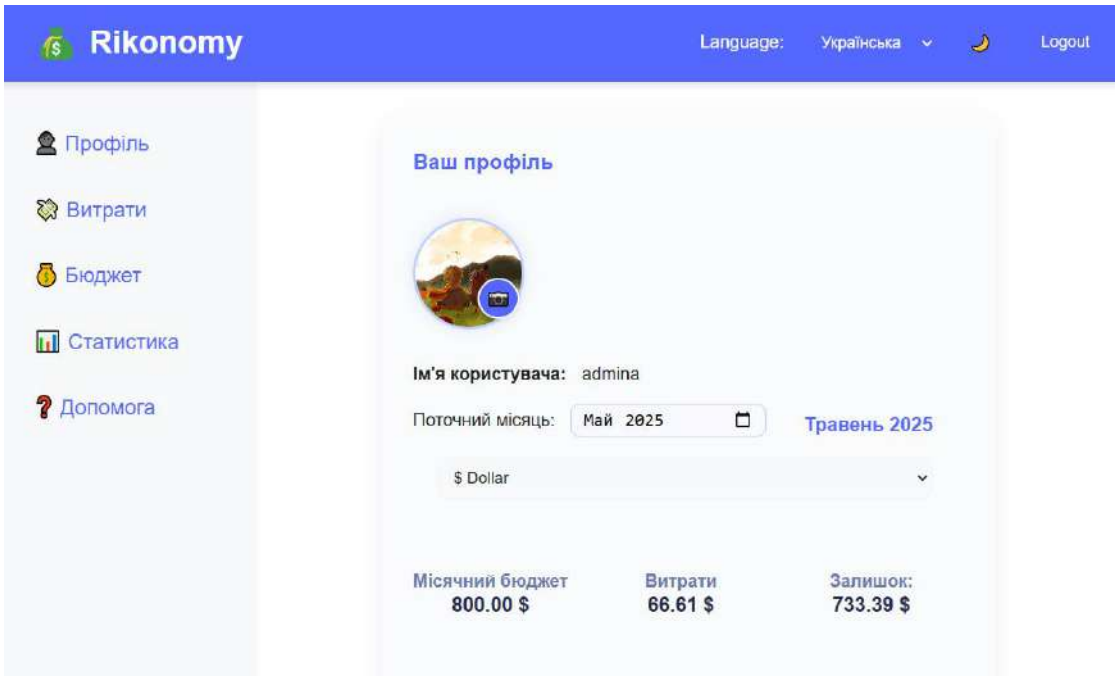

Рис. 3.10. Сторінка профілю користувача після входу

(Розроблено автором)

Суттєвою особливістю інтерфейсу є реалізація мультимовності. Усі написи, підказки й кнопки динамічно змінюються відповідно до вибраної мови користувача. Перемикання мови не призводить до перезавантаження сторінки, а усі текстові елементи підтягуються із локалізованих словників.

Розділ «Витрати» містить форму для додавання нових витрат, а також фільтри для пошуку та аналізу вже існуючих записів (рис. 3.11). Користувач має змогу обрати валюту, категорію витрати, вказати опис та дату.

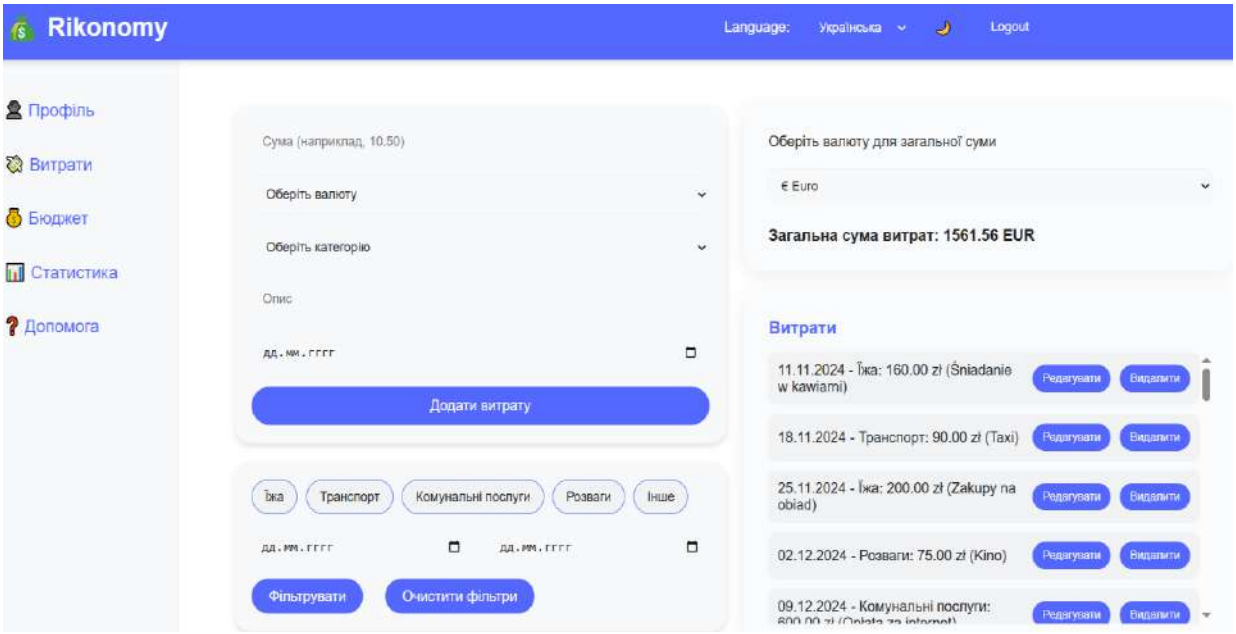

Рис. 3.11. Робота з витратами: додавання, перегляд і фільтрація

(Розроблено автором)

Після підтвердження новий запис одразу відображається у списку витрат, де також передбачено можливість редагування чи видалення кожного запису. Окремо реалізовано підрахунок загальної суми витрат з автоматичною конвертацією в обрану валюту – курси валют отримуються із зовнішнього API і відображаються в режимі реального часу. Фрагмент коду, що відповідає за рендеринг витрат, подано на рисунку 3.12.

```
function renderExpenses(expenses) {
    const expensesList = document.getElementById('expensesList');
    expensesList.innerHTML = '';

    currentRenderedExpenses = Array.isArray(expenses) ? expenses : [];

    if (currentRenderedExpenses.length === 0) {
        const noExpensesMessage = document.createElement('p');
        noExpensesMessage.textContent = translations[currentLang]?.noExpenses || "No expenses found.";
        expensesList.appendChild(noExpensesMessage);
        calculateTotalWithConversion();
        return;
    }

    currentRenderedExpenses.forEach(expense => {
        const li = createExpenseListItem(expense);
        expensesList.appendChild(li);
    });

    calculateTotalWithConversion();
}
```

Рис. 3.12. Рендеринг витрат

(Розроблено автором)

На сторінці «Бюджет» користувач має змогу налаштувати власний місячний бюджет із розподілом за категоріями (рис. 3.13).

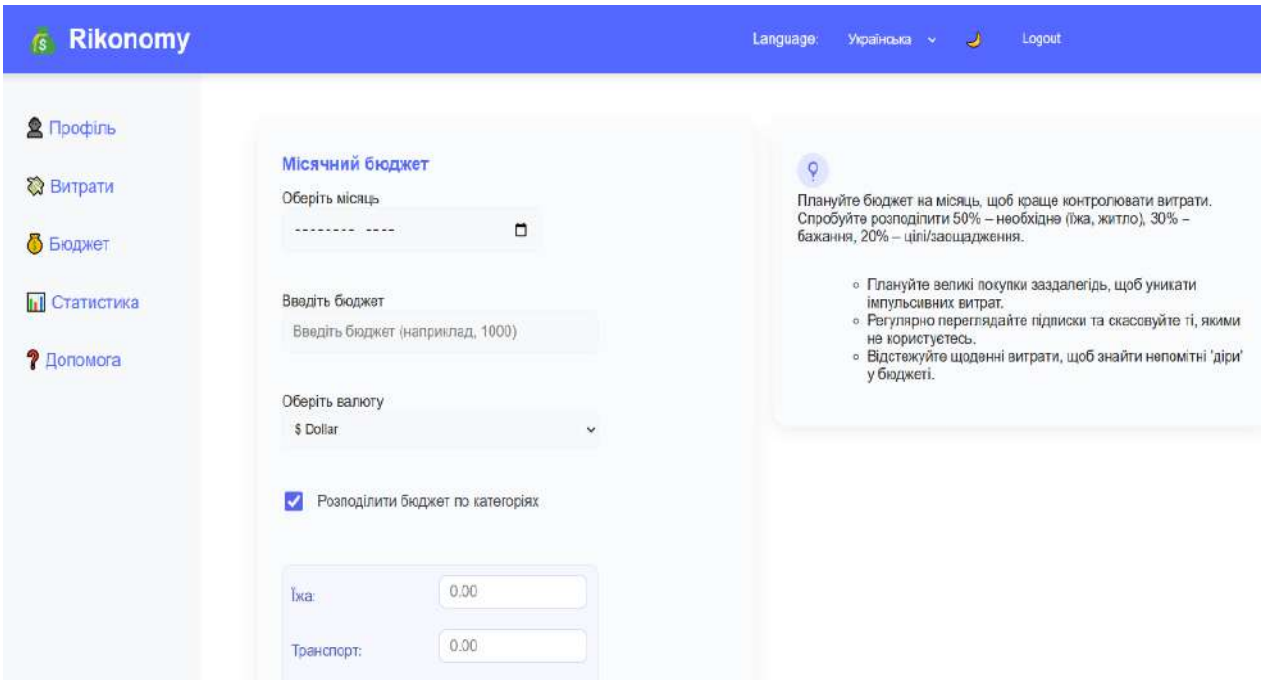

Рис. 3.13. Встановлення та налаштування місячного бюджету

(Розроблено автором)

Для організації бюджету передбачена форма, в якій доступний як ручний ввід сум за категоріями, так і автоматичний розподіл. Також користувачу пропонуються пояснення щодо основних принципів бюджетування, що позитивно впливає на його фінансову грамотність.

Аналітичний розділ «Статистика» реалізовано у вигляді інтерактивних графіків – кругових і стовпчикових діаграм, які дозволяють візуалізувати структуру витрат за категоріями та відстежувати динаміку за обраний період (рис. 3.14).

У верхній частині виводиться короткий аналітичний звіт – загальна сума, середня витрата, топ-категорія та відсоткове порівняння з минулим місяцем. Графіки генеруються автоматично при кожній зміні параметрів фільтрації.

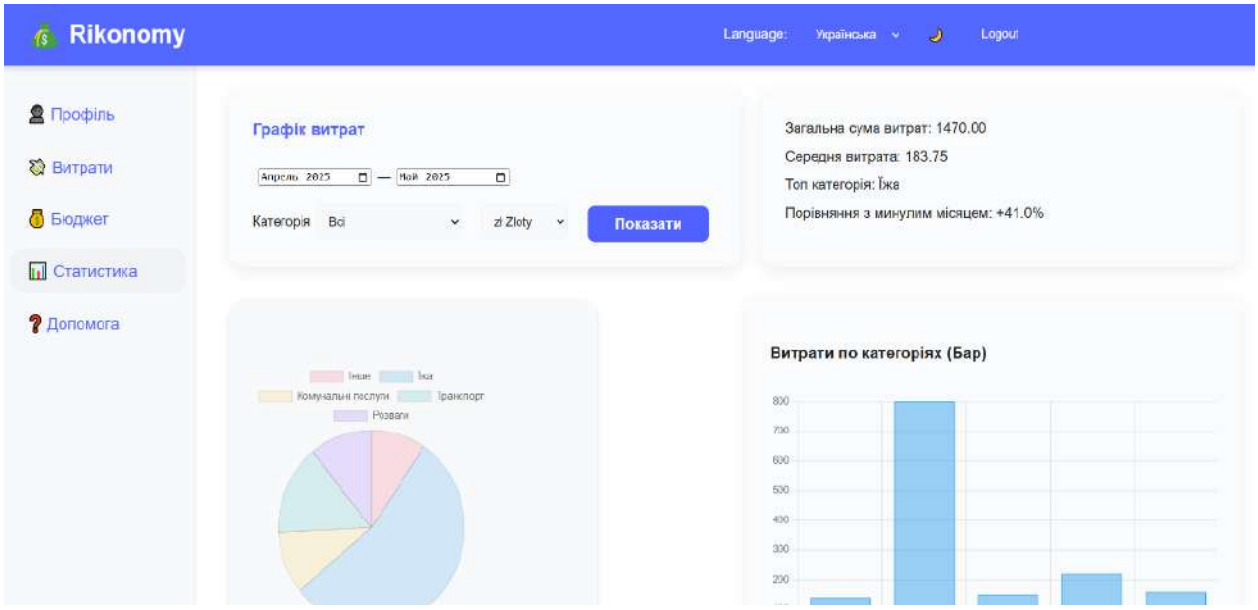

Рис. 3.14. Аналітика витрат у вигляді діаграм

(Розроблено автором)

У верхній частині виводиться короткий аналітичний звіт – загальна сума, середня витрата, топ-категорія та відсоткове порівняння з минулим місяцем. Графіки генеруються автоматично при кожній зміні параметрів фільтрації.

Розділ «Допомога» містить інтерактивні підказки, що відповідають на типові питання щодо використання додатку (рис. 3.15).

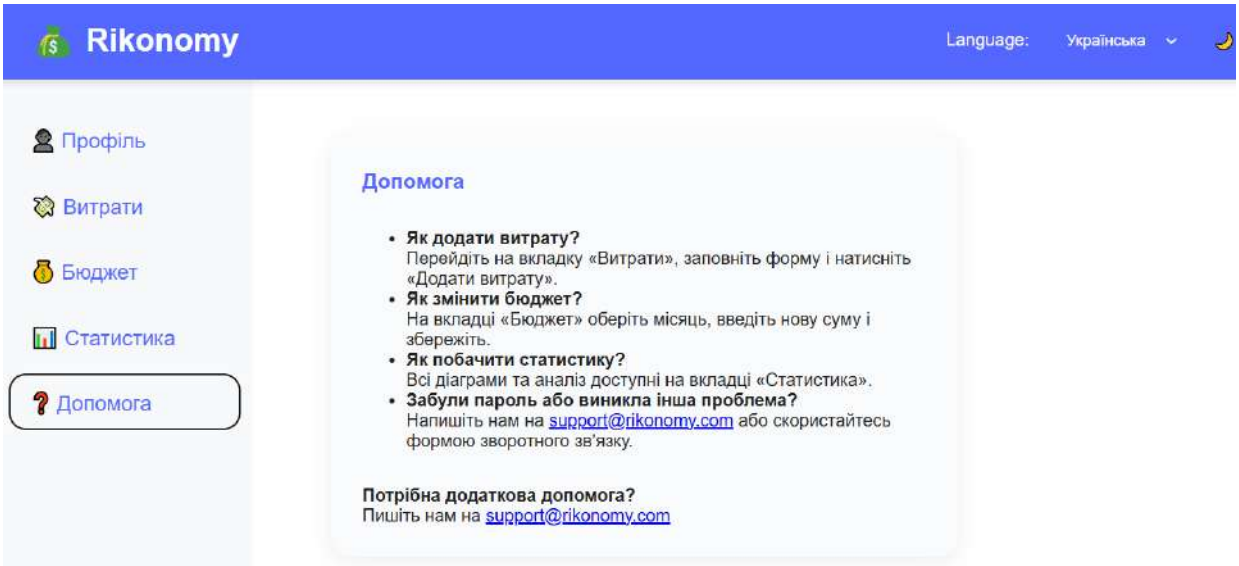

Рис. 3.15. Розділ допомоги з інструкціями для користувача

(Розроблено автором)

Користувач має можливість отримати докладні інструкції щодо додавання витрат, налаштування бюджету та перегляду статистики. У разі виникнення додаткових запитань передбачена опція звернення до розробників через зазначені контактні дані.

У таблиці 3.1 наведено коротку характеристику основних елементів інтерфейсу, реалізованих у Rikonomy.

Таблиця 3.1

Основні елементи інтерфейсу Rikonomy

Елемент	Опис	Призначення
Header	Логотип, перемикач мови і теми, вихід	Глобальні налаштування
Sidebar	Профіль, витрати, бюджет, статистика	Навігація по розділах
Profile	Дані користувача, вибір місяця	Персоналізація, налаштування
Expenses	Форма й список витрат, фільтри	Введення й аналіз витрат
Budget	Встановлення бюджету, підказки	Контроль та планування
Statistics	Графіки та діаграми	Візуалізація й аналітика
Help	FAQ, контакти	Підтримка користувача

(Розроблено автором)

На всіх етапах взаємодії з додатком дані користувача захищені: кожен обліковий запис має окремий набір витрат, бюджетів та налаштувань, які синхронізуються із сервером через захищені запити.

Таким чином, клієнтська частина Rikonomy реалізована відповідно до сучасних стандартів веб-інтерфейсів, що забезпечує як високий рівень зручності, так і гнучкість розширення функціоналу у майбутньому.

3.4 Адаптивні механізми конфігурування інтерфейсу користувача

У сучасних веб-застосунках важливу роль відіграє забезпечення комфортної роботи користувачів незалежно від їхніх мовних вподобань, стилю інтерфейсу та обраної валюти для фінансових розрахунків У розробленому застосунку Rikonomy реалізовано підтримку трьох мов – української, англійської

та польської. Вибір мови здійснюється через інтерактивний перемикач у верхній частині інтерфейсу. Зміна мови відбувається динамічно, без перезавантаження сторінки: всі текстові елементи підтягуються із відповідного словника перекладів. Як видно з рисунку 3.16, зміна мови в інтерфейсі здійснюється миттєво: всі підписи, кнопки, повідомлення та інші текстові елементи оновлюються відповідно до вибраної мови без необхідності перезавантажувати сторінку. Динамічна зміна мови інтерфейсу спрощує роботу з додатком для різних користувачів. Завдяки цьому кожен може отримати повний доступ до всіх функцій системи тією мовою, яка для нього звична. Такий підхід особливо важливий для проєктів, якими користуються люди з різних країн, оскільки дозволяє уникнути мовних бар'єрів. Багатомовне середовище стає зрозумілим і зручним для кожного, що суттєво підвищує рівень задоволеності та мотивацію використовувати програмний продукт надалі.

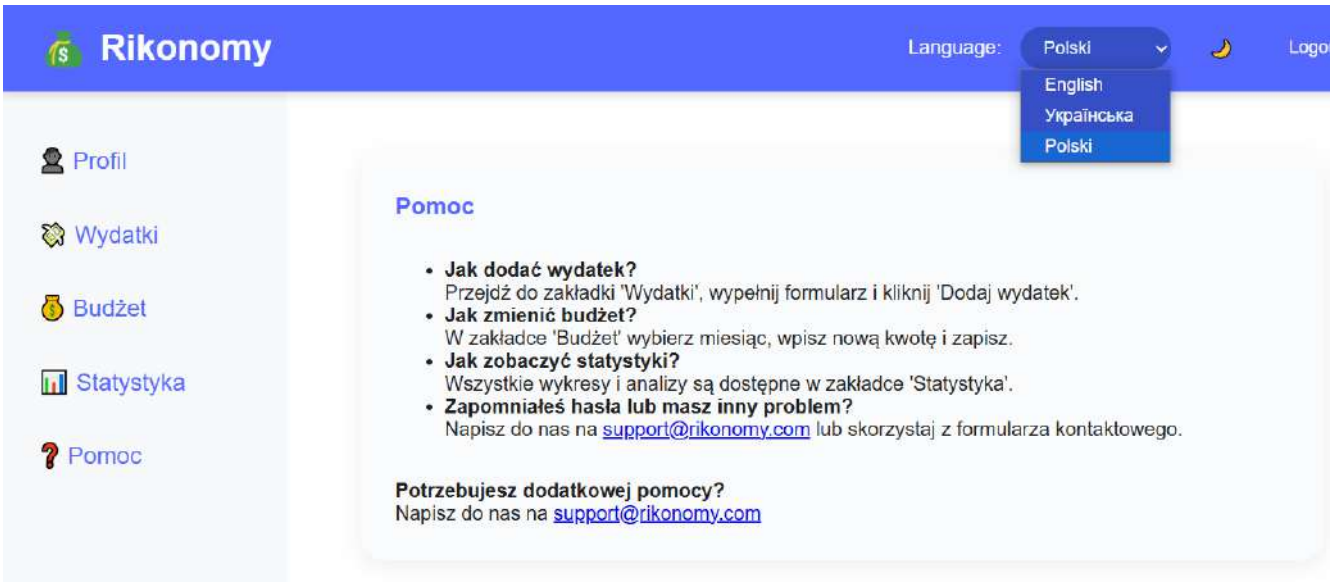

Рис. 3.16. Вигляд інтерфейсу при виборі різних мов

(Розроблено автором)

Механізм багатомовності реалізовано шляхом використання спеціальної структури даних – об’єкта translations, у якому для кожної мови передбачено набір ключів і відповідних текстових значень. Зміна мови ініціює функцію, що автоматично знаходить усі елементи з атрибутом data-translate і підставляє для

них потрібний текст згідно з обраною мовою. Завдяки цьому всі основні елементи інтерфейсу, зокрема повідомлення, кнопки та підказки, миттєво оновлюються відповідно до вибору користувача.

Окрім багатомовності, особливу увагу приділено темам оформлення. У Rikonomy реалізовано світлу та темну тему, які перемикаються через спеціальний контрол у верхньому меню (рис. 3.17). Для цього в CSS використовуються змінні, що дає змогу миттєво змінювати основні кольори інтерфейсу без необхідності перезавантаження сторінки. Зміна теми впливає на всі елементи застосунку, включаючи діаграми, таблиці й повідомлення.

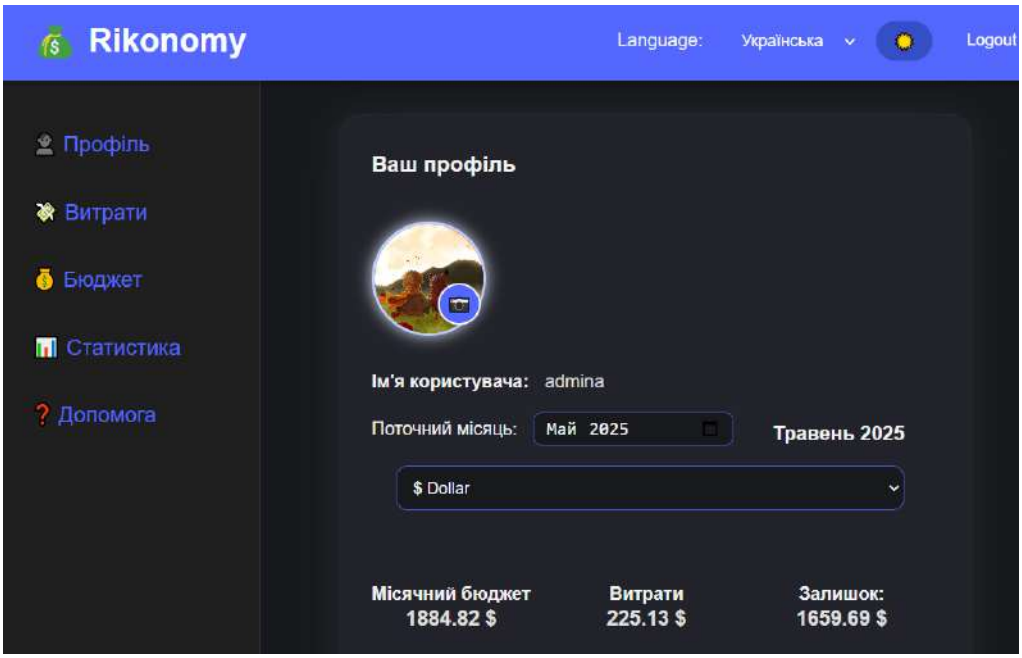

Рис. 3.17. Перемикання між світлою та темною темами

(Розроблено автором)

Зміна теми оформлення здійснюється шляхом додавання або видалення класу dark-theme до елемента <body>, у результаті чого всі кольорові параметри інтерфейсу оновлюються автоматично відповідно до вибраної теми.

Важливою складовою функціоналу є також автоматичний перерахунок валют при обліку витрат. Застосунок дозволяє користувачу обирати основну валюту, після чого всі фінансові дані конвертуються згідно з поточними курсами, які отримуються з зовнішнього API. Перерахунок сум здійснюється динамічно

під час зміни налаштувань профілю щодо валюти, що забезпечує актуальність і коректність фінансової інформації для користувача.

Для зберігання курсів валют використовується окремий об’єкт, який оновлюється при кожному зверненні до API. Якщо дані недоступні, система використовує останні збережені значення, що гарантує коректність обчислень навіть при нестабільному з’єднанні з інтернетом.

Реалізовані функції перемикання мови, вибору теми оформлення та автоматичного перерахунку валют дають змогу кожному користувачу налаштувати систему під власні потреби. Не має значення, з якої країни заходить людина чи які параметри їй зручніше обирати – додаток залишається простим і зрозумілим для всіх. Саме така гнучкість і увага до деталей дозволяють забезпечити позитивний досвід користування незалежно від індивідуальних вподобань чи регіону проживання.

Висновки до розділу 3

У третьому розділі було виконано повний цикл розробки веб-системи обліку особистих фінансів Rikonomy. Описано обґрунтований вибір програмних засобів для реалізації серверної та клієнтської частин, наведено структуру і логіку роботи серверної частини, зосереджено увагу на питаннях безпеки, захисту доступу до даних, а також ефективності обробки запитів і зберігання інформації.

Клієнтська частина проєкту розроблена як односторінковий адаптивний застосунок з інтерактивним інтерфейсом, що забезпечує зручну взаємодію користувача з основними функціями системи – обліком витрат, плануванням бюджету, переглядом аналітики та отриманням довідкової інформації. В роботі продемонстровано реалізацію мультимовності, змін тем оформлення та автоматичного перерахунку валют, що дозволяє кожному користувачу налаштувати додаток під власні потреби і підвищує рівень комфорту при використанні системи.

У результаті вдалося створити систему, якою легко користуватися і яку можна налаштувати під свої потреби. Програма добре працює для людей із різних країн, не вимагає спеціальних знань і дає можливість обирати потрібні параметри. Все це свідчить про те, що вибраний підхід спрацював, і в майбутньому Rikonomy можна буде доповнювати новими можливостями.

ВИСНОВКИ

Проведене дослідження всебічно охопило шлях від виявлення поточних проблем обліку особистих фінансів до створення та апробації веб-системи Rikonomy, що ці проблеми усуває. На аналітичному етапі з’ясовано, що сучасні користувачі стикаються з фрагментованістю платіжних каналів, браком часу на регулярний облік і занепокоєнням щодо конфіденційності даних; огляд існуючих сервісів підтвердив, що вони лише частково розв’язують зазначені завдання і потребують подальшого розвитку. З урахуванням виявлених вимог обґрунтовано вибір класичної клієнт-серверної архітектури з RESTful-API та реляційною СКБД MySQL, що забезпечує чітке розмежування ролей, спрощує масштабування і підвищує безпеку обробки фінансових записів. Структура сховища (таблиці users, expenses, budgets, budget_categories) гарантує референційну цілісність, підтримує транзакційність і дає змогу швидко розширювати модель без втрати вже накопичених даних.

У процесі розробки реалізовано сервер на Node.js / Express з JWT-авторизацією, bcrypt-хешуванням паролів, параметризованими SQL-запитами та CORS-фільтрами, що мінімізує ризики XSS і SQL-ін’єкцій; клієнтська частина побудована на JavaScript із підтримкою Service Worker та локального кешу, завдяки чому інтерфейс завантажується швидко й може працювати офлайн. Впроваджено багатомовність, вибір тем оформлення та динамічний перерахунок валют, що робить систему універсальною для користувачів різних регіонів. Експериментальне навантаження підтвердило стійкість рішення: балансувальник розподіляє трафік між екземплярами серверів, а горизонтальне масштабування бази та API-інтеграцій зберігає стабільність навіть у пікові години; регулярні резервні копії та можливість «гарячих» міграцій збільшують надійність зберігання даних.

Таким чином, у роботі сформульовано й систематизовано вимоги до сучасних інструментів персонального фінансового обліку, запропоновано архітектурну модель, що поєднує безпеку, продуктивність і можливість

подальшого розвитку, а також розроблено й апробовано прототип Rikonomy, який підтвердив життєздатність обраних технічних рішень і повністю задовольнив початкові критерії. Практична цінність полягає у готовому до розгортання веб-застосунку з відкритою структурою бази та документацією, придатному як для інтеграції у фінтех-проєкти, так і для розширення новими модулями (доходи, інвестиції, колективні бюджети). Наукова новизна проявляється у комплексному поєднанні механізмів багатомовності, тематичної персоналізації й мультивалютної підтримки з транзакційною обробкою даних у реальному часі. Подальші дослідження доцільно спрямувати на використання машинного навчання для прогнозу витрат і рекомендацій з оптимізації бюджету, а також на інтеграцію блокчейн-технологій для підвищення прозорості й незмінності фінансових записів.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Python Software Foundation. Django documentation (version 4.2). – 2024. – URL: https://docs.djangoproject.com/en/4.2/ (дата звернення: 23.03.2025).
2. MDN Web Docs. Progressive web apps (PWA). Mozilla, 2023. – URL: https://developer.mozilla.org/en-US/docs/Web/Progressive_web_apps (дата звернення: 23.03.2025).
3. OWASP Foundation. OWASP Top 10 – 2021 (Updated 2022). – URL: https://owasp.org/Top10 (дата звернення: 23.03.2025).
4. European Union Agency for Cybersecurity (ENISA). Threat Landscape for Supply Chain Attacks. – 2023. – URL: https://www.enisa.europa.eu/publications/threat-landscape-for-supply-chain-attacks (дата звернення: 23.03.2025).
5. Jest Documentation. Jest 29.0, 2024. – URL: https://jestjs.io/docs/getting-started (дата звернення: 23.03.2025).
6. Roekhoff, M. C., & van der Cruijsen, C. Paying in a blink of an eye: it hurts less, but you spend more // Journal of Economic Behavior & Organization. – 2024. – Vol. 221. – P. 110–133. – URL: https://www.sciencedirect.com/science/article/pii/S0167268124001100?via%3Dihub (дата звернення: 26.03.2025).
7. Bitrián, P., Buil, I., & Catalán, S. Enhancing user engagement through gamification in personal finance apps: The role of motivation and trust // International Journal of Bank Marketing. – 2021. – Vol. 39(7). – P. 1310–1324. – URL: https://doi.org/10.1108/IJBM-07-2020-0351 (дата звернення: 25.05.2025).
8. Netherlands Bank. Data privacy risks in mobile finance apps: a systemic review. DNB Working Paper No. 794. – 2022. – URL: https://www.dnb.nl/media/hwpei2bx/working_paper_no-794.pdf (дата звернення: 23.03.2025).
9. PostgreSQL Global Development Group. PostgreSQL 15. Documentation. – 2023. – URL: https://www.postgresql.org/docs/ (дата звернення: 23.03.2025).

10. Suprun A., Petrishina T., Vasylchuk I. Competition and cooperation between fintech companies and traditional financial institutions // The International Conference on Sustainable Futures: Environmental, Technological, Social and Economic Matters (ICSF 2020), Kryvyi Rih, Ukraine, Volume 166 (2020). – URL: https://doi.org/10.1051/e3sconf/202016601328 (дата звернення: 23.03.2025).
11. European Commission. Digital Finance: Emerging Risks in Crypto-Assets and Digital Finance. – 2022. – URL: https://finance.ec.europa.eu/publications/digital-finance-emerging-risks-crypto-assets-and-digital-finance_en (дата звернення: 23.03.2025).
12. Python Software Foundation. Django documentation (version 4.2). – 2024. – URL: https://docs.djangoproject.com/en/4.2/ (дата звернення: 23.03.2025).1
13. Konoplytskyi I.O., Kravchenko K.O. Web-programuvannia: pidruchnyk. – Kyiv: KNEU, 2021. – 412 s.
14. Gordienko D.M., Zhukov S.V. Osnovy proektuvannia informatsiinykh system. – Kyiv: Tsentr uchebnoi literatury, 2020. – 264 s.
15. HTML Living Standard. WHATWG, 2024. – URL: https://html.spec.whatwg.org/ (дата звернення: 23.03.2025).
16. State Statistics Service of Ukraine. Financial and economic indicators of the population. – 2023. – URL: https://ukrstat.gov.ua/ (дата звернення: 23.03.2025).
17. EBA Annual Report 2023. European Banking Authority. – 2023. – URL: https://www.eba.europa.eu/ (дата звернення: 23.03.2025).

ЗГОДА здобувачки вищої освіти

Державного університету економіки і технологій про перевірку кваліфікаційної роботи на прояви академічного плагіату

та розміщення в Репозитарії Університету

Я, **Коліогло Катерина Вячеславівна** (ПІП),

підтримую політику Державного університету економіки і технологій з академічної доброчесності і відкритого доступу.

Засвідчую, що кваліфікаційна бакалаврська робота

Розробка веб-застосунку для управління фінансами

(назва роботи повністю) виконана самостійно та не містить академічного плагіату. Я не надавав(ла) і не одержував(ла) недозволену допомогу під час підготовки цієї роботи. Робота містить результати власних досліджень. Використання ідей, результатів і текстів інших авторів мають посилання на відповідне джерело.

Із чинним Положенням про запобігання та виявлення академічного плагіату в роботах здобувачів вищої освіти Державного університету економіки і технологій ознайомлений(а). Чітко усвідомлюю, що в разі виявлення у кваліфікаційній роботі порушення норм академічної доброчесності робота не допускається до захисту або оцінюється незадовільно.

Також я поінформований(на), що відповідно до «Положення про Репозитарій (електронну базу даних) Державного університету економіки і технологій» зазначена робота буде розміщена в Електронному архіві Університету (Репозитарії ДУЕТ). З умовами такого розміщення ознайомлений(на).

01.06.2025 р. К.В. Коліогло

Дата підпис ініціали, прізвище