

**Міністерство освіти і науки України
Луцький національний технічний університет
Факультет комп'ютерних та інформаційних технологій
Кафедра комп'ютерних наук**

**КВАЛІФІКАЦІЙНА РОБОТА
ЗА СТУПЕНЕМ ВИЩОЇ ОСВІТИ «БАКАЛАВР»**

**РОЗРОБКА ВЕБ-СЕРВІСУ ОБЛІКУ ТА АНАЛІЗУ ФІНАНСОВИХ
ОПЕРАЦІЙ**

**DEVELOPMENT OF A WEB SERVICE FOR ACCOUNTING AND
ANALYSIS OF FINANCIAL TRANSACTIONS**

спеціальність 122 Комп'ютерні науки
освітня програма «Комп'ютерні науки»

Виконав: здобувач вищої освіти
групи КН-42
Ломака Дмитро Максимович

(підпис)

Керівник: д. пед. н., професор
Тулашвілі Юрій Йосипович

(підпис)

Кваліфікаційну роботу
допущено до захисту
«__» _____ 2026 р.
Гарант освітньої програми:
д.пед.н., професор
Тулашвілі Юрій Йосипович

(підпис)

Луцьк – 2026 року

ЛУЦЬКИЙ НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ

Факультет комп'ютерних та інформаційних технологій

Кафедра комп'ютерних наук

Ступінь вищої освіти: бакалавр

Галузь знань: 12 Інформаційні технології

Спеціальність: 122 Комп'ютерні науки

Освітня програма: «Комп'ютерні науки»

ЗАТВЕРДЖУЮ

Завідувач кафедри

_____ Валерій ЛІЩИНА

«16» січня 2026 р.

ЗАВДАННЯ НА КВАЛІФІКАЦІЙНУ РОБОТУ ЗДОБУВАЧА ПЕРШОГО (БАКАЛАВРСЬКОГО) РІВНЯ ВИЩОЇ ОСВІТИ

Ломака Дмитро Максимович

(прізвище, ім'я, по батькові)

1. Тема кваліфікаційної роботи «Розробка веб-сервісу обліку та аналізу фінансових операцій».

Керівник д.пед.н., професор Тулашвілі Юрій Йосипович

затверджені наказом закладу вищої освіти від «16» січня 2026 р. № 32/01-02.

2. Строк подання здобувачем вищої освіти кваліфікаційної роботи «02» червня 2026 р.

3. Вихідні дані до роботи: Технічна документація React.js 18, Vite, Tailwind CSS, Recharts; специфікація формату JSON (ECMA-404); стандарт Web Crypto API (W3C); специфікація Web Storage API (WHATWG HTML Living Standard); алгоритм хешування SHA-256 (FIPS PUB 180-4); результати аналізу існуючих аналогів (Spendee, Money Manager, Wallet by BudgetBakers); вимоги до функціоналу обліку фінансових операцій.

4. Зміст розрахунково-пояснювальної записки: Аналіз сучасного стану проблеми, існуючих методів і засобів її розв'язання, аналіз і вибір засобів проектування, опис функціонального наповнення об'єкта проектування, розробка й обґрунтування системного наповнення, функціонально-структурна схема роботи об'єкта проектування.

5. Перелік графічного матеріалу: 8 рисунків, 7 таблиць, презентація.

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис	
		завдання видав	завдання прийняв
<i>Аналіз проблеми за темою роботи та постановка завдань дослідження</i>	<i>Тулашвілі Ю. Й.</i>		
<i>Теоретичне дослідження</i>	<i>Тулашвілі Ю. Й.</i>		
<i>Практична реалізація</i>	<i>Тулашвілі Ю. Й.</i>		
<i>Спеціальні розрахунки</i>	<i>Тулашвілі Ю. Й.</i>		
<i>Показник запозичень тексту</i>	_____ %		
<i>Інструментальна перевірка</i>	<i>Кошелюк В. А.</i>		
<i>Нормоконтроль</i>	<i>Сачук В. О.</i>		
<i>Гарант ОП</i>	<i>Тулашвілі Ю. Й.</i>		

7. Дата видачі завдання «16» січня 2026 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів кваліфікаційної роботи бакалавра	Строк виконання етапів роботи	Примітка
1	<i>Провести огляд літературних джерел по темі кваліфікаційної роботи</i>	<i>до 17.02.2026 р</i>	
2	<i>Провести аналіз загальної проблеми і вибір напрямків дослідження</i>	<i>до 28.02.2026 р.</i>	
3	<i>Розробити функціональну схему роботи програмного продукту</i>	<i>до 12.03.2026 р</i>	
4	<i>Описати засоби розробки об'єкта проектування</i>	<i>до 26.03.2026 р.</i>	
5	<i>Практична реалізація об'єкта проектування</i>	<i>до 30.04.2026 р.</i>	
6	<i>Провести спеціальні розрахунки</i>	<i>до 18.05.2026 р.</i>	
7	<i>Здача чистового варіанту кваліфікаційної роботи бакалавра на кафедрі</i>	<i>до 02.06.2026 р.</i>	

Здобувач вищої освіти _____ Дмитро ЛОМАКА

Керівник роботи _____ Юрій ТУЛАШВІЛІ

АНОТАЦІЯ

Ломака Д. М. Розробка веб-сервісу обліку та аналізу фінансових операцій.
Рукопис.

Кваліфікаційна робота бакалавра ОП «Комп'ютерні науки». Луцький національний технічний університет. Луцьк, 2026.

Кваліфікаційна робота бакалавра складається зі вступу, чотирьох розділів, висновків та списку використаних джерел, присвячена розробці веб-сервісу обліку та аналізу фінансових операцій. У роботі проведено аналіз предметної області персонального фінансового обліку та існуючих аналогічних систем (Spendee, Money Manager, Wallet by BudgetBakers), обґрунтовано вибір архітектури SPA та технологічного стеку розробки. Розроблено клієнтський інтерфейс на основі React.js 18 з використанням Tailwind CSS та інструменту збірки Vite, спроектовано інфологічну, логічну та фізичну моделі даних на основі localStorage-сховища браузера. Реалізовано компонентну архітектуру застосунку, повний цикл CRUD-операцій над фінансовими транзакціями, алгоритм багатокритеріальної фільтрації за типом, категорією та часовим діапазоном, а також аналітичні діаграми засобами бібліотеки Recharts. Реалізовано механізм автентифікації користувача з хешуванням пароля за алгоритмом SHA-256 засобами Web Crypto API. Проведено функціональне тестування системи у браузерах Chrome, Firefox та Edge за 16 тест-кейсами, визначено мінімальні системні вимоги. Проаналізовано загрози інформаційній безпеці клієнтського веб-застосунку та реалізовано комплекс заходів захисту.

Ключові слова: веб-сервіс, фінансові операції, персональний фінансовий облік, React.js, SPA, Tailwind CSS, Web Crypto API, компонентна архітектура, localStorage, SHA-256

ANNOTATION

Lomaka Dmytro. Development of a web service for accounting and analysis of financial transactions. Manuscript.

Bachelor's qualification paper in the study program «Computer Science». Lutsk National Technical University. Lutsk, 2026.

This bachelor's thesis consists of an introduction, four chapters, conclusions and a list of references, and is dedicated to the development of a web service for the recording and analysis of financial transactions. The thesis analyses the subject area of personal financial accounting and existing similar systems (Spendee, Money Manager, Wallet by BudgetBakers), and justifies the choice of SPA architecture and development technology stack. A client interface has been developed based on React.js 18 using Tailwind CSS and the Vite build tool; information, logical and physical data models have been designed based on the browser's localStorage. A component-based application architecture was implemented, along with a full cycle of CRUD operations on financial transactions, a multi-criteria filtering algorithm by type, category and time range, as well as analytical charts using the Recharts library. A user authentication mechanism with password hashing using the SHA-256 algorithm via the Web Crypto API has been implemented. Functional testing of the system was carried out in Chrome, Firefox and Edge browsers using 16 test cases, and the minimum system requirements were determined. Information security threats to the client-side web application have been analysed and a set of protective measures has been implemented.

Keywords: web service, financial transactions, personal financial accounting, React.js, SPA, Tailwind CSS, Web Crypto API, component-based architecture, localStorage, SHA-256

ЗМІСТ

ВСТУП	7
РОЗДІЛ 1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ.....	9
1.1 Аналіз сучасного стану проблеми.....	9
1.2 Аналіз аналогічних програм, баз даних, систем, обґрунтування вибраного напрямку та вибір методів рішення	10
1.3 Аналіз і вибір засобів проектування веб-сервісу обліку та аналізу фінансових операцій.....	13
1.4 Постановка завдання на кваліфікаційну роботу бакалавра	17
РОЗДІЛ 2 СПЕЦИФІКАЦІЯ ВИМОГ ДО РОЗРОБЛЮВАНОЇ СИСТЕМИ.....	18
2.1 Обґрунтування вибору шляхів, технологій (алгоритмів) і засобів вирішення поставленого завдання.....	18
2.2 Функціонально-структурна схема роботи об'єкта проектування (блок-схеми алгоритмів, UML діаграми класів, діаграми компонентів).....	22
2.3 Проектування структури даних та технологія доступу до сховища.....	27
РОЗДІЛ 3 РОЗРОБКА ВЕБ-СЕРВІСУ ОБЛІКУ ТА АНАЛІЗУ ФІНАНСОВИХ ОПЕРАЦІЙ	31
3.1 Практична реалізація об'єкта проектування	31
3.2 Тестування та налагодження веб-сервісу обліку та аналізу фінансових операцій.....	39
3.3 Інструкція користувача з інсталяції та використання програмного продукту	42
РОЗДІЛ 4 СПЕЦІАЛЬНІ РОЗРАХУНКИ.....	45
ВИСНОВКИ	51
СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ.....	53

ВСТУП

Управління особистими фінансами є одним із ключових аспектів фінансової грамотності населення. За даними досліджень у цій галузі, значна частина людей не веде систематичного обліку власних доходів і витрат, що призводить до неефективного розподілу бюджету та накопичення боргових зобов'язань. Існуючі програмні рішення у сфері персонального фінансового обліку здебільшого орієнтовані на корпоративний сектор, є надмірно складними для індивідуального використання або передбачають обов'язкову передачу персональних фінансових даних стороннім серверам. Прогалиною у цій предметній галузі залишається відсутність простих, безкоштовних і конфіденційних клієнтських рішень, доступних без реєстрації та серверної інфраструктури.

Світові тенденції розвитку веб-технологій свідчать про стрімке зростання можливостей браузерного середовища як платформи для повноцінних застосунків. Поява стандартів Web Crypto API, Web Storage API та прогресивних JavaScript-фреймворків зробила можливою реалізацію безпечних і функціональних застосунків виключно на стороні клієнта без залучення серверної інфраструктури. Архітектура Single Page Application (SPA) стала домінуючим підходом у розробці сучасних веб-інтерфейсів завдяки високій швидкості відгуку та зручності використання.

Актуальність даної роботи обумовлена необхідністю створення доступного веб-сервісу обліку фінансових операцій, що забезпечує повну конфіденційність персональних даних, не потребує реєстрації та підключення до мережі, і при цьому надає користувачу зручний інструмент аналізу власного бюджету.

Метою кваліфікаційної роботи є проектування та реалізація клієнтського веб-сервісу обліку та аналізу фінансових операцій із забезпеченням захисту облікових даних користувача засобами Web Crypto API.

Для досягнення поставленої мети вирішено такі завдання:

- проведено аналіз предметної області та існуючих програмних аналогів;
- обґрунтовано вибір архітектурних і технологічних рішень;
- розроблено специфікацію вимог та модель даних системи;
- реалізовано клієнтський SPA-застосунок із повним набором CRUD-операцій, фільтрацією та аналітичними діаграмами;
- реалізовано механізм автентифікації з хешуванням пароля за алгоритмом SHA-256;
- проведено функціональне тестування та визначено системні вимоги;
- проаналізовано загрози інформаційній безпеці та реалізовано відповідні заходи захисту.

Об'єктом розробки є процес автоматизації персонального фінансового обліку засобами клієнтських веб-технологій.

Предметом розробки є архітектурні та технологічні рішення побудови SPA-застосунку на основі React.js 18, Vite, Tailwind CSS та Recharts із локальним зберіганням даних у localStorage браузера.

Новизна роботи полягає у комплексному застосуванні браузерних стандартів Web Crypto API та Web Storage API для реалізації захищеного персонального фінансового трекера без серверної складової, що забезпечує конфіденційність даних на архітектурному рівні – шляхом виключення їх передачі мережею.

Практична цінність роботи полягає у створенні готового до розгортання веб-застосунку, що може використовуватись без підключення до мережі, не потребує серверної інфраструктури та забезпечує користувачу повний контроль над власними фінансовими даними. Застосунок може бути розгорнутий на будь-якому статичному хостингу або використовуватись локально.

РОЗДІЛ 1

АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Аналіз сучасного стану проблеми

В умовах зростаючої цифровізації фінансових відносин питання автоматизації обліку та аналізу особистих фінансів набуває дедалі більшої актуальності. За даними досліджень у галузі фінансової грамотності, значна частина населення не веде систематичного обліку власних доходів і витрат, що призводить до неефективного розподілу бюджету та накопичення боргових зобов'язань. Вирішення цієї проблеми шляхом впровадження спеціалізованих програмних засобів є одним із пріоритетних напрямів розвитку персональних інформаційних технологій.

Сучасний ринок програмного забезпечення пропонує широкий спектр інструментів для управління особистими фінансами – від мобільних застосунків до повноцінних веб-платформ. Проте більшість із них або є надмірно складними для пересічного користувача, або передбачають платну підписку, або вимагають обов'язкової реєстрації з передачею персональних фінансових даних стороннім сервісам. Це формує запит на легкі, безпечні та доступні рішення, що працюють безпосередньо у браузері без необхідності серверної інфраструктури.

Об'єктом проектування є веб-сервіс обліку та аналізу фінансових операцій, призначений для фізичних осіб. Сервіс забезпечує введення, зберігання та категоризацію фінансових операцій (доходів і витрат), їх фільтрацію за датою та категорією, а також візуальне представлення аналітики у вигляді діаграм. Зберігання даних реалізується засобами браузера – через механізм `localStorage`, що виключає необхідність серверної частини та забезпечує конфіденційність даних користувача [16].

Предметна область системи охоплює такі інформаційні сутності: фінансова операція (транзакція), категорія операції, тип операції (дохід або витрата), часовий період, а також зведені аналітичні показники (баланс, сума витрат за категорією, динаміка за `period`). Функціональний опис вихідного

об'єкту передбачає такі операції: додавання нової транзакції із зазначенням суми, типу, категорії та дати; редагування та видалення існуючих записів; перегляд загального балансу та аналітичних звітів; фільтрацію операцій за заданими критеріями.

Доцільність створення нової системи обумовлена сукупністю чинників. По-перше, існуючі аналоги не забезпечують повноцінної роботи без підключення до мережі та серверної авторизації. По-друге, більшість рішень орієнтовані на корпоративний сектор і є надлишково складними для індивідуального використання. По-третє, розробка власного веб-сервісу дозволяє адаптувати функціональність під конкретні потреби користувача та забезпечити повний контроль над особистими фінансовими даними. Таким чином, розробка легкого, клієнтського SPA-застосунку для обліку фінансових операцій є актуальною та практично значущою задачею.

1.2 Аналіз аналогічних програм, баз даних, систем, обґрунтування вибраного напрямку та вибір методів рішення

Для обґрунтування доцільності розробки власного веб-сервісу було проведено аналіз існуючих програмних рішень у сфері обліку особистих фінансів. Розглянуто три найбільш поширені системи: Spendee, Money Manager та Wallet by BudgetBakers.

Spendee – мобільний та веб-застосунок для управління особистими фінансами, що надає можливість введення транзакцій, категоризації витрат, побудови звітів та синхронізації з банківськими рахунками. Інтерфейс відрізняється сучасним дизайном та зручною навігацією. Проте безкоштовна версія суттєво обмежена за функціональністю: доступний лише один гаманець, відсутня можливість підключення банківських рахунків та експорту даних. Повний функціонал передбачає платну підписку від 2,99 USD на місяць. Крім того, усі дані зберігаються на серверах компанії, що викликає питання конфіденційності персональних фінансових відомостей.

Money Manager – застосунок для обліку доходів і витрат з підтримкою подвійного запису, бюджетування та формування детальних звітів. Відрізняється розширеними аналітичними можливостями та підтримкою кількох валют. Основним недоліком є орієнтація переважно на мобільні платформи (Android, iOS) – повноцінна веб-версія відсутня. Інтерфейс є перевантаженим для користувачів, яким потрібен лише базовий облік, а крива навчання є відчутною для нових користувачів.

Wallet by BudgetBakers – комплексна платформа для управління особистими та сімейними фінансами з підтримкою автоматичного імпорту транзакцій із банків, спільного доступу та детального бюджетування. Має веб-версію та мобільні додатки. Недоліками є висока складність налаштування, обов’язкова реєстрація облікового запису, а також те, що більшість ключових функцій (зокрема підключення банків та спільний доступ) доступні виключно у платному тарифі від 3,49 USD на місяць.

Порівняльний аналіз розглянутих систем наведено у таблиці 1.1.

Таблиця 1.1 – Порівняльний аналіз аналогів

Критерій	Spendee	Money Manager	Wallet	Розроблюваний сервіс
Веб-версія	Так	Ні	Так	Так
Безкоштовність	Частково	Частково	Частково	Повністю
Локальне зберігання	Ні	Ні	Ні	Так
Реєстрація	Обов’язкова	Обов’язкова	Обов’язкова	Не потрібна
Простота інтерфейсу	Середня	Низька	Низька	Висока
Робота офлайн	Ні	Так	Ні	Так

Як видно з таблиці 1.1, жоден із розглянутих аналогів не забезпечує одночасно безкоштовного доступу до повного функціоналу, локального зберігання даних без передачі на сторонні сервери та простоти використання у браузері без обов’язкової реєстрації.

На підставі проведеного аналізу було визначено ключові вимоги до розроблюваного сервісу: повна безкоштовність, відсутність обов’язкової

реєстрації, локальне зберігання даних без їх передачі на сторонні сервери, робота у браузері без встановлення додаткового програмного забезпечення, а також простий та інтуїтивно зрозумілий інтерфейс.

З урахуванням зазначених вимог обрано напрям розробки клієнтського односторінкового застосунку (SPA – Single Page Application). Архітектура SPA передбачає завантаження єдиного HTML-документу при першому зверненні до сервісу з подальшим динамічним оновленням вмісту сторінки засобами JavaScript без перезавантаження браузера. Така архітектура забезпечує швидку реакцію інтерфейсу на дії користувача, характерну для нативних застосунків, та знижує навантаження на мережу, оскільки між клієнтом і сервером передаються лише дані, а не розмітка сторінки. Для фінансового трекера, де користувач часто виконує повторювані дії – додає транзакції, перемикається між категоріями, переглядає фільтровані списки – це є суттєвою перевагою порівняно з традиційною багатосторінковою архітектурою.

Для зберігання даних обрано механізм `localStorage` браузера. Цей підхід обумовлений кількома міркуваннями. По-перше, він повністю виключає необхідність серверної інфраструктури, що спрощує розгортання та підтримку застосунку. По-друге, всі фінансові дані залишаються виключно на пристрої користувача, що є критично важливим з точки зору конфіденційності. По-третє, `localStorage` забезпечує персистентність даних між сесіями браузера без необхідності автентифікації. Обсяг сховища (до 5-10 МБ залежно від браузера) є достатнім для зберігання тисяч фінансових транзакцій у форматі JSON. Дані структуруються у вигляді масиву об'єктів-транзакцій із фіксованим набором полів, що забезпечує передбачуваність структури та спрощує алгоритми вибірки і фільтрації.

Методом реалізації інтерфейсу обрано компонентний підхід на основі бібліотеки `React.js`. Компонентна архітектура передбачає декомпозицію інтерфейсу на незалежні, повторно використовувані модулі – компоненти, кожен з яких відповідає за власну частину відображення та логіку. Це забезпечує чітке

розмежування відповідальностей між частинами застосунку, спрощує підтримку та масштабування коду.

Управління станом застосунку реалізується засобами вбудованих хуків React (useState, useEffect), що є достатнім для застосунку даного рівня складності та виключає необхідність підключення зовнішніх бібліотек управління станом на кшталт Redux.

Для візуалізації аналітичних даних обрано метод побудови інтерактивних діаграм засобами спеціалізованої бібліотеки, що інтегрується з React-компонентами. Такий підхід дозволяє відображати структуру витрат за категоріями та динаміку балансу у наочній формі, що суттєво підвищує інформативність сервісу порівняно з табличним представленням даних [8].

1.3 Аналіз і вибір засобів проектування веб-сервісу обліку та аналізу фінансових операцій

Вибір інструментів розробки є одним із ключових етапів проектування програмного продукту, оскільки безпосередньо впливає на якість, продуктивність та підтримуваність системи. З урахуванням вимог, сформульованих у попередніх пунктах, було проведено аналіз існуючих засобів розробки клієнтських веб-застосунків.

Мова програмування та середовище виконання. Для реалізації клієнтської частини застосунку використовується мова JavaScript – стандартна мова програмування для веб-браузерів, що підтримується усіма сучасними платформами без необхідності встановлення додаткового середовища виконання. Як надмножину JavaScript обрано TypeScript розглядати не було доцільним, оскільки для застосунку даного масштабу статична типізація не дає суттєвих переваг, але збільшує поріг входу та час розробки.

Бібліотека побудови інтерфейсу. Для реалізації компонентного інтерфейсу розглядалися три основні рішення: React.js, Vue.js та Angular.

React.js – бібліотека з відкритим кодом, розроблена та підтримувана

компанією Meta. Використовує декларативний підхід до побудови інтерфейсу на основі віртуального DOM, що забезпечує ефективне оновлення відображення при зміні стану. React має найбільшу екосистему серед фронтенд-інструментів, широку спільноту розробників та детальну документацію [9].

Vue.js – прогресивний фреймворк з нижчим порогом входу порівняно з React, проте меншою екосистемою та меншою поширеністю на ринку праці.

Angular – повноцінний фреймворк від Google з вбудованими засобами маршрутизації, управління станом та DI-контейнером. Є надлишково складним для застосунків даного рівня та вимагає знання TypeScript.

За результатами порівняння для розробки обрано React.js 18 як найбільш зрілу, широко підтримувану та функціонально достатню бібліотеку для реалізації SPA даного рівня складності.

Інструмент збірки. Розглядалися Webpack та Vite.

Webpack є класичним інструментом збірки з широкими можливостями налаштування, проте характеризується відносно повільним часом запуску dev-сервера через необхідність повного бандлінгу при старті.

Vite – сучасний інструмент збірки, що використовує нативні ES-модулі браузера у режимі розробки, що забезпечує миттєвий запуск сервера незалежно від розміру проєкту. Для виробничої збірки Vite використовує Rollup, що генерує оптимізовані бандли. За результатами порівняння обрано Vite як інструмент, що забезпечує значно вищу швидкість розробки [12].

Засоби стилізації. Розглядалися CSS Modules, styled-components та Tailwind CSS.

CSS Modules забезпечують локальну область видимості стилів, проте вимагають написання значного обсягу власного CSS-коду. Styled-components реалізують підхід CSS-in-JS, що зручно для динамічних стилів, але збільшує розмір бандлу.

Tailwind CSS – утилітарний CSS-фреймворк, що надає готовий набір класів для стилізації безпосередньо у розмітці компонентів. Такий підхід прискорює розробку інтерфейсу, забезпечує консистентність дизайну та не потребує

перемикання між файлами розмітки та стилів. Для даного проєкту обрано Tailwind CSS [11].

Бібліотека візуалізації даних. Для побудови аналітичних діаграм та графічного представлення фінансової інформації розглядалися бібліотеки Chart.js та Recharts.

Chart.js є популярною універсальною JavaScript-бібліотекою, яка підтримує широкий набір типів графіків і характеризується високою продуктивністю. Проте для використання в React-застосунках вона потребує додаткових обгортки або спеціальних інтеграційних компонентів, що ускладнює розробку та супровід інтерфейсу.

Бібліотека Recharts розроблена спеціально для React і базується на компонентному підході. Вона використовує SVG-рендеринг та дозволяє описувати структуру діаграм за допомогою JSX-синтаксису, що забезпечує високу читабельність коду та спрощує інтеграцію з іншими компонентами застосунку. Крім того, Recharts підтримує адаптивність графіків, налаштування зовнішнього вигляду та інтерактивні елементи, такі як підказки й легенди [10; 15].

З огляду на повну сумісність із React та зручність розробки, для реалізації візуалізації даних у проєкті було обрано бібліотеку Recharts.

Сховище даних. Як зазначалося у пункті 1.2, для зберігання інформації про фінансові транзакції обрано механізм localStorage браузера. Даний підхід дозволяє зберігати дані локально на пристрої користувача без необхідності розгортання серверної інфраструктури або використання зовнішньої бази даних. Основними перевагами localStorage є простота реалізації, підтримка в усіх сучасних браузерах, достатній обсяг пам'яті для потреб персонального фінансового трекера та можливість швидкого доступу до даних.

Як альтернативний варіант розглядалася технологія IndexedDB, яка являє собою вбудовану об'єктно-орієнтовану систему зберігання даних у браузері. Вона підтримує роботу з великими обсягами інформації, транзакціями, індексами та складними запитамі. Однак використання IndexedDB

супроводжується значно складнішим програмним інтерфейсом та потребує додаткових витрат часу на розробку. Оскільки фінансовий трекер орієнтований на зберігання відносно невеликого обсягу текстових даних і не передбачає виконання складних операцій над ними, застосування IndexedDB є недоцільним. Тому для даного проєкту було обрано localStorage як більш просте та ефективне рішення [4].

Узагальнені результати аналізу та вибору програмних засобів проєктування наведено в таблиці 1.2.

Таблиця 1.2 – Вибрані засоби проєктування

Складова	Розглянуті варіанти	Обраний засіб	Обґрунтування вибору
Мова програмування	JavaScript, TypeScript	JavaScript	Достатній для даного масштабу, нижчий поріг входу
Бібліотека UI	React.js, Vue.js, Angular	React.js 18	Зріла екосистема, компонентний підхід, широка підтримка
Інструмент збірки	Webpack, Vite	Vite	Швидкий dev-сервер, оптимізована збірка
Стилізація	CSS Modules, styled-components, Tailwind CSS	Tailwind CSS	Прискорення розробки, консистентність дизайну
Візуалізація	Chart.js, Recharts	Recharts	Нативна React-інтеграція, SVG-рендеринг
Сховище даних	localStorage, IndexedDB	localStorage	Простий API, достатній обсяг, персистентність між сесіями

Апаратні вимоги до розробки визначаються потребами обраного технологічного стеку. Для розробки достатньо персонального комп'ютера з процесором із тактовою частотою від 1,6 ГГц, оперативною пам'яттю обсягом не менше 4 ГБ та встановленим середовищем виконання Node.js версії 18 або вище. Для кінцевого користувача вимоги є мінімальними: будь-який сучасний браузер (Chrome 90+, Firefox 88+, Edge 90+, Safari 14+) без необхідності встановлення додаткового програмного забезпечення.

1.4 Постановка завдання на кваліфікаційну роботу бакалавра

Проведений аналіз предметної області, існуючих аналогів і технологічних засобів дав змогу сформулювати основні цілі кваліфікаційної роботи, спрямовані на розробку функціонального веб-сервісу для обліку та аналізу фінансових операцій.

- провести аналіз предметної області персонального фінансового обліку, дослідити існуючі програмні аналоги та обґрунтувати вибір архітектурних і технологічних рішень для розроблюваного веб-сервісу;

- розробити специфікацію вимог до системи, визначити її функціональні характеристики, побудувати діаграми варіантів використання, компонентів та структури даних для формального опису проєктованого застосунку;

- спроектувати модель даних веб-сервісу на основі локального JSON-сховища браузера, визначити структуру сутностей, їх атрибути та взаємозв'язки, розробити інфологічну та фізичну моделі даних;

- реалізувати клієнтський SPA-застосунок засобами React.js 18 та Vite з компонентною архітектурою, що забезпечує виконання повного циклу CRUD-операцій над фінансовими транзакціями, їх категоризацію, фільтрацію за датою та категорією, а також відображення аналітичних діаграм засобами бібліотеки Recharts;

- реалізувати механізм авторизації користувача з використанням localStorage та забезпечити базовий захист даних від несанкціонованого доступу в межах браузерного середовища;

- провести тестування розробленого веб-сервісу на відповідність функціональним вимогам, перевірити коректність роботи у різних браузерах та сформулювати мінімальні системні вимоги для розгортання і використання застосунку;

- виконати спеціальні розрахунки відповідно до обраного напрямку четвертого розділу та підготувати повний комплект технічної документації до кваліфікаційної роботи.

РОЗДІЛ 2

СПЕЦИФІКАЦІЯ ВИМОГ ДО РОЗРОБЛЮВАНОЇ СИСТЕМИ

2.1 Обґрунтування вибору шляхів, технологій (алгоритмів) і засобів вирішення поставленого завдання

Вибір архітектурних і технологічних рішень для реалізації веб-сервісу обліку та аналізу фінансових операцій ґрунтується на результатах аналізу предметної області, огляду існуючих аналогів та сформульованих функціональних вимог, наведених у першому розділі. Основним критерієм вибору було забезпечення достатньої функціональності персонального фінансового трекера за мінімальної складності інфраструктури, високої швидкодії інтерфейсу та збереження конфіденційності користувацьких даних. У цьому пункті подано мотивовану оцінку прийнятої схеми побудови системи з науково-технічного погляду.

Архітектурне рішення. Для реалізації застосунку обрано архітектуру Single Page Application (SPA). Порівняно з традиційною багатосторінковою архітектурою (MPA – Multi Page Application), SPA забезпечує вищу швидкість відгуку інтерфейсу, оскільки після початкового завантаження ресурсів взаємодія з користувачем відбувається без повного перезавантаження сторінки. При кожній дії користувача оновлюється лише та частина DOM, яка безпосередньо змінилася, що досягається завдяки механізму віртуального DOM бібліотеки React.

Для фінансового трекера, де користувач виконує часті інтерактивні дії – додає транзакції, застосовує фільтри, сортує записи та перемикається між розділами – така архітектура є оптимальною з точки зору користувацького досвіду. Важливо також, що SPA добре поєднується з компонентною моделлю React, дозволяючи локалізувати стан і логіку окремих частин інтерфейсу та уникати надлишкових мережевих запитів.

Порівняно з архітектурою, що передбачає повноцінний серверний бекенд (наприклад, Node.js + Express + PostgreSQL), клієнтська SPA з локальним

сховищем має такі переваги в контексті даного проєкту:

- відсутність витрат на серверну інфраструктуру та її адміністрування;
- миттєве розгортання у вигляді статичних файлів на будь-якому хостингу або локально;
- повна конфіденційність даних користувача, оскільки фінансова інформація не передається мережею;
- простота супроводу та відсутність залежності від серверної доступності.

Недоліком такого підходу є відсутність автоматичної синхронізації між пристроями та залежність від конкретного браузера/профілю користувача. Проте для персонального фінансового трекера базового рівня це є прийнятним обмеженням, яке дозволяє суттєво спростити реалізацію системи.

Компонентний підхід до побудови інтерфейсу. Інтерфейс застосунку декомпозовано на незалежні функціональні компоненти React, кожен з яких відповідає за відображення та логіку окремої частини системи (форма введення транзакції, таблиця операцій, панель фільтрів, блоки статистики, графіки тощо).

Такий підхід забезпечує:

- чітке розмежування відповідальностей між модулями;
- можливість повторного використання компонентів;
- спрощення тестування окремих частин застосунку;
- зручність супроводу та подальшого розширення функціональності.

Управління станом реалізується за допомогою вбудованих хуків React – `useState` для локального стану компонентів та `useEffect` для обробки побічних ефектів, зокрема синхронізації стану з `localStorage`. Вибір саме цих механізмів є достатнім для масштабу проєкту та дозволяє уникнути впровадження складніших бібліотек глобального стану (`Redux`, `MobX` тощо), потреба в яких виникає переважно у великих багатомодульних застосунках [8].

Модель зберігання даних. Для зберігання фінансових транзакцій обрано механізм `localStorage` у поєднанні з серіалізацією даних у форматі JSON. Такий підхід забезпечує персистентність даних між сесіями браузера без необхідності серверної авторизації та налаштування бази даних.

Усі операції з даними – читання, запис, оновлення та видалення – реалізуються на рівні JavaScript-логіки застосунку шляхом десеріалізації масиву транзакцій, його модифікації та повторної серіалізації. Фактично це відтворює CRUD-парадигму роботи з даними, характерну для повноцінних серверних застосунків, але засобами клієнтського середовища.

З науково-технічного погляду такий вибір є виправданим для невеликих обсягів даних (сотні або тисячі транзакцій), характерних для персонального використання. Водночас слід враховувати обмеження localStorage: невеликий доступний обсяг пам'яті, відсутність транзакційності, конкурентного доступу та механізмів резервного копіювання. Для корпоративних або багатокористувацьких систем доцільнішим був би перехід до серверної БД.

Алгоритм фільтрації та пошуку

Вибірка транзакцій за заданими критеріями реалізується шляхом послідовного застосування фільтрів до масиву об'єктів у пам'яті браузера. Алгоритм передбачає фільтрацію за:

- типом операції (дохід або витрата);
- категорією;
- часовим діапазоном.

Для кожного критерію використовується відповідна функція-предикат, результати яких об'єднуються логічним «І». Часова складність операції є лінійною відносно кількості транзакцій, що є достатнім для очікуваних обсягів даних. Відмова від складніших структур індексування (наприклад, В-дерев або повнотекстових індексів) зумовлена невеликим розміром набору даних і прагненням зберегти простоту реалізації.

У разі суттєвого збільшення кількості записів архітектура допускає подальший перехід до IndexedDB або серверної БД з індексованими запитамі без зміни загальної моделі роботи інтерфейсу.

Візуалізація аналітичних даних

Для відображення аналітики обрано два типи діаграм:

- кругова діаграма (Pie Chart) – для структури витрат за категоріями;

– стовпчаста діаграма (Bar Chart) – для динаміки доходів і витрат за місяцями.

Обидва типи реалізуються засобами бібліотеки `Recharts` з використанням SVG-рендерингу. SVG забезпечує чітке масштабоване відображення на екранах із різною роздільною здатністю та дозволяє легко інтегрувати діаграми в компонентну модель `React`.

Вхідні дані для графіків формуються шляхом агрегації масиву транзакцій за відповідними ознаками безпосередньо у `React`-компонентах: групування за категоріями для кругової діаграми та за календарними місяцями для стовпчастої. Такий підхід мінімізує дублювання даних і гарантує узгодженість аналітики з поточним станом транзакцій.

Надійність і обмеження обраного підходу. Зберігання даних локально зменшує ризики витоку інформації через серверну інфраструктуру, однак переносить відповідальність за збереження даних на користувача та браузерне середовище. У межах поставленого завдання передбачається використання стандартних механізмів браузера для персистентності даних. Для підвищення надійності в майбутньому можливе додавання експорту/імпорту резервних копій або переходу до більш стійкого локального сховища (`IndexedDB`).

Також важливо враховувати, що локальна модель зберігання не підтримує одночасну роботу кількох користувачів і не забезпечує централізованого аудиту змін. Отже, поточна архітектура свідомо орієнтована на персональне використання одним користувачем у межах одного браузерного профілю.

Прийнята схема побудови системи є обґрунтованою з науково-технічного погляду, оскільки забезпечує оптимальний баланс між функціональністю, простотою реалізації, продуктивністю інтерфейсу та конфіденційністю даних у межах поставленого завдання. Архітектура SPA на базі `React`, компонентний підхід, локальне зберігання даних через `localStorage` та використання бібліотеки `Recharts` утворюють цілісний технологічний стек, що відповідає сучасним практикам розробки клієнтських веб-застосунків [10].

Водночас обрана архітектура не суперечить можливості подальшого розвитку системи: за потреби локальне сховище може бути замінено на серверну БД, а SPA-інтерфейс – інтегрований із REST або GraphQL API без кардинальної зміни користувацького інтерфейсу та компонентної структури застосунку.

2.2 Функціонально-структурна схема роботи об'єкта проектування (блок-схеми алгоритмів, UML діаграми класів, діаграми компонентів)

На основі результатів аналізу предметної області та визначених вимог до програмного продукту сформовано функціональну модель майбутньої системи. На цьому етапі проектування визначаються цілі та завдання веб-сервісу, а також функціональні можливості, реалізація яких забезпечить ефективне ведення обліку та аналізу особистих фінансів користувача.

Веб-сервіс обліку та аналізу фінансових операцій призначений для забезпечення інформаційної підтримки процесу управління особистими фінансами користувача.

Основними цілями системи є:

- забезпечення зручного введення та зберігання фінансових транзакцій;
- надання користувачу актуальної інформації про стан особистого бюджету;
- візуальне представлення аналітичних даних для підтримки прийняття фінансових рішень;
- забезпечення цілісності та персистентності даних між сесіями роботи з застосунком.

Система орієнтована на широке коло користувачів, які бажають вести облік особистих доходів і витрат, контролювати стан власного бюджету та отримувати узагальнену аналітичну інформацію щодо фінансової діяльності. Використання веб-технологій забезпечує доступ до функціональних можливостей застосунку через сучасний браузер без необхідності встановлення додаткового програмного забезпечення.

Під час проектування системи особливу увагу приділено простоті використання та мінімізації кількості дій, необхідних для виконання основних операцій. Інтерфейс має забезпечувати швидке введення фінансових даних, наочне представлення результатів аналізу та зручну навігацію між окремими розділами застосунку.

Функціональні вимоги до системи сформовано таким чином, щоб забезпечити повний цикл роботи з фінансовими даними: від введення та зберігання інформації до її обробки, фільтрації та візуалізації.

Реалізація зазначених вимог дозволяє користувачу отримувати актуальну інформацію про структуру власних витрат і доходів, а також відстежувати зміни фінансового стану в динаміці.

Розроблюваний веб-сервіс виконує такі функції:

- автентифікація користувача при вході до системи;
- введення нової фінансової транзакції із зазначенням суми, типу (дохід або витрата), категорії, дати та опису;
- перегляд повного списку транзакцій з можливістю сортування за датою;
- редагування та видалення існуючих транзакцій;
- фільтрація транзакцій за типом, категорією та часовим діапазоном;
- відображення поточного балансу, загальної суми доходів та витрат;
- візуалізація структури витрат за категоріями у вигляді кругової діаграми;
- відображення динаміки доходів і витрат за місяцями у вигляді стовпчастої діаграми;
- збереження усіх даних у локальному сховищі браузера (localStorage).

Діаграма варіантів використання. Для формального опису функціональних вимог до системи побудовано діаграму варіантів використання (Use Case Diagram), що відображає взаємодію актора – користувача системи – з її функціональними можливостями.

Діаграма варіантів використання наведена на рисунку 2.1.

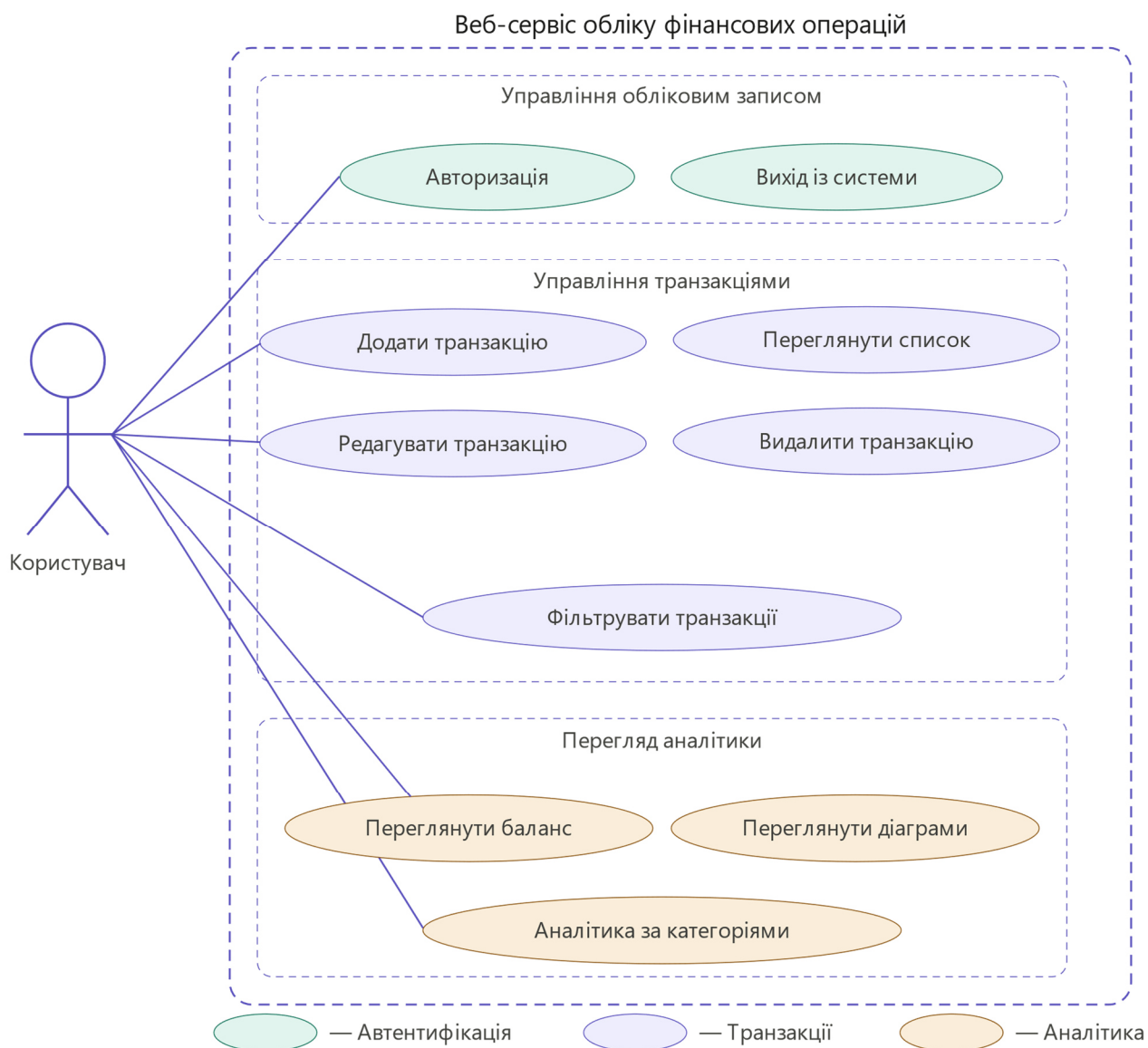


Рисунок 2.1 – Діаграма варіантів використання веб-сервісу

Як видно з рисунку 2.1, система має одного актора – користувача, який після успішної автентифікації отримує доступ до повного функціоналу застосунку.

Варіанти використання згруповано у три функціональні блоки: управління транзакціями, перегляд аналітики та управління обліковим записом.

Діаграма компонентів. Архітектуру застосунку на рівні програмних компонентів відображає діаграма компонентів (Component Diagram), наведена на рисунку 2.2.

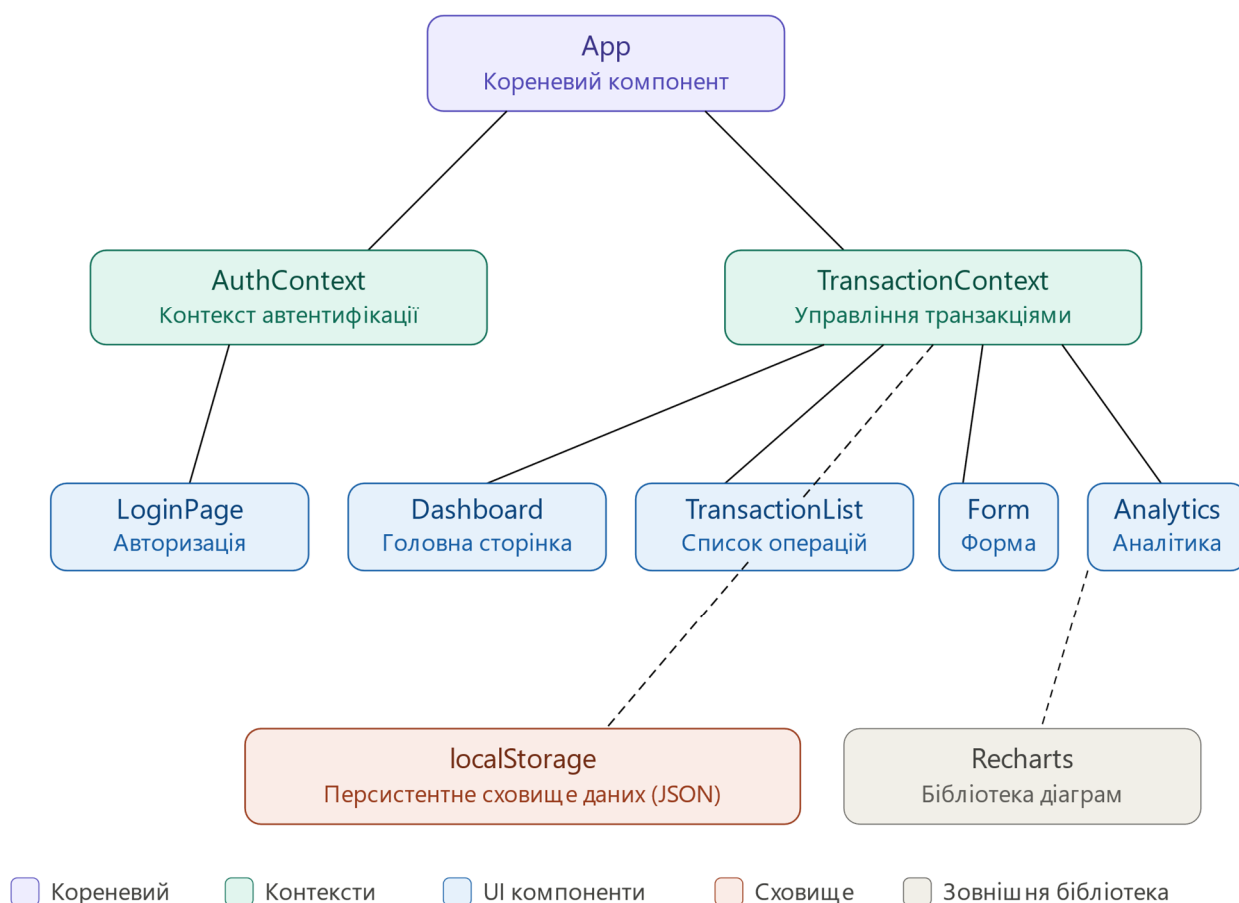


Рисунок 2.2 – Діаграма компонентів застосунку

Діаграма на рисунку 2.2 відображає декомпозицію застосунку на функціональні модулі.

Компонент App є кореневим і відповідає за маршрутизацію між розділами застосунку.

Компонент AuthContext реалізує контекст автентифікації та надає дочірнім компонентам інформацію про поточного користувача.

Компонент TransactionContext забезпечує централізоване управління станом транзакцій та їх синхронізацію з localStorage.

Компоненти Dashboard, TransactionList, TransactionForm та Analytics відповідають за відображення відповідних розділів інтерфейсу користувача.

Функціонально-структурна схема. Загальна схема роботи застосунку наведена на рисунку 2.3 та відображає інформаційні потоки між основними модулями системи.

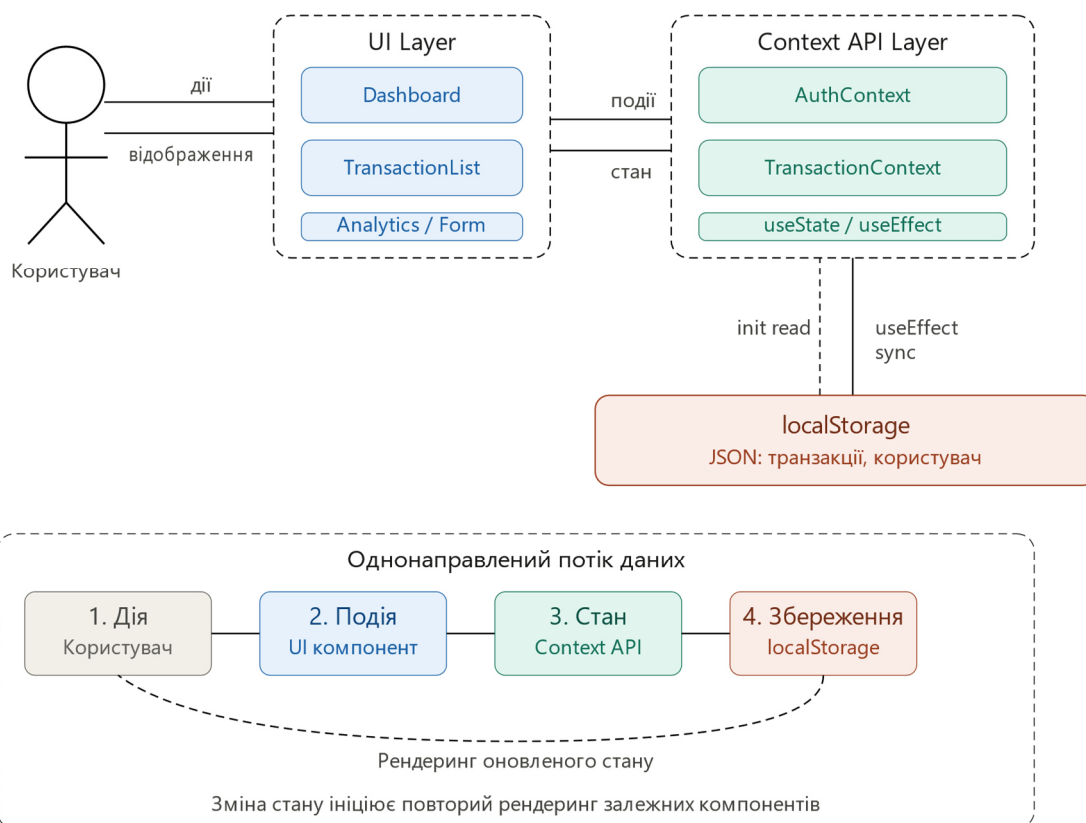


Рисунок 2.3 – Функціонально-структурна схема веб-сервісу

Як показано на рисунку 2.3, усі операції з даними проходять через шар управління станом (Context API), який забезпечує єдину точку доступу до даних для всіх компонентів інтерфейсу. Зміни стану автоматично зберігаються у localStorage через механізм useEffect, що гарантує актуальність збережених даних після кожної операції користувача.

Опис інформаційних потоків. Інформаційні потоки системи організовано за однонаправленим принципом, характерним для React-застосунків. Користувач взаємодіє з компонентами інтерфейсу, які генерують події. Події обробляються функціями-обробниками, що оновлюють централізований стан застосунку через Context API. Зміна стану автоматично ініціює повторний рендеринг залежних компонентів, що відображають актуальні дані. Паралельно з оновленням стану відбувається синхронізація даних з localStorage для забезпечення їх персистентності. Такий однонаправлений потік даних спрощує відстеження змін стану та усунення помилок у логіці застосунку.

2.3 Проектування структури даних та технологія доступу до сховища

Предметна область веб-сервісу обліку фінансових операцій охоплює три основні інформаційні сутності:

- користувач (User);
- фінансова транзакція (Transaction);
- категорія операції (Category).

Інфологічна модель даних у вигляді ER-діаграми наведена на рисунку 2.4.

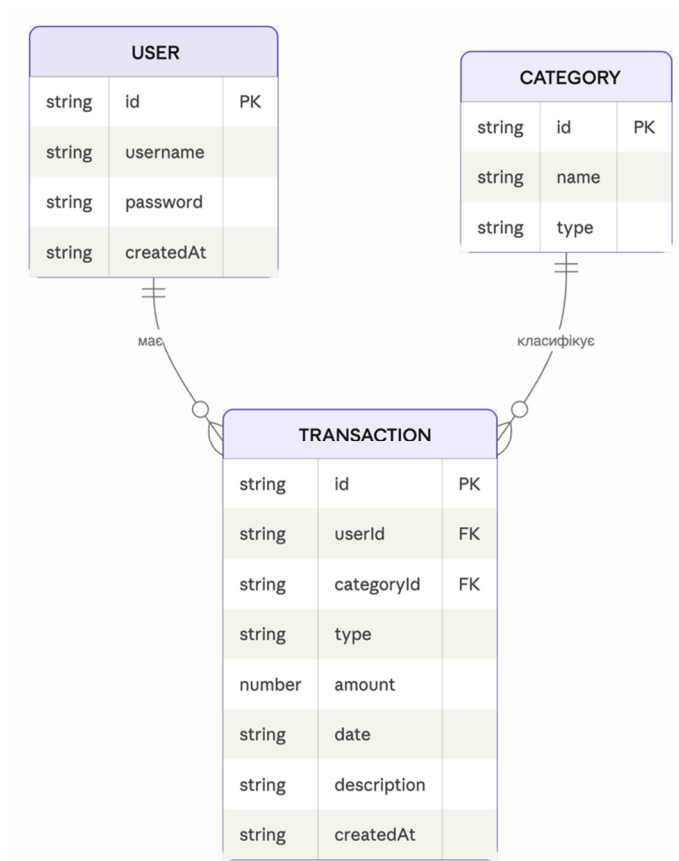


Рисунок 2.4 – ER-діаграма інфологічної моделі даних

Сутність User описує облікові дані користувача системи та містить атрибути: унікальний ідентифікатор, логін та пароль. Сутність Category визначає тематичну групу, до якої належить транзакція (наприклад, «Продукти», «Транспорт», «Зарплата»), і характеризується ідентифікатором, назвою та типом – дохід або витрата. Сутність Transaction є центральною сутністю системи і

описує окрему фінансову операцію; її атрибутами є ідентифікатор, сума, тип (дохід або витрата), категорія, дата та текстовий опис. Між сутностями визначено такі зв'язки: один користувач може мати довільну кількість транзакцій (1:N); одна категорія може бути пов'язана з довільною кількістю транзакцій (1:N).

Логічна модель даних. На основі інфологічної моделі побудовано логічну модель даних, яка визначає структуру зберігання інформації у форматі JSON. Кожна сутність представлена відповідним об'єктом або масивом об'єктів у локальному сховищі браузера. Логічна модель передбачає денормалізацію даних: назва категорії зберігається безпосередньо у полі об'єкта транзакції, а не у вигляді зовнішнього ключа. Такий підхід є обґрунтованим для key-value сховища, оскільки спрощує операції читання та виключає необхідність об'єднання даних з кількох джерел.

Вибір технології зберігання даних. Сучасні системи зберігання даних поділяються на реляційні (MySQL, PostgreSQL, SQLite), документні (MongoDB, CouchDB), key-value (Redis, localStorage), колонкові (Cassandra) та графові (Neo4j). Для клієнтського SPA-застосунку без серверної частини розглядалися такі варіанти браузерного сховища: localStorage, sessionStorage та IndexedDB [1].

localStorage забезпечує синхронний доступ до даних у форматі рядків, персистентність між сесіями та обсяг до 5-10 МБ. sessionStorage аналогічне за API, але дані очищуються після закриття вкладки, що є неприйнятним для фінансового трекера. IndexedDB є повноцінною транзакційною базою даних у браузері з асинхронним API, підтримкою індексів та значно більшим обсягом сховища, однак його складність є надлишковою для обсягів даних персонального трекера.

З урахуванням зазначеного для реалізації проекту обрано localStorage як технологію зберігання, що забезпечує оптимальний баланс між простотою API та функціональними потребами застосунку.

Технологія доступу до сховища даних. Доступ до даних реалізується через Web Storage API – стандартизований браузерний інтерфейс, специфікований у

стандарті WHATWG HTML Living Standard. Web Storage API надає синхронний доступ до сховища через два методи запису та читання: `localStorage.setItem(key, value)` та `localStorage.getItem(key)`. Оскільки `localStorage` зберігає виключно рядкові значення, усі операції з даними передбачають серіалізацію об'єктів у формат JSON при записі (`JSON.stringify`) та десеріалізацію при читанні (`JSON.parse`).

Усі операції з даними інкапсульовані у функціях `TransactionContext`, що забезпечує єдину точку доступу до сховища з будь-якого компонента застосунку. При ініціалізації застосунку дані зчитуються з `localStorage` та завантажуються у стан `React`. При кожній зміні стану транзакцій хук `useEffect` автоматично синхронізує оновлений масив з `localStorage`, що гарантує актуальність збережених даних [1].

Фізична структура даних. Дані зберігаються у `localStorage` за трьома ключами. Ключ «`transactions`» містить масив об'єктів транзакцій, ключ «`user`» – об'єкт із даними авторизованого користувача, ключ «`categories`» – масив об'єктів категорій. Фізична структура JSON-об'єктів наведена у таблиці 2.1.

Таблиця 2.1 – Фізична структура об'єкта транзакції

Поле	Тип	Опис
<code>id</code>	<code>string (UUID)</code>	Унікальний ідентифікатор транзакції
<code>type</code>	<code>string («income» / «expense»)</code>	Тип операції: дохід або витрата
<code>amount</code>	<code>number</code>	Сума операції у гривнях
<code>category</code>	<code>string</code>	Назва категорії операції
<code>date</code>	<code>string (ISO 8601)</code>	Дата операції у форматі YYYY-MM-DD
<code>description</code>	<code>string</code>	Текстовий опис операції
<code>createdAt</code>	<code>string (ISO 8601)</code>	Дата та час створення запису

Наведена у таблиці 2.1 структура об'єкта транзакції є основною одиницею зберігання даних у системі. Поле `id` формується за стандартом `UUID v4`, що гарантує унікальність кожного запису без необхідності централізованого генератора ідентифікаторів. Поле `type` приймає одне з двох фіксованих значень – «`income`» або «`expense`», що забезпечує однозначну класифікацію операції на

рівні фізичної структури. Дата зберігається у форматі ISO 8601 (YYYY-MM-DD), що спрощує операції сортування та фільтрації за часовим діапазоном засобами стандартних методів JavaScript. Поле `createdAt` фіксує точний момент створення запису і використовується для визначення порядку відображення транзакцій за замовчуванням. Таблиця 2.2 визначає частину фізичної структури даних застосунку, а саме об'єкту користувача.

Таблиця 2.2 – Фізична структура об'єкта користувача

Поле	Тип	Опис
id	string (UUID)	Унікальний ідентифікатор користувача
username	string	Логін користувача
password	string (hash)	Хешований пароль
createdAt	string (ISO 8601)	Дата реєстрації

Структура об'єкта користувача, наведена у таблиці 2.2, є мінімально достатньою для реалізації механізму автентифікації в межах браузерного середовища. Поле `password` зберігається у хешованому вигляді – перед записом до `localStorage` рядок пароля перетворюється за допомогою алгоритму хешування, що унеможливорює отримання оригінального значення при прямому читанні сховища. Об'єкт користувача зберігається за окремим ключем «user» у `localStorage` і завантажується при ініціалізації застосунку для перевірки наявності активної сесії.

Спроектowana структура даних повністю відповідає функціональним вимогам веб-сервісу, визначеним у пункті 2.2. Інфологічна модель охоплює три сутності – User, Category та Transaction – з чітко визначеними атрибутами та зв'язками типу «один до багатьох». Денормалізована логічна модель на основі JSON-масивів є оптимальною для обраної технології зберігання – Web Storage API браузера. Фізична структура об'єктів забезпечує зберігання всієї необхідної інформації для виконання CRUD-операцій, фільтрації, агрегації та побудови аналітичних звітів без залучення серверної інфраструктури.

РОЗДІЛ 3

РОЗРОБКА ВЕБ-СЕРВІСУ ОБЛІКУ ТА АНАЛІЗУ ФІНАНСОВИХ ОПЕРАЦІЙ

3.1 Практична реалізація об'єкта проектування

Застосунок реалізовано засобами React.js 18 [9] з використанням інструменту збірки Vite. Файлова структура проекту організована за компонентним принципом, де кожен модуль системи розміщено в окремій директорії відповідно до його функціонального призначення. Структуру файлів проєкту наведено на рисунку 3.1.

```

finance-tracker/
├── public/
│   └── index.html
├── src/
│   ├── components/
│   │   ├── Dashboard.jsx
│   │   ├── TransactionForm.jsx
│   │   ├── TransactionList.jsx
│   │   ├── TransactionFilter.jsx
│   │   └── Analytics.jsx
│   ├── context/
│   │   ├── AuthContext.jsx
│   │   └── TransactionContext.jsx
│   ├── utils/
│   │   ├── storage.js
│   │   └── helpers.js
│   ├── App.jsx
│   └── main.jsx
├── package.json
└── vite.config.js
  
```

Рисунок 3.1 – Структуру файлів проєкту

Директорія components/ містить React-компоненти інтерфейсу

користувача, context/ – модулі глобального управління станом, utils/ – допоміжні функції роботи зі сховищем.

Реалізація утиліт роботи зі сховищем. Усі операції читання та запису до localStorage інкапсульовано у модулі storage.js. Функції читання обгорнуто блоком try/catch для захисту від помилок десеріалізації. Лістинг 3.1 демонструє реалізацію функцій читання та запису транзакцій.

Лістинг 3.1 – Функції читання та запису транзакцій

```
const KEYS = {
  TRANSACTIONS: 'transactions',
  USER: 'user',
  CATEGORIES: 'categories',
};

export const getTransactions = () => {
  try {
    const data = localStorage.getItem(KEYS.TRANSACTIONS);
    return data ? JSON.parse(data) : [];
  } catch {
    return [];
  }
};

export const saveTransactions = (transactions) => {
  localStorage.setItem(
    KEYS.TRANSACTIONS,
    JSON.stringify(transactions)
  );
};
```

кінець лістингу 3.1

Константа KEYS централізує назви ключів сховища, що унеможливорює помилки через розбіжність рядків у різних частинах застосунку. Функція getTransactions повертає порожній масив як значення за замовчуванням, що гарантує коректну роботу компонентів при першому запуску.

Реалізація механізму автентифікації. Функція hashPassword використовує Web Crypto API – стандартний браузерний інтерфейс криптографічних операцій. Метод crypto.subtle.digest [3] обчислює хеш за алгоритмом SHA-256 та повертає ArrayBuffer, який перетворюється на шістнадцятковий рядок довжиною

64 символи. Функція `login` оголошена як `async`, оскільки операція хешування є асинхронною [14]. У сховище записується виключно хешоване значення пароля – оригінальний рядок ніде не зберігається. Реалізацію наведено у лістингу 3.2.

Лістинг 3.2 – Функція автентифікації

```
const hashPassword = async (password) => {
  const encoder = new TextEncoder();
  const data = encoder.encode(password);
  const hash = await crypto.subtle.digest('SHA-256', data);
  return Array.from(new Uint8Array(hash))
    .map(b => b.toString(16).padStart(2, '0'))
    .join('');
};

const login = async (username, password) => {
  const stored = getUser();
  const hashedPassword = await hashPassword(password);
  if (stored && stored.username === username
    && stored.password === hashedPassword) {
    setUser(stored);
    return true;
  }
  if (!stored && username && password) {
    const newUser = {
      id: crypto.randomUUID(),
      username,
      password: hashedPassword,
      createdAt: new Date().toISOString(),
    };
    saveUser(newUser);
    setUser(newUser);
    return true;
  }
  return false;
};
```

кінець лістингу 3.2

Для генерації унікальних ідентифікаторів використовується стандартний браузерний метод `crypto.randomUUID()`, що забезпечує унікальність без підключення зовнішніх бібліотек [2].

Відновлення сесії при перезавантаженні. Хук `useEffect` при монтуванні компонента `AuthProvider` зчитує збережені дані користувача та відновлює стан сесії. Реалізацію наведено у лістингу 3.3.

Лістинг 3.3 – Відновлення сесії

```
const [user, setUser] = useState(null);
const [loading, setLoading] = useState(true);

useEffect(() => {
  const stored = getUser();
  if (stored) setUser(stored);
  setLoading(false);
}, []);
```

кінець лістингу 3.3

Прапорець `loading` запобігає рендерингу дочірніх компонентів до завершення ініціалізації – це виключає короткочасне відображення сторінки авторизації для вже автентифікованого користувача.

Реалізація CRUD-операцій над транзакціями [17]. Функції додавання, редагування та видалення транзакцій реалізовано у `TransactionContext.jsx`. Лістинг 3.4 демонструє функцію додавання нової транзакції.

Лістинг 3.4 – Функція додавання транзакції

```
const addTransaction = (data) => {
  const transaction = {
    id: crypto.randomUUID(),
    ...data,
    createdAt: new Date().toISOString(),
  };
  setTransactions((prev) => [transaction, ...prev]);
};
```

кінець лістингу 3.4

Новий запис розміщується на початку масиву, що забезпечує відображення останніх транзакцій першими без додаткового сортування. Лістинг 3.5 демонструє функції редагування та видалення.

Лістинг 3.5 – Функції редагування та видалення

```
const editTransaction = (id, data) => {
  setTransactions((prev) =>
    prev.map((t) => (t.id === id ? { ...t, ...data } : t))
  );
};
```

```
const deleteTransaction = (id) => {
  setTransactions((prev) =>
    prev.filter((t) => t.id !== id)
  );
};
```

кінець лістингу 3.5

Обидві функції не мутують вихідний масив, а повертають новий – це відповідає принципу незмінності стану React та гарантує коректне визначення змін для повторного рендерингу.

Синхронізація стану зі сховищем. Автоматичне збереження транзакцій реалізовано через другий хук `useEffect`, що спрацьовує при кожній зміні масиву, як показано у лістингу 3.6.

Лістинг 3.6 – Синхронізація з `localStorage`

```
useEffect(() => {
  setTransactions(getTransactions());
  setCategories(getCategories());
}, []);

useEffect(() => {
  saveTransactions(transactions);
}, [transactions]);
```

кінець лістингу 3.6

Перший `useEffect` з порожнім масивом залежностей виконується одноразово при монтуванні та завантажує початкові дані. Другий реагує на зміну `transactions` і автоматично зберігає актуальний стан – жодна операція зміни даних не потребує явного виклику функції збереження.

Алгоритм фільтрації реалізовано у функції `getFiltered` шляхом послідовного застосування предикатів до масиву транзакцій (ліст. 3.7).

Лістинг 3.7 – Алгоритм фільтрації

```
const getFiltered = ({ type, category, dateFrom, dateTo }) => {
  return transactions.filter((t) => {
    if (type && t.type !== type) return false;
    if (category && t.category !== category) return false;
    if (dateFrom && t.date < dateFrom) return false;
  });
};
```

```

    if (dateTo && t.date > dateTo) return false;
    return true;
  });
};

```

кінець лістингу 3.7

Блок-схема алгоритму фільтрації (рис. 3.2):

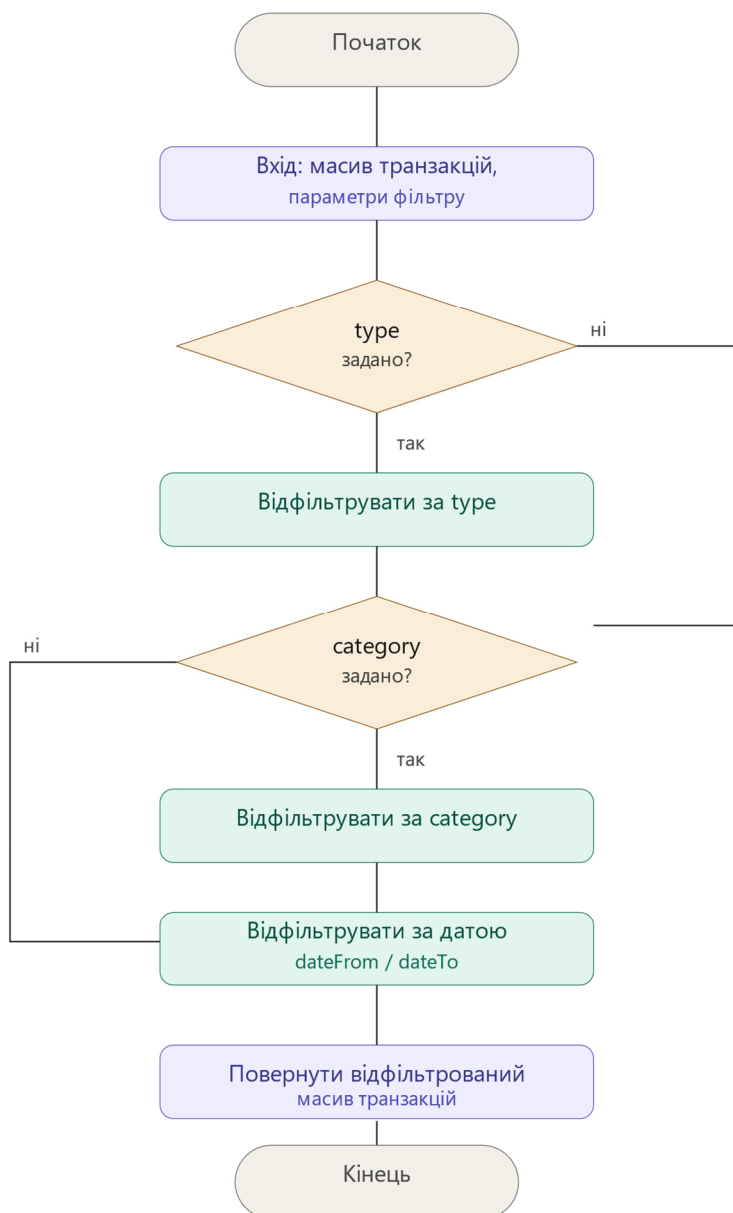


Рисунок 3.2 – Блок-схема алгоритму фільтрації

Кожен критерій є необов'язковим – порожнє значення параметра означає відсутність обмеження за цим критерієм. Порівняння дат виконується як рядкове

у форматі ISO 8601, що є коректним завдяки лексикографічному збігу цього формату з хронологічним порядком.

Реалізація аналітичних обчислень. Дані для діаграм формуються за допомогою хука `useMemo`, що кешує результати обчислень.

Блок-схему алгоритму обробки форми наведено на рисунку 3.3.

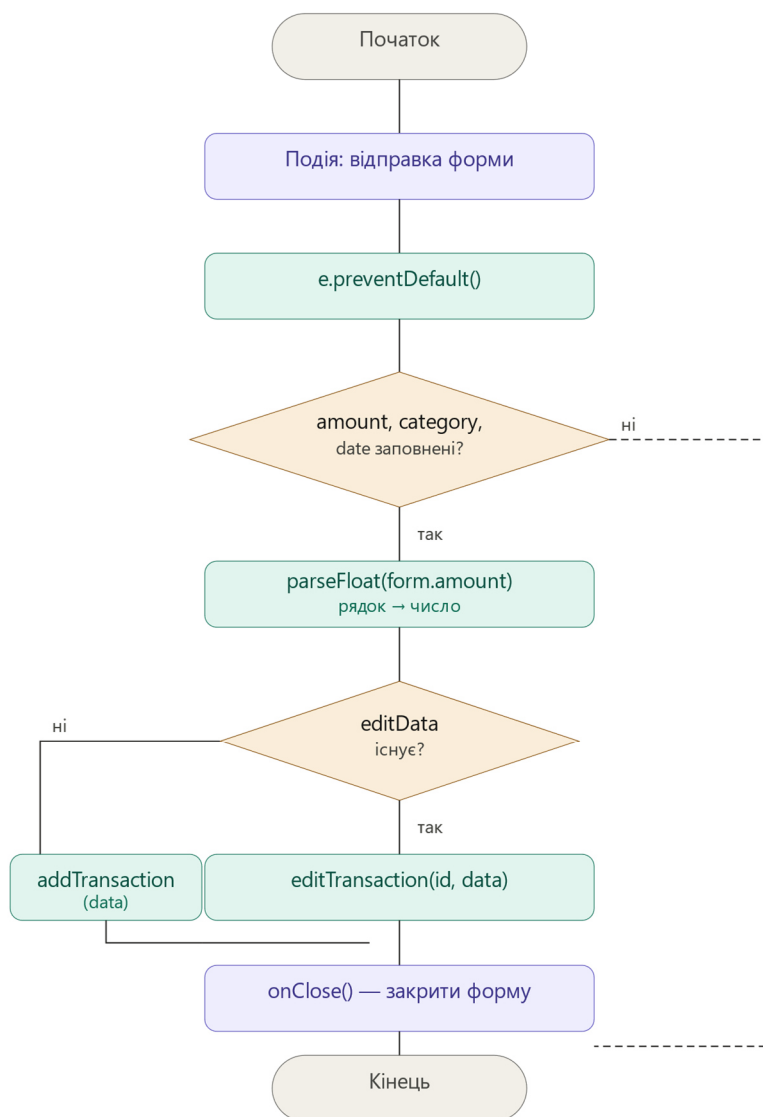


Рисунок 3.3 – Блок-схема алгоритму обробки форми

Лістинг 3.8 демонструє агрегацію витрат за категоріями для кругової діаграми.

Лістинг 3.8 – Агрегація витрат за категоріями

```

const pieData = useМемо(() => {
  const map = {};

```

```

transactions
  .filter((t) => t.type === 'expense')
  .forEach((t) => {
    map[t.category] = (map[t.category] || 0) + t.amount;
  });

return Object.entries(map)
  .map(([name, value]) => ({ name, value }));
}, [transactions]);

```

кінець лістингу 3.8

Лістинг 3.9 демонструє формування даних для стовпчастої діаграми динаміки по місяцях.

Лістинг 3.9 – Агрегація даних по місяцях

```

const barData = useMemo(() => {
  const map = {};
  transactions.forEach((t) => {

    const month = t.date.slice(0, 7);
    if (!map[month]) map[month] = { month, income: 0, expense: 0 };
    if (t.type === 'income') map[month].income += t.amount;
    else map[month].expense += t.amount;
  });

  return Object.values(map)
    .sort((a, b) => a.month.localeCompare(b.month));
}, [transactions]);

```

кінець лістингу 3.9

Ключем групування є перші 7 символів поля date у форматі YYYY-MM. Результуючий масив сортується лексикографічно за цим ключем, що відповідає хронологічному порядку.

Реалізація компонента форми. Компонент TransactionForm.jsx підтримує два режими роботи – додавання та редагування – залежно від наявності пропу editData. Лістинг 3.10 демонструє логіку обробки форми.

Лістинг 3.10 – Обробник форми транзакції

```

const handleSubmit = (e) => {
  e.preventDefault();
  if (!form.amount || !form.category || !form.date) return;

```

```

const data = { ...form, amount: parseFloat(form.amount) };
if (editData) {
  editTransaction(editData.id, data);
} else {
  addTransaction(data);
}
onClose();
};

```

кінець лістингу 3.10

Перед відправкою виконується базова валідація обов'язкових полів. Значення суми явно перетворюється з рядка на число через `parseFloat`, оскільки поля введення HTML повертають рядкові значення незалежно від атрибута `type=«number»`.

3.2 Тестування та налагодження веб-сервісу обліку та аналізу фінансових операцій

Методи тестування. Для перевірки коректності роботи розробленого веб-сервісу застосовано метод ручного функціонального тестування. Цей метод передбачає безпосередню взаємодію тестувальника з інтерфейсом застосунку шляхом виконання заздалегідь визначених сценаріїв використання та перевірку відповідності фактичних результатів очікуваним. Для клієнтського SPA-застосунку даного рівня складності ручне функціональне тестування є достатнім методом верифікації, оскільки дозволяє охопити всі визначені варіанти використання системи.

Тестування проводилося у браузері Google Chrome версії 124 на операційній системі Windows 11. Додатково виконано перевірку сумісності у браузерах Mozilla Firefox 125 та Microsoft Edge 124 для підтвердження коректної роботи Web Storage API та SVG-рендерингу діаграм у різних середовищах виконання.

Тест-кейси. Для систематизації процесу тестування розроблено набір тест-кейсів, що охоплюють основні функціональні сценарії системи. Результати

тестування наведено у таблиці 3.1.

Таблиця 3.1 – Результати функціонального тестування

№	Сценарій	Вхідні дані	Очікуваний результат	Фактичний результат	Статус
1	Перший запуск застосунку	–	Відображається форма авторизації, localStorage порожній	Відповідає очікуваному	Пройдено
2	Реєстрація нового користувача	login: «user1», password: «pass123»	Створено обліковий запис, виконано вхід, відображається Dashboard	Відповідає очікуваному	Пройдено
3	Повторний вхід з коректними даними	login: «user1», password: «pass123»	Успішна автентифікація, відновлення сесії	Відповідає очікуваному	Пройдено
4	Вхід з невірним паролем	login: «user1», password: «wrong»	Форма не закривається, вхід не виконується	Відповідає очікуваному	Пройдено
5	Додавання витрати	Сума: 250, категорія: «Продукти», дата: поточна	Транзакція з'являється у списку, баланс зменшується на 250	Відповідає очікуваному	Пройдено
6	Додавання доходу	Сума: 15000, категорія: «Зарплата», дата: поточна	Транзакція з'являється у списку, баланс збільшується на 15000	Відповідає очікуваному	Пройдено
7	Відправка форми з порожніми полями	Усі поля порожні	Форма не відправляється, транзакція не створюється	Відповідає очікуваному	Пройдено
8	Редагування транзакції	Зміна суми з 250 на 300	Запис у списку оновлюється, баланс перераховується	Відповідає очікуваному	Пройдено
9	Видалення транзакції	Існуючий запис	Запис зникає зі списку, баланс перераховується	Відповідає очікуваному	Пройдено

Продовження таблиці 3.2

№	Сценарій	Вхідні дані	Очікуваний результат	Фактичний результат	Статус
10	Фільтрація за типом	Тип: «Витрата»	У списку відображаються лише витрати	Відповідає очікуваному	Пройдено
11	Фільтрація за категорією	Категорія: «Транспорт»	У списку лише транзакції категорії «Транспорт»	Відповідає очікуваному	Пройдено
12	Фільтрація за датою	dateFrom: 01.05.2025, dateTo: 31.05.2025	Відображаються лише транзакції травня 2025	Відповідає очікуваному	Пройдено
13	Відображення кругової діаграми	Наявні транзакції-витрати різних категорій	Діаграма відображає сегменти пропорційно сумам	Відповідає очікуваному	Пройдено
14	Відображення стовпчастої діаграми	Транзакції за кілька місяців	Діаграма відображає доходи та витрати по місяцях	Відповідає очікуваному	Пройдено
15	Персистентність даних	Перезавантаження сторінки після додавання транзакцій	Усі транзакції збережено, сесія відновлена	Відповідає очікуваному	Пройдено
16	Вихід із системи	Натискання кнопки «Вийти»	Сесія завершена, відображається форма авторизації	Відповідає очікуваному	Пройдено

Як видно з таблиці 3.1, усі 16 тест-кейсів пройдено успішно. Критичних помилок у роботі застосунку не виявлено. Функціональність системи відповідає вимогам, визначеним у розділі 2.

Налагодження. У процесі розробки для виявлення та усунення помилок використовувались інструменти розробника браузерa Google Chrome DevTools. Консоль браузерa застосовувалась для відстеження помилок JavaScript та перевірки значень змінних стану. Вкладка Application → Local Storage використовувалась для безпосереднього контролю вмісту сховища після кожної

операції запису. Вкладка React Developer Tools дозволяла інспектувати стан компонентів та значення контекстів у режимі реального часу.

Мінімальні системні вимоги. За результатами тестування визначено мінімальні системні вимоги для коректної роботи веб-сервісу, наведені у таблиці 3.2.

Таблиця 3.2 – Системні вимоги

Параметр	Мінімальні вимоги	Рекомендовані параметри
Браузер	Chrome 90+, Firefox 88+, Edge 90+, Safari 14+	Chrome 120+ або Firefox 120+
Оперативна пам'ять	512 МБ	2 ГБ і більше
Роздільна здатність екрана	1024×768 пікселів	1920×1080 пікселів
Підключення до мережі	Не потрібне (офлайн-режим)	—
Операційна система	Будь-яка з підтримкою сучасного браузера	Windows 10/11, macOS 12+, Ubuntu 20.04+
Вільне місце у сховищі браузера	1 МБ (localStorage)	5 МБ

Як показано у таблиці 3.2, застосунок не висуває вимог до серверної інфраструктури та підключення до мережі, що є суттєвою перевагою для кінцевого користувача.

3.3 Інструкція користувача з інсталяції та використання програмного продукту

Інсталяція та запуск. Веб-сервіс обліку та аналізу фінансових операцій є клієнтським застосунком і не потребує встановлення додаткового програмного забезпечення на пристрій кінцевого користувача. Для розгортання застосунку у режимі розробки необхідно виконати такі кроки.

Крок 1. Встановити середовище виконання Node.js версії 18 або вище з офіційного сайту nodejs.org.

Крок 2. Завантажити вихідний код проєкту та розпакувати його у довільну

директорію.

Крок 3. Відкрити термінал у директорії проєкту та виконати команду встановлення залежностей, наведену у лістингу 3.11.

Лістинг 3.11 – Встановлення залежностей та запуск

```
npm install
npm run dev
```

кінець лістингу 3.11

Крок 4. Після успішного запуску у терміналі відобразиться адреса локального сервера розробки. Відкрити зазначену адресу у браузері для запуску застосунку.

Для розгортання у вигляді статичного сайту (production-збірка) необхідно виконати команду `npm run build`, після чого вміст директорії `dist/` розмістити на будь-якому статичному хостингу (GitHub Pages, Netlify, Vercel тощо).

Реєстрація та вхід. При першому запуску застосунок відображає форму авторизації. Оскільки сервіс не передбачає підключення до зовнішнього сервера, перший вхід є одночасно реєстрацією: користувач вводить довільний логін та пароль і натискає кнопку «Увійти». Введені дані зберігаються у `localStorage` браузера і використовуються для автентифікації при наступних сеансах роботи. При повторному відкритті застосунку сесія відновлюється автоматично без повторного введення даних.

Робота з транзакціями. Після входу користувач опиняється на головній сторінці `Dashboard`, де відображається поточний баланс, загальна сума доходів та витрат. Для додавання нової транзакції необхідно натиснути кнопку «Додати транзакцію» та заповнити форму: обрати тип операції (дохід або витрата), ввести суму, обрати категорію зі списку, вказати дату та за потреби додати текстовий опис. Поля суми, категорії та дати є обов'язковими.

Перелік усіх транзакцій доступний у розділі «Транзакції». Для редагування запису необхідно натиснути кнопку редагування поруч з відповідним рядком –

форма відкриється із заповненими поточними значеннями. Для видалення запису необхідно натиснути кнопку видалення та підтвердити дію.

Фільтрація. У розділі «Транзакції» доступна панель фільтрації, що дозволяє звузити перелік операцій за типом (дохід або витрата), категорією та часовим діапазоном. Фільтри є незалежними та можуть застосовуватися одночасно у будь-якій комбінації. Для скидання фільтрів достатньо очистити відповідні поля.

Аналітика. Розділ «Аналітика» містить два типи діаграм. Кругова діаграма відображає структуру витрат у розрізі категорій за весь період – розмір сегменту пропорційний сумі витрат відповідної категорії. Стовпчаста діаграма відображає динаміку доходів і витрат у розрізі місяців, що дозволяє відстежувати зміни фінансового стану у часі. Діаграми оновлюються автоматично при додаванні або зміні транзакцій.

Вихід із системи. Для завершення сеансу роботи необхідно натиснути кнопку «Вийти» у верхньому меню застосунку. Дані транзакцій при цьому зберігаються у localStorage і будуть доступні при наступному вході.

РОЗДІЛ 4

СПЕЦІАЛЬНІ РОЗРАХУНКИ

Веб-застосунки, що працюють у браузерному середовищі та зберігають персональні дані користувача на стороні клієнта, є об'єктом специфічного класу загроз інформаційній безпеці. На відміну від серверних застосунків, де дані захищені на рівні інфраструктури, клієнтський SPA-застосунок функціонує у відкритому середовищі браузера, що обумовлює необхідність окремого аналізу актуальних загроз та визначення відповідних заходів захисту.

Для розробленого веб-сервісу обліку фінансових операцій визначено такі основні категорії загроз.

Несанкціонований доступ до даних сховища. Механізм `localStorage` доступний через консоль браузера будь-якому JavaScript-коду, що виконується на сторінці. У разі отримання фізичного доступу до пристрою або відкриття браузерних інструментів розробника стороння особа може безпосередньо переглянути вміст сховища. Ця загроза є актуальною для застосунків, що обробляють фінансові дані.

Атаки типу XSS (Cross-Site Scripting). Міжсайтовий скриптинг передбачає впровадження шкідливого JavaScript-коду у сторінку застосунку з метою отримання несанкціонованого доступу до даних, зокрема до вмісту `localStorage`. Успішна XSS-атака дозволяє зловмиснику зчитати або модифікувати збережені дані без відома користувача [5].

Розкриття облікових даних. Зберігання пароля у відкритому (незашифрованому) вигляді у `localStorage` створює критичну вразливість: будь-яка особа з доступом до сховища отримує пароль у вигляді відкритого тексту, що може бути використано для несанкціонованого доступу до інших сервісів у разі повторного використання пароля користувачем.

Підробка даних. Відсутність серверної валідації означає, що дані у `localStorage` можуть бути безпосередньо модифіковані через консоль браузера.

Зловмисник або сам користувач може змінити значення транзакцій, що порушує цілісність фінансових записів [6].

Аналіз загроз є підґрунтям для визначення конкретних заходів захисту, реалізованих у застосунку та описаних у наступних пунктах.

Загальна схема автентифікації. У розробленому застосунку реалізовано механізм автентифікації на основі локального сховища браузера. Схему процесу автентифікації наведено на рисунку 4.1.

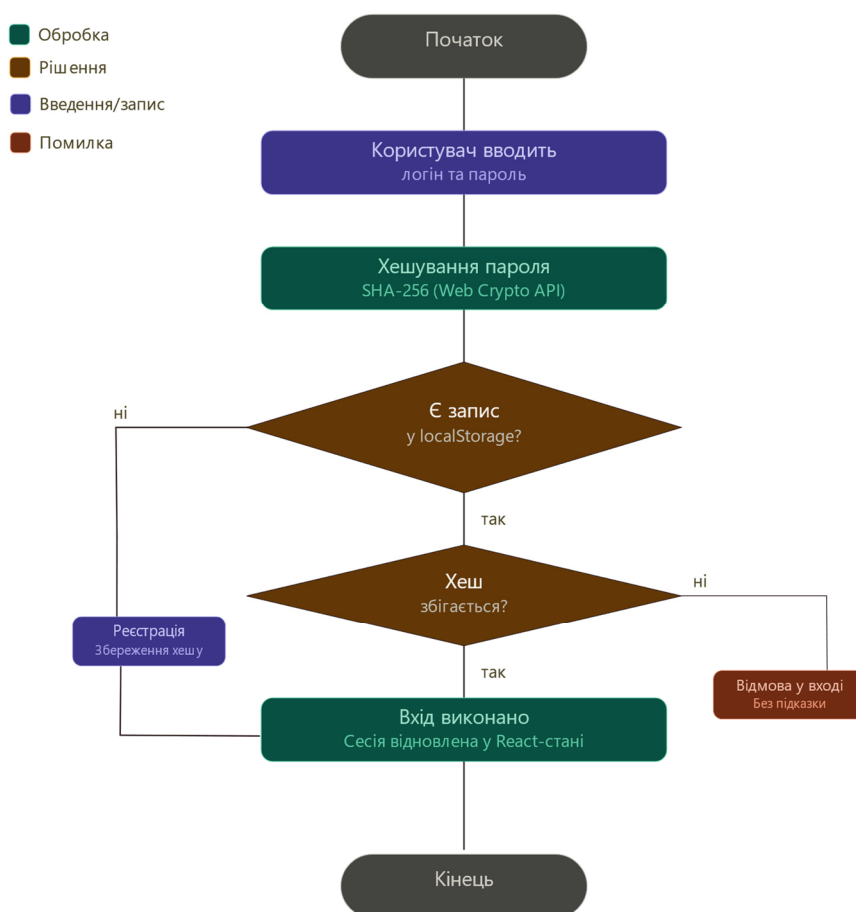


Рисунок 4.1 – Схема процесу автентифікації

Як показано на рисунку 4.1, при вході користувач вводить логін та пароль. Пароль негайно хешується на стороні клієнта, після чого хеш порівнюється зі збереженим значенням. Оригінальний рядок пароля ніде не зберігається і не передається – він існує виключно у пам’яті браузера протягом часу виконання функції хешування.

Алгоритм хешування SHA-256. Для захисту пароля застосовано алгоритм хешування SHA-256 (Secure Hash Algorithm 256-bit), реалізований через стандартний браузерний інтерфейс Web Crypto API. SHA-256 належить до сімейства криптографічних хеш-функцій SHA-2, розроблених Агентством національної безпеки США та стандартизованих NIST у 2001 році. Алгоритм перетворює вхідний рядок довільної довжини на хеш фіксованого розміру – 256 біт (64 шістнадцяткових символи). Ключовими властивостями алгоритму є: детермінованість (однаковий вхід завжди дає однаковий вихід), незворотність (відновлення вихідного рядку з хешу є обчислювально неможливим), та лавинний ефект (незначна зміна вхідних даних призводить до повної зміни хешу).

Реалізацію функції хешування наведено у лістингу 4.1.

Лістинг 4.1 – Функція хешування пароля (Web Crypto API)

```
const hashPassword = async (password) => {
  const encoder = new TextEncoder();
  const data = encoder.encode(password);
  const hash = await crypto.subtle.digest('SHA-256', data);
  return Array.from(new Uint8Array(hash))
    .map(b => b.toString(16).padStart(2, '0'))
    .join('');
};
```

кінець лістингу 4.1

Як видно з лістингу 4.1, функція використовує TextEncoder для перетворення рядка пароля на масив байтів у кодуванні UTF-8. Метод crypto.subtle.digest виконує обчислення хешу та повертає ArrayBuffer – низькорівневий буфер бінарних даних. Результат перетворюється на масив беззнакових 8-бітних цілих (Uint8Array), кожен елемент якого конвертується у двосимвольний шістнадцятковий рядок методом toString(16) з вирівнюванням нулем (padStart(2, '0')). Остаточний хеш є рядком довжиною 64 символи [3].

Захист від несанкціонованого входу. Функція автентифікації login, наведена у лістингу 3.2, реалізує захист від підбору пароля на рівні логіки застосунку: при невідповідності хешу введеного пароля збереженому значенню

функція повертає false без будь-яких підказок щодо причини відмови. Це унеможлиблює розрізнення між помилкою логіну та помилкою пароля, що ускладнює атаки перебором [7].

Природний захист React від XSS. Основним засобом захисту від XSS-атак у розробленому застосунку є архітектурна особливість бібліотеки React.js – автоматичне екранування всіх значень, що виводяться у JSX-розмітці. При рендерингу React перетворює спеціальні HTML-символи (<, >, &, ", ') на відповідні HTML-сутності, що унеможлиблює інтерпретацію введених користувачем даних як HTML-розмітки або JavaScript-коду. Наприклад, якщо користувач введе у поле опису транзакції рядок <script>alert('xss')</script>, React відобразить його як звичайний текст, а не виконає як код [8].

Лістинг 4.2 демонструє, як React безпечно відображає довільні дані транзакції.

Лістинг 4.2 – Безпечне відображення даних у JSX

```
const TransactionItem = ({ transaction }) => (
  <div className="flex justify-between items-center p-3 border rounded-
lg">
    <span className="text-sm">{transaction.description}</span>
    <span className="font-medium">
      {transaction.type === 'income' ? '+' : '-'}
      {transaction.amount.toFixed(2)} грн
    </span>
  </div>
);
```

кінець лістингу 4.2

У лістингу 4.2 значення transaction.description та transaction.amount виводяться через JSX-вирази у фігурних дужках. React автоматично екранує ці значення перед вставкою у DOM, що унеможлиблює XSS навіть у разі, якщо дані містять шкідливий код.

Валідація вхідних даних. Додатковим рівнем захисту є клієнтська валідація форми, реалізована в обробнику handleSubmit (лістинг 3.10). Поле суми приймає виключно числові значення завдяки атрибуту type=«number» та явному

перетворенню через `parseFloat`. Поле дати обмежено форматом `YYYY-MM-DD` через `type=«date»`. Поле категорії є випадającym списком із фіксованим набором значень, що виключає введення довільних рядків.

Структура зберігання даних. Усі фінансові дані зберігаються у `localStorage` виключно на пристрої користувача і не передаються мережею. Це забезпечує конфіденційність даних від зовнішніх загроз: перехоплення мережевого трафіку, витоки з серверної бази даних або несанкціонований доступ до хмарного сховища є неможливими за архітектурою системи.

Цілісність масиву транзакцій. Для забезпечення цілісності даних кожна транзакція при створенні отримує унікальний ідентифікатор, сформований методом `crypto.randomUUID()`, як показано у лістингу 4.3 [2; 13].

Лістинг 4.3 – Генерація унікального ідентифікатора транзакції

```
const addTransaction = (data) => {
  const transaction = {
    id: crypto.randomUUID(),
    ...data,
    createdAt: new Date().toISOString(),
  };
  setTransactions((prev) => [transaction, ...prev]);
};
```

кінець лістингу 4.3

Метод `crypto.randomUUID()` генерує ідентифікатор версії 4 стандарту RFC 4122 – 128-бітне псевдовипадкове число у форматі `xxxxxxxx-xxxx-4xxx-uxxx-xxxxxxxxxxxx`. Використання криптографічно стійкого генератора псевдовипадкових чисел виключає можливість передбачення або колізії ідентифікаторів [2].

Захист сесії. Стан автентифікації зберігається виключно в оперативній пам'яті React-застосунку (у стані компонента `AuthProvider`) і не дублюється у `localStorage` у вигляді токена сесії. Це означає, що після перезавантаження сторінки перевірка наявності облікового запису виконується повторно через зчитування збереженого об'єкта користувача, а не через перевірку сесійного

токена. Такий підхід унеможливилює підробку сесії шляхом прямого запису даних до localStorage [1; 9].

Узагальнена таблиця заходів захисту. Реалізовані заходи інформаційної безпеки систематизовано у таблиці 4.1.

Таблиця 4.1 – Заходи захисту інформації у веб-сервісі

Загроза	Захід захисту	Засіб реалізації
Розкриття пароля	Хешування SHA-256	Web Crypto API (crypto.subtle)
XSS-атаки	Автоматичне екранування виводу	React JSX rendering
Підробка введених даних	Валідація типів та форматів	HTML5 input types, parseFloat
Несанкціонований вхід	Порівняння хешів без підказок	Функція login
Передбачення ідентифікаторів	Криптостійка генерація UUID	crypto.randomUUID()
Підробка сесії	Відсутність сесійного токена	Стан у пам'яті React
Витік даних через мережу	Відсутність передачі даних	Архітектура localStorage SPA

Як видно з таблиці 4.1, реалізовані заходи захисту охоплюють усі загрози, визначені у пункті 4.1, та забезпечують достатній рівень інформаційної безпеки для персонального фінансового веб-сервісу клієнтського типу.

ВИСНОВКИ

У кваліфікаційній роботі вирішено актуальну задачу розробки клієнтського веб-сервісу обліку та аналізу фінансових операцій, що функціонує без серверної інфраструктури та забезпечує конфіденційність персональних фінансових даних користувача. Усі завдання, визначені у пункті 1.4, виконано в повному обсязі.

Проведено аналіз сучасного стану проблеми персонального фінансового обліку та досліджено три програмні аналоги – Spendee, Money Manager та Wallet by BudgetBakers. Встановлено, що жоден із розглянутих аналогів не забезпечує одночасно безкоштовного доступу до повного функціоналу, локального зберігання даних та роботи без обов’язкової реєстрації, що підтвердило доцільність розробки власного рішення.

Розроблено специфікацію вимог до системи, побудовано діаграму варіантів використання (16 варіантів використання, 1 актор), діаграму компонентів та функціонально-структурну схему застосунку. Спроековано модель даних на основі трьох сутностей – User, Category, Transaction – та визначено фізичну структуру JSON-об’єктів із сімома атрибутами транзакції.

Реалізовано клієнтський SPA-застосунок засобами React.js 18 та Vite, що забезпечує повний цикл CRUD-операцій над фінансовими транзакціями, фільтрацію за чотирма незалежними критеріями, відображення аналітики у вигляді кругової та стовпчастої діаграм засобами Recharts. Загальний обсяг реалізованого програмного коду становить 10 модулів із 10 ключовими лістингами, що описують алгоритми системи.

Реалізовано механізм автентифікації з хешуванням пароля за алгоритмом SHA-256 засобами Web Crypto API [14]. Застосування криптографічно стійкого хешування унеможливлює отримання оригінального пароля при прямому читанні localStorage. Додатково реалізовано генерацію унікальних ідентифікаторів транзакцій засобами crypto.randomUUID().

Проведено функціональне тестування застосунку за 16 тест-кейсами у

браузерах Chrome, Firefox та Edge. Усі тест-кейси пройдено успішно, критичних помилок не виявлено. Визначено мінімальні системні вимоги: будь-який сучасний браузер (Chrome 90+, Firefox 88+, Edge 90+), оперативна пам'ять від 512 МБ, без необхідності підключення до мережі.

Проаналізовано загрози інформаційній безпеці клієнтського веб-застосунку та реалізовано комплекс заходів захисту, що охоплює сім категорій загроз. Встановлено, що архітектура SPA з локальним сховищем забезпечує захист від витоку даних через мережу на структурному рівні.

Розроблений веб-сервіс відповідає тенденціям розвитку клієнтських веб-технологій – зростанню ролі браузерного середовища як повноцінної платформи для безпечних застосунків без серверної складової. Використані технології React.js, Vite та Web Crypto API є актуальними інструментами сучасної веб-розробки з широкою підтримкою у промисловому секторі.

Результати роботи можуть бути практично використані у таких галузях: персональне фінансове планування для фізичних осіб; навчальні курси з веб-розробки як демонстраційний приклад SPA-архітектури з контекстами React; малий бізнес для базового обліку господарських операцій без необхідності впровадження повноцінної бухгалтерської системи.

Подальше вдосконалення розробленого веб-сервісу може здійснюватись у таких напрямках: реалізація серверної частини на Node.js або Python для забезпечення синхронізації даних між пристроями; впровадження технології Progressive Web App (PWA) для встановлення застосунку на мобільні пристрої та розширеної офлайн-підтримки; додавання функціоналу бюджетування з визначенням лімітів витрат за категоріями та системою сповіщень; реалізація експорту фінансових звітів у форматах PDF та CSV; застосування алгоритмів машинного навчання для автоматичної категоризації транзакцій та прогнозування витрат.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. localStorage, sessionStorage. Сучасний підручник з JavaScript. URL: <https://uk.javascript.info/localstorage> (date of access: 04.04.2026).
2. MDN Web Docs. crypto.randomUUID(). Mozilla Developer Network. URL: <https://developer.mozilla.org/en-US/docs/Web/API/Crypto/randomUUID> (дата звернення: 10.05.2025).
3. MDN Web Docs. SubtleCrypto: digest() method. Mozilla Developer Network. URL: <https://developer.mozilla.org/en-US/docs/Web/API/SubtleCrypto/digest> (дата звернення: 10.05.2025).
4. MDN Web Docs. Web Storage API. Mozilla Developer Network. URL: https://developer.mozilla.org/en-US/docs/Web/API/Web_Storage_API (дата звернення: 10.05.2025).
5. NIST. FIPS PUB 180-4: Secure Hash Standard (SHS). National Institute of Standards and Technology. URL: <https://csrc.nist.gov/publications/detail/fips/180/4/final> (дата звернення: 10.05.2025).
6. OWASP Foundation. Cross Site Scripting (XSS). OWASP Cheat Sheet Series. URL: https://cheatsheetseries.owasp.org/cheatsheets/Cross_Site_Scripting_Prevention_Cheat_Sheet.html (дата звернення: 10.05.2025).
7. OWASP Foundation. OWASP Top Ten — Web Application Security Risks. URL: <https://owasp.org/www-project-top-ten/> (дата звернення: 10.05.2025).
8. React Documentation. React Reference Overview. Meta Open Source, 2024. URL: <https://react.dev/reference/react> (дата звернення: 10.05.2025).
9. React Documentation. React v18.0. Official Release Notes. Meta Open Source. URL: <https://react.dev/blog/2022/03/29/react-v18> (дата звернення: 10.05.2025).
10. Recharts. A Composable Charting Library Built on React Components. Official Documentation. URL: <https://recharts.org/en-US> (дата звернення: 10.05.2025).

11. Tailwind CSS. A Utility-First CSS Framework. Official Documentation. URL: <https://tailwindcss.com/docs> (дата звернення: 10.05.2025).
12. Vite. Next Generation Frontend Tooling. Official Documentation. URL: <https://vitejs.dev/guide/> (дата звернення: 10.05.2025).
13. W3C. Web Cryptography Level 2. First Public Working Draft. World Wide Web Consortium, 2023. URL: <https://www.w3.org/TR/webcrypto-2/> (дата звернення: 10.05.2025).
14. Web Crypto API – Web APIs | MDN. MDN Web Docs. URL: https://developer.mozilla.org/en-US/docs/Web/API/Web_Crypto_API (дата звернення: 11.03.2026).
15. WHATWG. Web Storage (Second Edition). HTML Living Standard. URL: <https://html.spec.whatwg.org/multipage/webstorage.html> (дата звернення: 10.05.2025).
16. Криптографія для розробників: Chapter 2. Jstify Community. Medium. URL: <https://surl.li/uvpyhi> (дата звернення: 14.04.2026).
17. Операції з даними (CRUD) – Про Програмування. ABout IT And Programming. URL: <https://abitap.com/1-5-operacziyi-z-danymy-crud/> (дата звернення: 21.03.2026).