**ДИПЛОМНА РОБОТА**

**ДР.122.042.012.ПЗ**

**Ковшун Максим Аналоійович**

**2025**

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

ЖИТОМИРСЬКИЙ АГРОТЕХНІЧНИЙ ФАХОВИЙ КОЛЕДЖ

ВІДДІЛЕННЯ «Інженерна інфраструктура та комп'ютерні науки»

ЦИКЛОВА КОМІСІЯ «Інженерна інфраструктура та комп'ютерні науки»

**ПОЯСНЮВАЛЬНА ЗАПИСКА**

до дипломної роботи освітнього ступеня «фаховий молодший бакалавр»

за спеціальністю 122 «Комп’ютерні науки»

(освітня програма 122 «Комп’ютерні науки»)

на тему:

**«Розробка калькулятора для фінансових розрахунків»**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Студента | 4 | курсу | П-42 | групи |
| Ковшуна Максима Анатолійовича |
| (ПІБ) |
| Керівник | Данилюк Ілля Володимирович |
| Рецензент | Лавріщев Олександр Олександрович |
|  |
| Національна шкала |  |
| Кількість балів |  | Оцінка: ECTS |  |
|  |  |  |  |
| Члени комісії |  | Ісаєв А.М. |
|  | (підпис) | (прізвище та ініціали) |
|  |  | Габрійчук Н.І. |
|  | (підпис) | (прізвище та ініціали) |
|  |  | Устименко Л.М. |
|  | (підпис) | (прізвище та ініціали)Устименко Я. І. |
|  | (підпис) | (прізвище та ініціали) |

Житомир – 2025

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

ЖИТОМИРСЬКИЙ АГРОТЕХНІЧНИЙ ФАХОВИЙ КОЛЕДЖ

ВІДДІЛЕННЯ «Інженерна інфраструктура та комп'ютерні науки»

ЦИКЛОВА КОМІСІЯ «Інженерна інфраструктура та комп'ютерні науки»

**«ЗАТВЕРДЖУЮ»**

Голова циклової комісії спеціальності

«Комп’ютерні науки»

 Діана ПАЛІЙ

«\_\_\_\_»\_\_\_\_\_\_\_\_\_\_\_\_\_202\_\_ р.

**ЗАВДАННЯ**

**на дипломну роботу**

Здобувач вищої освіти: **КОВШУН Максим Анатолійович**

Керівник роботи: **ДАНИЛЮК Ілля Володимирович**

Тема роботи: **«Розробка калькулятора для фінансових розрахунків»**, затверджена наказом закладу вищої освіти від «27» грудня 2025 р., №532у.

Вихідні дані для роботи: «розробка функціонального калькулятору для фінансових розрахунків з підтримкою конвертації валют в реальному часі»

|  |  |  |  |
| --- | --- | --- | --- |
| Розділ | Консультант | Завданнявидав | Завдання прийняв |
| 1 | Ілля ДАНИЛЮК  | 01-02-2025 | 01-02-2025 |
| 2 | Ілля ДАНИЛЮК  | 18-03-2025 | 18-03-2025 |
| 3 | Ілля ДАНИЛЮК  | 23-04-2025 | 23-04-2025 |

Консультанти з дипломної роботи із зазначенням розділів, що їх стосується:

**Календарний план**

|  |  |  |  |
| --- | --- | --- | --- |
| № з/п | Етап роботи | Термін виконання | Примітка |
| 1 |

|  |
| --- |
|  |

|  |
| --- |
| Формулювання мети, визначення завдань проєкту |

 | 01 лютого 2025 | Виконано |
| 2 | Планування структури застосунку та вибір технологій | 13 лютого 2025 | Виконано |
| 3 | Створення базового каркасу (React, Vite) | 02 березня 2025 | Виконано |
| 4 | Розробка дизайну інтерфейсу | 10 березня 2025 | Виконано |
| 5 | Реалізація калькулятора депозитів | 27 березня 2025 | Виконано |
| 6 | Створення калькулятора відсотків | 9 квітня 2025 | Виконано |
| 7 | Реалізація кредитного калькулятора | 22 квітня 2025 | Виконано |
| 8 | Інтеграція API конвертера валют | 29 квітня 2025 | Виконано |
| 9 | Тестування функціоналу, перевірка обрахунків та графіків | 9 травня 2025 | Виконано |
| 10 | Оптимізація, стилізація, збереження історії розрахунків | 19 травня 2025 | Виконано |
| 11 | Формування пояснювальної записки та оформлення проєкту | 29 травня 2025 | Виконано |
| 12 | Попередній захист роботи | 19 червня 2025 |  |

Здобувач фахової передвищої освіти\_\_\_\_\_\_\_\_\_\_ Максим КОВШУН

Керівник \_\_\_\_\_\_\_\_\_\_ Ілля ДАНИЛЮК

РЕФЕРАТ

Записка: 68 стор., 8 рис., 19 джерел.

Ключові слова: веб-застосунок, React, Vite, JavaScript, фінансові розрахунки, SPA, API, локальне збереження, адаптивний дизайн.

Об'єкт дослідження — автоматизація фінансових розрахунків і створення інтерактивних веб-інструментів для персонального фінансового планування.

Мета — розробити веб-застосунок фінансового калькулятора з точними розрахунками депозитів, кредитів, відсотків і валют.

Методи — аналіз існуючих рішень, проєктування компонентів, застосування React, Vite, математичне моделювання, інтеграція з API.

Результат — створено SPA з модулями депозитів, кредитів, відсотків і конвертації валют. Реалізовано історію розрахунків, експорт, графіки, адаптивний інтерфейс і кросбраузерну сумісність.

ABSTRACT

Report: 68 pages, 8 figures, 19 sources.

Keywords: web application, React, Vite, JavaScript, financial calculations, SPA, API, local storage, responsive design.

The study focuses on automating financial calculations and developing web tools for personal finance planning.

The goal is to build a financial calculator app for deposits, loans, interest, and currency conversion.

Methods include analysis, component design, React/Vite use, financial modeling, and API integration.

The result is a SPA with modules for deposits, loans, interest, and currency conversion, featuring history, export, responsive UI, charts, and fast cross-browser performance.

**ЗМІСТ**

[ВСТУП 5](#_Toc201167416)

[РОЗДІЛ 1. АНАЛІЗ ТЕХНОЛОГІЙ ДЛЯ РОЗРОБКИ КАЛЬКУЛЯТОРУ 6](#_Toc201167417)

[1.1. Постановка основної задачі проєкту 6](#_Toc201167418)

[1.2. Вибір технологій та інструментів 7](#_Toc201167419)

[Висновки до розділу 1 8](#_Toc201167420)

[РОЗДІЛ 2. ПРОЄКТУВАННЯ «КАЛЬКУЛЯТОР ДЛЯ ФІНАНСОВИХ ЗАСТОСУНКІВ» 9](#_Toc201167421)

[2.1. Аналіз вимог до застосунку 9](#_Toc201167422)

[2.2. Розробка загальної архітектури 13](#_Toc201167423)

[2.3. Нефункціональні вимоги 17](#_Toc201167424)

[2.4. Визначення вимог до користувацького інтерфейсу 20](#_Toc201167425)

[2.5. Дизайн інтерфейсу користувача 25](#_Toc201167426)

[Висновки до розділу 2 29](#_Toc201167427)

[РОЗДІЛ 3. РОЗРОБКА ЗАСТОСУНКА В JAVASCRIPT З ВИКОРИСТАННЯМ REACT 30](#_Toc201167428)

[3.1. Структура проєкту 30](#_Toc201167429)

[3.2. Сторінки застосунку та їх функціональне призначення 34](#_Toc201167430)

[3.3. Ключові програмні компоненти: Калькулятори та сервісна логіка 42](#_Toc201167431)

[ВИСНОВКИ 60](#_Toc201167432)

[СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ 62](#_Toc201167433)

[ДОДАТКИ 63](#_Toc201167434)

# ВСТУП

Сьогодні багато людей мають необхідність обраховувати купу фінансових питань - від вибору вигідного депозиту до оцінки кредитного навантаження. Водночас, не всі мають достатньо часу або знань, щоб самостійно проводити складні розрахунки або аналізувати фінансові показники. У таких випадках людям в нагоді стають калькулятори для фінансових розрахунків.

Метою цього проєкту стало створення простого, але функціонального веб-застосунку - фінансового калькулятора, який дозволяє користувачу швидко й точно розраховувати різні фінансові сценарії: прибуток за вкладом, розмір відсотків, платежі за кредитом, з урахуванням інфляції та актуального валютного курсу. Основний акцент зроблено на зручності, наочності та практичності: користувач може не лише побачити результат розрахунку, а й переглянути динаміку змін у вигляді графіків, зберегти свої розрахунки й повернутись до них пізніше.

У цьому документі описано весь шлях створення застосунку — від вибору технологій і проектування структури до реалізації основних функцій і побудови інтерфейсу. Результатом є готовий до використання калькулятор, який можна розгорнути у браузері без потреби у серверній частині.

# РОЗДІЛ 1. АНАЛІЗ ТЕХНОЛОГІЙ ДЛЯ РОЗРОБКИ КАЛЬКУЛЯТОРУ

## 1.1. Постановка основної задачі проєкту

Основною задачею проєкту є розробка калькулятору для фінансових розрахунків з інтуїтивно простим і зручним в керуванні інтерфейсом. Для реалізації проєкту потрібно було реалізувати такі функції:

Кредитний калькулятор: Розробка кредитного калькулятора з можливістю вибору платежу (Ануїтетний платіж/Диференційований платіж), який буде задовільняти потребу користувача, проводити обрахування і давати детальні результати користувачеві.

Депозитний калькулятор: Розробка модуля для розрахунку прибутковості банківського вкладу з урахуванням початкової суми, регулярних поповнень, процентної ставки, періоду розміщення, частоти капіталізації та інфляції. Калькулятор дозволяє проводити обрахунки як зі щомісячними внесками, так і без них, з можливістю вибору типу розміщення внесків (на початку або в кінці періоду). Результати виводяться у вигляді числових значень та графіків для зручності аналізу.

Інвестиційний калькулятор: Реалізація інструменту для аналізу фінансових результатів довгострокового інвестування із можливістю вибору типу відсотків (прості або складні), тривалості інвестування, внесків та інтервалу капіталізації. Калькулятор дозволяє визначити сумарний прибуток, величину інвестованих коштів, а також ефективність інвестиції з урахуванням реального приросту. Передбачено гнучке налаштування параметрів і графічну візуалізацію результатів.

Конвертер валют: Розробка модуля для швидкого перерахунку суми грошей між різними валютами на основі актуального обмінного курсу. Користувач має можливість обирати валютну пару (наприклад, EUR → USD), вводити бажану суму та миттєво отримувати результат. Калькулятор підтримує більшість основних валют і адаптований до змін ринкових курсів у режимі реального часу.

## 1.2. Вибір технологій та інструментів

У данному розділі будуть зкоротко зазначені основні технології та інструменти які використовувалися для реалізації проєкту

|  |  |  |
| --- | --- | --- |
| **Технологія / Інструмент** | Призначення  | Переваги використання |
| React | Основний фреймворк для створення SPA | Компонентна архітектура, швидкість роботи, велика екосистема |
| Vite | Інструмент збирання та запуску | Швидке збирання, моментальне оновлення в dev-середовищі |
| React Router | Клієнтська маршрутизація | Просте створення навігації між сторінками без перезавантаження |
| Recharts | Побудова графіків | Гнучка та проста у використанні бібліотека для візуалізації даних |
| CSS-файли | Стилізація інтерфейсу | Ізоляція стилів, легка підтримка, можливість адаптивної верстки |
| FreeCurrencyAPI | Отримання актуальних курсів валют | Безкоштовний API з підтримкою основних валют |
| JavaScript (ES6) | Мова реалізації логіки | Гнучка та динамічна мова з широкими можливостями обчислень |

## Висновки до розділу 1

Обраний комбінація технологій дозволяє розробити легкий та продуктивний клієнтський веб-застосунок без використання серверної інфраструктури. Комбінація «React» + «Vite» забезпечує швидкий старт проєкту, знижує використання ресурсів пристрою, на якому може використовуватися данний калькулятор та дозволяє гнучко допрацьовувати, розширювати чи дебажити проєкт при потребі. Бібліотека «Recharts» дозволяє застосунку виконувати просту і в одночас інформативну побудову графіків для візуальної демонстрації результатів. Завдяки використанню «localStorage» реалізовано збереження історії розрахунків без потреби в базі даних. Використання технології API дозволяє отримувати актуальний курс валют. Загалом вибір технологій повністю відповідає поставленим функціональним і нефункціональним вимогам проєкту.

# РОЗДІЛ 2. ПРОЄКТУВАННЯ «КАЛЬКУЛЯТОР ДЛЯ ФІНАНСОВИХ ЗАСТОСУНКІВ»

## 2.1. Аналіз вимог до застосунку

На етапі проєктування веб-застосунку важливо чітко визначити функціональні та нефункціональні вимоги до системи. Це дозволяє сформувати загальне уявлення про структуру, поведінку й можливості додатку ще до початку розробки, а також дає змогу уникнути непередбачених змін архітектури в майбутньому.

У цьому підрозділі подано основні функціональні та нефункціональні вимоги до розробки вебзастосунку фінансового калькулятора, реалізованого з використанням технологій «React», «Vite» та сторонніх API для фінансових даних.

**1. Функціональні вимоги**

Функціональні вимоги описують конкретні можливості, які повинен забезпечувати застосунок:

**1.1 Калькулятор депозитів:**

Модуль повинен обчислювати майбутню вартість вкладу з урахуванням:

початкової суми,

регулярних внесків,

відсоткової ставки,

періоду зберігання,

частоти капіталізації та інфляції.

Передбачено відображення результатів у числовому форматі та у вигляді графіка.

**1.2 Калькулятор відсотків:**

Забезпечує розрахунок як простих, так і складних відсотків. Можливість вибору типу нарахування, введення базових параметрів (сума, ставка, термін), а також перегляд реального приросту капіталу.

**1.3 Кредитний калькулятор:**

Дає змогу обирати між двома моделями погашення кредиту: ануїтетною та диференційованою. Обраховує щомісячний платіж, загальну суму переплати, графік змін боргового навантаження протягом часу.

**1.4 Конвертер валют:**

Отримання актуальних курсів валют за допомогою зовнішнього API (FreeCurrencyAPI). Підтримка базових валют, можливість двостороннього перерахунку.

**1.5 Збереження історії розрахунків:**

Останні обрахунки зберігаються в localStorage. Користувач може переглядати до 10 попередніх операцій, а також очищати або експортувати їх.

**1.6 Інтерфейс користувача (UI):**

Зручна навігація через «Sidebar», адаптивна верстка, інформативні форми з підказками, графіки результатів. Кожен розділ має окрему сторінку та стилізацію.

**2. Нефункціональні вимоги**

Нефункціональні вимоги визначають загальні характеристики проєкту, такі як сумісність з різними пристроями (комп'ютер, телефон, планшет), продуктивність та використання ресурсів додатком, гнучкість архітектури:

Продуктивність:

Застосунок повинен швидко завантажуватись, “миттєво” обраховувати значення. Всі розрахунки відбуваються на клієнтській стороні без потреби звернення до серверу (за винятком API для валют).

Масштабованість:

Архітектура повинна мати можливіть легко інтегрувати в застосунок нові типи калькуляторів(наприклад, пенсійний, іпотечний тощо), без зміни базової логіки або структури коду.

Кросбраузерність:

Додаток повинен працювати у всіх сучасних браузерах (Chrome, Firefox, Edge, Safari), а також мати адаптивний інтерфейс для планшетів і смартфонів.

Стилістична цілісність:

Весь інтерфейс має єдиний дизайн - структура сторінок, компоненти введення, кнопки та графіки стилізовані в єдиному візуальному стилі.

Використання сучасного стеку:

Для розробки застосовано «React» + «Vite», що забезпечує швидку розробку, гаряче оновлення, модульну структуру та зручне керування станами.

**3. Обмеження та технічні припущення**

Проєкт реалізується виключно у вигляді SPA (Single Page Application) без серверної частини.

Усі розрахунки виконуються на стороні клієнту з використанням мови програмування JavaScript.

Дані валют отримуються лише через веб-запит до FreeCurrencyAPI, без зберігання курсів на пристрої користувача.

Для графіків використовується бібліотека «Recharts».

Збереження історії реалізоване через «localStorage», без застосування баз даних чи хмарних сховищ.

Усі калькулятори реалізовані окремо - кожен представлений окремим React-компонентом.

## 2.2. Розробка загальної архітектури

Архітектура програмного забезпечення відіграє ключову роль у створенні будь-якого проєкту, оскільки вона визначає внутрішню організацію компонентів, їхні взаємозв'язки та принципи функціонування системи. У межах даного проєкту архітектура побудована на засадах компонентної структури «React», модульності та чіткого розподілу обов'язків між різними частинами системи. Такий підхід забезпечує не лише стабільне функціонування калькулятора, але й створює передумови для подальшого розвитку та масштабування застосунку.

**Фундаментальні архітектурні засади:**

Під час формування загальної архітектури проєкту було покладено в основу наступні принципи:

Компонентна модульність – кожен розрахунковий модуль (депозитний, кредитний, відсотковий калькулятор та конвертер валют) реалізовано як самостійний React-компонент із власною логікою обчислень та управління станом.

Відокремлення логіки від представлення – математичні розрахунки винесено в окремі сервісні модулі, що дозволяє тестувати та модифікувати обчислювальну логіку незалежно від інтерфейсу користувача.

Уніфікація кодової бази - застосування спільних стилістичних рішень, однотипних компонентів форм та єдиних підходів до валідації даних забезпечує узгодженість усього застосунку.

Реактивність та керування станом - використання «React hooks» (useState, useEffect) для управління локальним станом компонентів та автоматичного оновлення інтерфейсу при зміні даних.

Гнучкість розширення - архітектурні рішення дозволяють інтегрувати додаткові типи фінансових калькуляторів без перебудови існуючої структури проєкту.

**Структурні складові архітектури**

Архітектура вебзастосунку організована у вигляді багаторівневої системи з чітким розподілом функціональності:

**1. Презентаційний рівень (UI Components)**

Цей рівень включає всі візуальні компоненти застосунку:

основні сторінки калькуляторів (DepositCalculator, LoanCalculator, InterestCalculator, CurrencyConverter);

навігаційні елементи (Sidebar, Header);

форми введення даних з валідацією;

компоненти відображення результатів (графіки, таблиці);

елементи керування інтерфейсом (кнопки, поля введення).

Кожен компонент відповідає за власну область відповідальності та взаємодіє з іншими через передачу пропсів або контекст.

**2. Бізнес-логічний рівень (Services)**

Містить основну обчислювальну логіку застосунку:

2.1 currencyApi - сервіс для роботи з API курсів валют, включаючи обробку запитів та кешування результатів;

2.2 localStorage - модуль управління локальним збереженням історії розрахунків, експорту та імпорту даних;

2.3 Математичні функції розрахунку складних відсотків, ануїтетних платежів, конвертації валют;

2.4 Валідаційна логіка для перевірки коректності введених користувачем параметрів.

**3. Рівень управління даними (State Management)**

Реалізований через локальний стан компонентів та включає:

3.1 Збереження параметрів розрахунків (суми, ставки, терміни);

3.2 Кешування результатів обчислень для оптимізації продуктивності;

3.3 Управління станами завантаження та помилок;

3.4 Синхронізацію даних між різними компонентами через «localStorage».

**4. Інфраструктурний рівень (External Integration)**

Забезпечує взаємодію з зовнішніми системами:

4.1 Інтеграція з FreeCurrencyAPI для отримання актуальних валютних курсів;

4.2 Обробка мережевих запитів та помилок з'єднання;

4.3 Управління конфігурацією API-ключів та параметрів підключення.

**Патерни проєктування та архітектурні рішення:**

У процесі розробки було застосовано кілька ключових патернів:

1. Композиція компонентів - складні інтерфейси побудовані з простих, багаторазово використовуваних компонентів. Наприклад, форми введення даних використовують спільні компоненти полів вводу, кнопок та блоків валідації.

2. Інверсія залежностей - компоненти верхнього рівня передають функції-обробники подій дочірнім компонентам, що забезпечує гнучкість та тестованість коду.

3. Стратегія - для різних типів розрахунків (простий/складний відсоток, ануїтетний/диференційований платіж) використовуються різні алгоритми, які можуть змінюватися динамічно залежно від вибору користувача.

4. Фасад - сервісні модулі приховують складність внутрішніх операцій та надають простий інтерфейс для компонентів представлення.

**Організація міжкомпонентної взаємодії**

1. Взаємодія між компонентами організована за принципом однонаправленого потоку даних:

2. Користувач вводить параметри через форми інтерфейсу

3. Дані валідуються на рівні компонента

4. Коректні параметри передаються до обчислювальних функцій

5. Результати розрахунків оновлюють стан компонента

6. Інтерфейс автоматично перерендерюється для відображення нових даних

7. Історія розрахунків зберігається в «localStorage» для подальшого використання

Така архітектура забезпечує передбачувану поведінку застосунку та спрощує процеси тестування й подальшої підтримки коду. Модульна структура дозволяє незалежно розвивати окремі частини системи, а чіткий розподіл відповідальностей мінімізує ризик виникнення помилок при модифікації функціональності.

## 2.3. Нефункціональні вимоги

Нефункціональні вимоги визначають загальні характеристики вебзастосунку, які не пов’язані безпосередньо з функціональністю, проте суттєво впливають на зручність використання, якість реалізації, ефективність роботи, довговічність та сприйняття кінцевим користувачем. Для проєкту сформовано наступні ключові вимоги:

**1. Швидкодія та відгук системи**

Фінансовий калькулятор має забезпечувати максимально швидкий відгук на дії користувача. Усі розрахунки повинні відбуватись миттєво на клієнтській стороні, без потреби надсилання даних на сервер.
Час обрахунку результату після введення даних не перевищує 100 мс.

Перемикання між модулями калькулятора повинно здійснюватись без перезавантаження сторінки завдяки SPA-архітектурі.

Відображення графіків має бути плавним, без візуальних затримок чи ривків.

**2. Надійність інтерфейсу та перевірка введених даних**

Оскільки додаток працює з фінансовими обчисленнями, критично важливо забезпечити стабільну поведінку у випадках помилкового або неповного вводу.

Реалізована система валідації на рівні форм, яка не допускає обрахунків із некоректними або неповними даними.

Усі поля мають дефолтні значення для уникнення непередбачуваної поведінки.

Передбачено обробку крайніх випадків (наприклад, нульова відсоткова ставка, від’ємні значення, пусті поля).

**3. Простота використання та доступність**

Інтерфейс застосунку має бути зрозумілим навіть для користувачів без спеціальної технічної або фінансової підготовки.

Усі поля введення супроводжуються підписами та одиницями вимірювання (наприклад, %, місяці, грн).

Користувач завжди бачить результати в читабельному форматі з автоматичним округленням до 2–4 знаків.

Модулі навігації (сайдбар, кнопки повернення, перемикання калькуляторів) реалізовані у звичному та очікуваному вигляді.

**4. Модульність та підтримка розширень**

Проєкт має бути зручним для подальшого розширення та масштабування.

Кожен тип калькулятора (депозит, кредит, інвестиції, валюта) реалізований як окремий компонент із власною логікою.

Усі обчислення винесені в сервісні функції, що дозволяє додавати нові формули без зміни інтерфейсу.

Можливе додавання нових типів калькуляторів (наприклад, іпотечний, пенсійний) без порушення структури застосунку.

**5. Візуальна узгодженість і графічна подача результатів**

Для полегшення аналізу результатів реалізована візуалізація обчислень у вигляді графіків.

Для кожного типу розрахунку (накопичення, кредит, відсотки) будуються відповідні лінійні або стовпчикові діаграми.

Графіки автоматично оновлюються після кожного нового введення.

Стилістика інтерфейсу — мінімалістична, сучасна, без зайвих елементів, з акцентом на інформаційність.

**6. Адаптивність**

Застосунок повинен коректно відображатися та працювати на різних пристроях і розмірах екранів.

Всі компоненти побудовані з урахуванням адаптивної верстки.

Сайдбар, поля введення та графіки масштабуються відповідно до екрана.

Підтримка управління як мишею, так і сенсорними елементами (на планшетах, телефонах).

**7. Незалежність від сервера та збереження локальних даних**

Застосунок не потребує серверної частини, що забезпечує простоту розгортання та збереження даних користувача лише на локальному пристрої.

Дані останніх 10 розрахунків зберігаються в localStorage і можуть бути відновлені після оновлення сторінки.

Збереження повністю автоматизоване — користувач не взаємодіє з файлами або формами.

Дані не надсилаються стороннім сервісам (за винятком запиту до FreeCurrencyAPI).

## 2.4. Визначення вимог до користувацького інтерфейсу

Користувацький інтерфейс (UI) є ключовим елементом у забезпеченні ефективної, комфортної та інтуїтивно зрозумілої взаємодії користувача із вебзастосунком. Оскільки проєкт «Фінансовий калькулятор» орієнтований на кінцевого користувача без спеціальної фінансової чи технічної підготовки, інтерфейс має бути простим, логічним і візуально привабливим. У цьому розділі сформульовано вимоги до структури, логіки та візуальної реалізації інтерфейсу.

**1. Загальні вимоги до візуального стилю**

Мінімалізм і читабельність: Інтерфейс побудований за принципом “нічого зайвого” — чітка типографіка, світла кольорова схема, добре організовані поля введення і блоки виводу. Акценти зроблено на ключових елементах (результат, формула, графік).

Однорідність елементів: Всі калькулятори побудовані за єдиним принципом — однакова структура сторінки, форма введення зліва, результат та графіки — справа. Це підвищує зручність при перемиканні між модулями.

Колірне кодування результатів: Вивід фінансових результатів використовує кольорові підказки: зелений — прибуток, помаранчевий — інфляція, синій — вкладення. Графіки синхронізовані за кольорами з результатами.

**2. Вимоги до Sidebar-навігації**

Sidebar розташований ліворуч і завжди доступний на всіх сторінках.

Складається з логотипу застосунку та чотирьох навігаційних пунктів:

* Кредитний калькулятор
* Депозитний калькулятор
* Відсотковий калькулятор
* Конвертер валют

Активний пункт підсвічується кольором і іконкою. Іконки підібрані відповідно до тематики кожного модуля.

Sidebar згортається на мобільних пристроях автоматично.

**3. Вимоги до форм введення**

Поля введення мають чітке позначення: назва параметра, одиниці вимірювання, тип значення.

Передбачено підказки у вигляді тексту під формами (наприклад, “Річна процентна ставка (%)”).

Для вибору валют, одиниць часу, частоти капіталізації використовуються випадаючі списки (<select>).

При включенні/виключенні інфляції або щомісячних внесків активуються відповідні підполя.

Дані вводяться вручну або вибираються з випадачки; за помилки введення — підсвічуються червоним та блокують обрахунок.

**4. Вимоги до блоку результатів**

Результати обчислення виводяться праворуч від форми в окремому блоці «Прогноз зростання» / «Результати розрахунку».

Структура виводу:

Загальна сума

Отримані відсотки

Реальна вартість (з урахуванням інфляції)

Порівняння типів відсотків (для інвест. калькулятора)

Щомісячний платіж і переплата (для кредитного)

Результати підсвічуються зеленим кольором, мають фіксовану кількість знаків після коми.

Кнопка “Експорт” дозволяє зберегти розрахунок у файл.

**5. Вимоги до графіків і таблиць**

Графіки будуються з використанням Recharts і відображають:

Накопичення депозиту

Зростання боргу та платежів

Порівняння складних/простих відсотків

У кожному модулі є кнопка перемикання між графіком і табличним виводом результатів.

Графіки мають легенду, кольорову сітку та підпис осей. Дані автоматично оновлюються при зміні параметрів.

**6. Вимоги до історії розрахунків**

Під основним калькулятором реалізовано компонент “Історія розрахунків”.

Виводяться до 10 останніх обрахунків із зазначенням дати, суми, відсотка і тривалості.

Користувач може:

Завантажити результат

Видалити запис

Очистити всю історію

Імпортувати/експортувати історію в JSON

**7. Вимоги до адаптивності**

На мобільних пристроях Sidebar згортається в іконку меню.

Всі блоки калькулятора розташовуються вертикально — спочатку форма, потім результати.

Графіки масштабуються автоматично та не обрізаються.

Елементи введення мають збільшений розмір і клік-зони.

 **8. Додаткові елементи взаємодії**

Формули обчислень розміщені внизу сторінки для прозорості логіки (натхненно принципами open source).

Усі кнопки мають візуальний зворотний зв’язок (hover-ефекти, зміна кольору при натисканні).

Інтерактивні елементи (чекбокси, селектори) мають інтуїтивно зрозумілий вигляд.

## 2.5. Дизайн інтерфейсу користувача

Дизайн інтерфейсу користувача (UI) у є одним із найважливіших чинників, що визначає зручність використання, доступність функцій і зрозумілість представлених фінансових результатів. Візуальне оформлення застосунку має не лише бути естетичним, а й підтримувати функціональну логіку: розділення інформаційних блоків, акценти на результатах і підказки для користувача.

**1. Візуальна концепція**

Стиль оформлення:

Додаток виконаний у стилі сучасного мінімалізму з акцентом на чистоту, контрастність і простоту. Світла кольорова палітра з м’якими акцентами (блакитний, зелений, сірий) дозволяє сфокусувати увагу користувача на даних і результатах.

Кольорові акценти:

Основні результати (напр., фінальний баланс, реальна вартість, відсотки) підсвічуються зеленим кольором. Додаткові дані (інфляція, внески) виділяються помаранчевим, синім або сірим.

Інформаційна ієрархія:

Вся інформація в інтерфейсі структурована за принципом «зліва — введення, справа — результат». Це відповідає звичному порядку взаємодії в західних UI-патернах (LTR)

**2. Дизайн навігації (Sidebar)**

Sidebar розміщено зліва. Він постійно доступний на сторінці й дозволяє перемикатися між усіма розділами застосунку:

* Кредитний калькулятор
* Депозитний калькулятор
* Відсотковий калькулятор
* Конвертер валют

Для кожного розділу передбачено іконку та текстову мітку. Активний розділ підсвічується (блакитним кольором із фоном).

Sidebar автоматично згортається на мобільних пристроях, перетворюючись у вертикальне меню-гамбургер.

**3. Дизайн форм і блоків введення**

Поля введення (input/select) мають чітку структуру:

* Назва параметра
* Одиниця виміру (у placeholder або label)
* Валідація (від’ємні значення блокуються)
* Місце для формул і підказок нижче поля

Кнопки (наприклад, “Розрахувати”, “Експорт”) мають сучасний вигляд: закруглені, кольорові, з візуальним зворотним зв’язком (hover/active-ефекти).

Блок «Формула» розташовано під формою і дозволяє користувачеві перевірити, яка саме формула використовується для обчислень. Це сприяє прозорості і довірі до результату.

**4. Дизайн секції результатів**

Результати завжди розміщено праворуч від основної форми, у виділеному блоці з м’яким фоном і великим шрифтом.

У кожному калькуляторі структура результатів індивідуальна:

* Кредитний — щомісячний платіж, загальні відсотки, сума платежів.
* Депозитний/Відсотковий — фінальний баланс, відсотки, реальна вартість.
* Інвестиційний — порівняння простого і складного відсотку.

Усі результати легко копіюються, підкріплені валютними позначками, і можуть бути експортовані через кнопку «Експорт».

**5. Дизайн графіків та виводу**

Графіки реалізовано через бібліотеку «Recharts». Вони мають адаптивну ширину, легенду та маркування по осях.

Є перемикач між графіком та таблицею, який дозволяє користувачеві обрати зручніший формат перегляду.

У кредитному калькуляторі доступні два графіки одночасно:

* Розподіл платежів
* Залишок заборгованості

Колірні коди графіків відповідають тим, що використовуються у блоці результатів.

**6. Дизайн історії розрахунків**

Розташовано під формою. Кожен запис історії оформлено у вигляді «картки» з короткою інформацією (дата, параметри розрахунку).

Передбачено дії:

* Завантажити (відкрити знову)
* Видалити
* Очистити все
* Імпорт / експорт в JSON

**7. Адаптивність та доступність**

Усі елементи мають достатню контрастність для комфортного перегляду навіть при слабкому освітленні.

Шрифти адаптуються до ширини екрану.

На мобільних пристроях компоненти розміщуються вертикально:

1. Спочатку форма
2. Потім результати
3. Далі — графік або історія

Всі кнопки й поля введення мають достатній розмір для взаємодії пальцем на сенсорних екранах.

**8. Додаткові візуальні особливості**

Використано плавні анімації появи блоків результатів і оновлення графіків (зменшує ефект «рикань»).

Hover-ефекти для всіх інтерактивних елементів (кнопки, історія, меню).

Весь застосунок підтримує єдину стилістику, незалежно від розділу калькулятора (крім конвертера валют, який має трохи відмінне візуальне компонування).

## Висновки до розділу 2

У цьому розділі було детально проаналізовано, як саме має виглядати майбутній вебзастосунок — від функціоналу до зовнішнього вигляду. Спочатку були сформовані всі основні вимоги: що саме має вміти калькулятор, які типи обчислень потрібні, яким має бути інтерфейс, наскільки швидко все має працювати й наскільки просто користувачеві має бути з цим всім взаємодіяти. Після цього визначено, як саме ці вимоги реалізовуватимуться — спроєктовано структуру застосунку, поділено логіку на окремі частини, описано, як будуть виглядати сторінки, і що саме буде бачити й використовувати користувач. Важливим моментом стало також те, що логіка обчислень була винесена окремо від інтерфейсу — це дозволяє в майбутньому легко додавати нові типи калькуляторів або змінювати формули без потреби переписувати дизайн. Окрему увагу приділено зручності: інтерфейс простий, зрозумілий, з фокусом на те, щоб користувач швидко отримував потрібні дані без зайвих кроків. Загалом, усе, що описано в цьому розділі, стало основою для подальшої реалізації застосунку — логічної, надійної та зручно структурованої..

# РОЗДІЛ 3. РОЗРОБКА ЗАСТОСУНКА В JAVASCRIPT З ВИКОРИСТАННЯМ REACT

## 3.1. Структура проєкту

У проєкті, реалізованому за допомогою бібліотеки React та інструменту збирання Vite, особливу увагу було приділено чіткій модульній структурі, яка дозволяє ефективно масштабувати застосунок, розділяти логіку, стилі та компоненти за призначенням, а також забезпечити простоту супроводу коду.

Основною директорією розробки є src/, в якій організовано всі ключові елементи застосунку. Нижче наведено короткий опис кожної логічної частини:

**1. src/components/**

Ця папка містить повторно використовувані елементи інтерфейсу, які не залежать від логіки окремих калькуляторів, а застосовуються на рівні всього додатку:

Sidebar.jsx — компонент навігаційної панелі, яка забезпечує доступ до всіх розділів застосунку.

Footer.jsx — підвал сторінки, який може містити авторські дані або посилання.

Для кожного з компонентів створено окремі модульні стилі: Sidebar.module.css, Footer.module.css.

**2. src/layout/**

Цей розділ відповідає за загальне компонування сторінок (layout):

HomeLayout.jsx — обгортка для головного контенту, яка включає sidebar, header, основну область контенту.

Стилі компонента винесено у HomeLayout.module.css.

**3.1. src/pages/**

У папці pages/ розміщені основні функціональні сторінки калькулятора, кожна з яких відповідає за окремий розрахунковий модуль. Кожен файл реалізований як React-компонент:

* DepositCalculator.jsx — депозитний калькулятор.
* LoanCalculator.jsx — кредитний калькулятор.
* InterestCalculator.jsx — відсотковий калькулятор
* CurrencyConverter.jsx — конвертер валют.

Кожна сторінка містить форму введення даних, блок виводу результатів, а також (де потрібно) графік або таблицю результатів.

**3.2. src/pages/styles/**

Кожна сторінка має власний ізольований модуль стилів CSS, що дозволяє уникати конфліктів між різними елементами інтерфейсу. Назви стилів відповідають сторінкам, наприклад:

* DepositCalculator.module.css
* LoanCalculator.module.css
* CurrencyConverter.module.css
* InterestCalculator.module.css

**4. src/services/**

Цей каталог містить функціональні модулі логіки та роботи з даними:

currencyApi.js — логіка отримання валютних курсів із FreeCurrencyAPI.

localStorage.js — обробка історії розрахунків (збереження, імпорт/експорт, очищення).

Це дає змогу відокремити бізнес-логіку від UI, що спрощує тестування й обслуговування.

**5. src/App.jsx, src/main.jsx**

App.jsx — головний кореневий компонент застосунку, у якому визначається маршрутизація за допомогою React Router.

main.jsx — точка входу до застосунку, в якій рендериться <App /> до DOM.

**6. src/index.css, src/App.css**

Файли зі спільними стилями або базовими CSS-налаштуваннями. За потреби можуть бути замінені повністю на CSS-модулі.

Загальна схема структури:


(Рис 3.1)

## 3.2. Сторінки застосунку та їх функціональне призначення

У веб-застосунку кожна сторінка реалізована як окремий функціональний модуль формату SPA (Single Page Application). Вони відповідають за виконання конкретних розрахунків, надають інтерфейс введення параметрів, виводять результати та забезпечують навігацію між розділами. Усі сторінки пов’язані між собою через компонент Sidebar, що дозволяє швидко переходити між модулями без перезавантаження сторінки.

У застосунку реалізовано такі основні сторінки:

**1. Депозитний калькулятор (DepositCalculator)**

Призначення:

Обчислення майбутньої вартості накопичень із урахуванням відсоткової ставки, інфляції, капіталізації та щомісячних внесків.

Зміст:

Форма введення параметрів: початковий депозит, щомісячні внески, ставка, термін, інфляція, частота капіталізації.

Перемикач внесків на початку або в кінці місяця.

Блок результатів (фінальна сума, загальний вклад, отримані відсотки, реальна вартість).

Кнопки експорту розрахунку.

Вибір між графіком і табличним поданням результатів.

Формула обчислень у нижній частині сторінки.

Історія попередніх розрахунків.

Функціонал:

Реалізує повноцінну симуляцію зростання депозиту та надає користувачу наочні графіки результатів. Дані зберігаються у localStorage.



(Рис 3.2.1)


(Рис 3.2.2)

**2. Відсотковий калькулятор (InterestCalculator)**

Призначення:

Порівняння простого та складного відсоткового нарахування на вкладені кошти.

Зміст:

Форма введення: сума, відсоткова ставка, термін, кількість капіталізацій на рік.

Перемикач типу відсотку (простий/складний).

Порівняння результатів: фінальна сума, різниця між простим та складним відсотком.

Блок із формулами та поясненням.

Графік зростання капіталу з обома варіантами для візуального порівняння.

Функціонал:

Дає змогу користувачу зрозуміти різницю між видами відсоткового нарахування й оцінити довгостроковий вплив складного відсотку.


(Рис 3.2.3)


(Рис 3.2.4)

**3. Кредитний калькулятор (LoanCalculator)**

Призначення:

Розрахунок кредитних платежів за двома схемами: ануїтетною та диференційованою.

Зміст:

Форма введення: сума кредиту, термін, річна ставка, тип платежу.

Вивід результатів:

Щомісячний платіж (при ануїтетах — фіксований).

Загальні відсотки.

Загальна сума платежів.

Два графіки:

Розбивка платежів по місяцях.

Залишок заборгованості.

Можливість експорту результатів.

Функціонал:

Калькулятор дозволяє порівнювати обидві схеми погашення, наочно бачити, як змінюється платіж у диференційованій моделі, та загальне навантаження за весь період.


(Рис 3.2.5)

**4. Конвертер валют (CurrencyConverter)**

Призначення:

Швидкий перерахунок між обраними валютами на основі актуального курсу.

Зміст:

Поля для введення суми та вибору пар валют.

Отримання даних з FreeCurrencyAPI.

Виведення конвертованої суми та курсу обміну.

Функціонал:

Конвертер реалізовано у спрощеному дизайні без графіків. Він дозволяє отримати результат в реальному часі та підтримує основні валюти. Курс оновлюється при кожному запиті.


(Рис 3.2.5)

**5. Історія розрахунків**

Призначення:

Надає користувачу доступ до попередніх 10 операцій для повторного перегляду або збереження.

Зміст:

Картки з параметрами розрахунку (дата, сума, ставка, термін).

Кнопки:

Завантажити — повторити розрахунок.

Видалити — очистити окремий запис.

Експортувати/Імпортувати — зберегти або відновити всі розрахунки.

Функціонал:

Забезпечує персоналізовану взаємодію, дозволяє працювати з історичними даними та повторно використовувати введені параметри.


(Рис 3.2.6)

## 3.3. Ключові програмні компоненти: Калькулятори та сервісна логіка

Цей розділ містить опис основних функціональних одиниць застосунку, які забезпечують взаємодію з користувачем, виконують фінансові обчислення, будують графіки, працюють із даними та обробляють API-запити. Компоненти організовано у вигляді окремих сторінок (React-компонентів) та сервісних модулів, що формують основу логіки програми.

**3.3.1. Сторінка «DepositCalculator.jsx»**

Цей компонент відповідає за обчислення приросту капіталу при розміщенні коштів на депозит з урахуванням капіталізації, регулярних внесків, інфляції та тривалості вкладу. Калькулятор дозволяє вказати частоту капіталізації, періодичність внесків, їхню початкову точку (на початку чи в кінці періоду) та виводить точний прогноз накопичень.

Інтерфейс реалізовано у вигляді окремої сторінки з формою вводу параметрів зліва, та результатами обчислень праворуч. Результати відображаються як у вигляді фінансових підсумків (фінальний баланс, вкладені кошти, отримані відсотки, реальна вартість), так і графічно — у вигляді стовпчикового або лінійного графіка. Формули, які використовуються для обчислення, подані внизу сторінки, що дозволяє користувачу бачити, як саме виконується розрахунок. Усі обчислення локальні та миттєві.

Код до файлу «DepositCalculator.jsx»

**3.3.2. Сторінка «InterestCalculator.jsx»**

Цей модуль реалізує калькулятор порівняння простих і складних відсотків. Користувач вводить основні параметри (початкову суму, термін, ставку, періодичність капіталізації), після чого додаток розраховує обидва варіанти обчислення — за класичною та складеною формулою.

Окрім числових результатів (фінальна сума, відсотки, різниця), калькулятор візуалізує результат на графіку для наочного порівняння. Також відображається порівняльна таблиця, що дозволяє користувачу оцінити реальний вплив складного відсотку. Форма, логіка, історія і вивід результатів реалізовані повністю автономно.

**3.3.3. Сторінка «LoanCalculator.jsx»**

Цей компонент реалізує повноцінний кредитний калькулятор з підтримкою двох типів платежів: ануїтетного та диференційованого. Користувач може ввести суму кредиту, річну ставку, термін у місяцях і обрати тип розрахунку.

Після обчислення виводиться докладна інформація: щомісячний платіж, загальна сума виплат, переплата (відсотки), а також два окремих графіки:

* розбиття платежів за перші 12 місяців;
* динаміка залишку боргу протягом усього терміну кредитування.

Уся логіка розрахунку реалізована локально, з використанням точних фінансових формул. Калькулятор підтримує збереження історії розрахунків.

**3.3.4. Сторінка «CurrencyConverter.jsx»**

Цей компонент надає користувачу можливість швидко конвертувати валюту за актуальним курсом. Для цього реалізовано інтеграцію з API — FreeCurrencyAPI, що дозволяє динамічно отримувати курси валют.

Користувач обирає базову та цільову валюту, вводить суму, і одразу бачить результат конвертації. Сторінка має мінімалістичний інтерфейс і не включає графіків або історії — її функціонал сконцентрований на простоті та швидкості операцій.

Код до файлу «CurrencyConverter.jsx»

import React, { useState, useEffect } from 'react'

import { currencyApi } from '../services/currencyApi'

import styles from './styles/CurrencyConverter.module.css'

const CurrencyConverter = () => {

  const [formData, setFormData] = useState({

    amount: '',

    fromCurrency: 'USD',

    toCurrency: 'EUR'

  })

  const [exchangeRates, setExchangeRates] = useState({})

  const [convertedAmount, setConvertedAmount] = useState(0)

  const [loading, setLoading] = useState(false)

  const [error, setError] = useState(null)

  const [lastUpdated, setLastUpdated] = useState(null)

  const [currencies, setCurrencies] = useState([])

  // Валюти за замовчуванням (резерв, якщо API не працює)

  const defaultCurrencies = [

    { code: 'USD', name: 'Долар США' },

    { code: 'EUR', name: 'Євро' },

    { code: 'GBP', name: 'Британський фунт' },

    { code: 'JPY', name: 'Японська єна' },

    { code: 'CAD', name: 'Канадський долар' },

    { code: 'AUD', name: 'Австралійський долар' },

    { code: 'CHF', name: 'Швейцарський франк' },

    { code: 'CNY', name: 'Китайський юань' },

    { code: 'INR', name: 'Індійська рупія' },

    { code: 'KRW', name: 'Південнокорейський вон' },

    { code: 'MXN', name: 'Мексиканський песо' },

    { code: 'SGD', name: 'Сінгапурський долар' },

    { code: 'NOK', name: 'Норвезька крона' },

    { code: 'SEK', name: 'Шведська крона' },

    { code: 'DKK', name: 'Данська крона' },

    { code: 'PLN', name: 'Польський злотий' }

  ]

  useEffect(() => {

    const loadCurrencies = async () => {

      try {

        const response = await currencyApi.getSupportedCurrencies()

        if (response?.data) {

          const currencyList = Object.entries(response.data).map(([code, details]) => ({

            code,

            name: details.name || code

          }))

          setCurrencies(currencyList)

        } else {

          setCurrencies(defaultCurrencies)

        }

      } catch (error) {

        console.error('Не вдалося завантажити валюти:', error)

        setCurrencies(defaultCurrencies)

      }

    }

    loadCurrencies()

  }, [])

  useEffect(() => {

    if (formData.fromCurrency && formData.toCurrency) {

      fetchExchangeRates()

    }

  }, [formData.fromCurrency, formData.toCurrency])

  useEffect(() => {

    if (formData.amount && exchangeRates[formData.toCurrency]) {

      const converted = currencyApi.convertAmount(formData.amount, exchangeRates[formData.toCurrency])

      setConvertedAmount(converted)

    } else {

      setConvertedAmount(0)

    }

  }, [formData.amount, exchangeRates, formData.toCurrency])

  const fetchExchangeRates = async () => {

    setLoading(true)

    setError(null)

    try {

      const response = await currencyApi.getLatestRates(

        formData.fromCurrency,

        [formData.toCurrency]

      )

      if (response?.data) {

        setExchangeRates(response.data)

        setLastUpdated(new Date().toLocaleString())

      }

    } catch (error) {

      setError('Не вдалося отримати курси валют. Спробуйте ще раз.')

      console.error('Помилка отримання курсу валют:', error)

    } finally {

      setLoading(false)

    }

  }

  const handleInputChange = (e) => {

    const { name, value } = e.target

    setFormData(prev => ({

      ...prev,

      [name]: value

    }))

  }

  const handleSwapCurrencies = () => {

    setFormData(prev => ({

      ...prev,

      fromCurrency: prev.toCurrency,

      toCurrency: prev.fromCurrency

    }))

    setConvertedAmount(0)

  }

  const handleConvert = async (e) => {

    e.preventDefault()

    if (!formData.amount) {

      setError('Будь ласка, введіть суму для конвертації')

      return

    }

    await fetchExchangeRates()

  }

  const getCurrentRate = () => {

    return exchangeRates[formData.toCurrency] || 0

  }

  return (

    <div className={styles.container}>

      <div className={styles.header}>

        <h1 className={styles.title}>Конвертер валют</h1>

        <p className={styles.description}>

          Конвертуйте між різними валютами з актуальними курсами

        </p>

      </div>

      <div className={styles.content}>

        <div className={styles.formSection}>

          {error && (

            <div className={styles.errorMessage}>

              <span className={styles.errorIcon}>⚠️</span>

              {error}

            </div>

          )}

          <form onSubmit={handleConvert} className={styles.form}>

            <div className={styles.inputGroup}>

              <label htmlFor="amount" className={styles.label}>

                Сума

              </label>

              <input

                type="number"

                id="amount"

                name="amount"

                value={formData.amount}

                onChange={handleInputChange}

                className={styles.input}

                placeholder="Введіть суму для конвертації"

                min="0"

                step="0.01"

              />

            </div>

            <div className={styles.currencyRow}>

              <div className={styles.currencyGroup}>

                <label htmlFor="fromCurrency" className={styles.label}>

                  З

                </label>

                <select

                  id="fromCurrency"

                  name="fromCurrency"

                  value={formData.fromCurrency}

                  onChange={handleInputChange}

                  className={styles.select}

                >

                  {currencies.map((currency) => (

                    <option key={currency.code} value={currency.code}>

                      {currency.code} - {currency.name}

                    </option>

                  ))}

                </select>

              </div>

              <button

                type="button"

                onClick={handleSwapCurrencies}

                className={styles.swapButton}

                title="Поміняти валюти місцями"

                disabled={loading}

              >

                ⇄

              </button>

              <div className={styles.currencyGroup}>

                <label htmlFor="toCurrency" className={styles.label}>

                  В

                </label>

                <select

                  id="toCurrency"

                  name="toCurrency"

                  value={formData.toCurrency}

                  onChange={handleInputChange}

                  className={styles.select}

                >

                  {currencies.map((currency) => (

                    <option key={currency.code} value={currency.code}>

                      {currency.code} - {currency.name}

                    </option>

                  ))}

                </select>

              </div>

            </div>

            <button

              type="submit"

              className={styles.convertButton}

              disabled={loading || !formData.amount}

            >

              {loading ? 'Конвертуємо...' : 'Конвертувати валюту'}

            </button>

          </form>

          <div className={styles.quickConvert}>

            <h3 className={styles.quickTitle}>Швидка конвертація</h3>

            <div className={styles.quickButtons}>

              <button

                type="button"

                className={styles.quickButton}

                onClick={() => setFormData(prev => ({ ...prev, amount: '100' }))}

              >

                100

              </button>

              <button

                type="button"

                className={styles.quickButton}

                onClick={() => setFormData(prev => ({ ...prev, amount: '1000' }))}

              >

                1000

              </button>

              <button

                type="button"

                className={styles.quickButton}

                onClick={() => setFormData(prev => ({ ...prev, amount: '10000' }))}

              >

                10000

              </button>

            </div>

          </div>

        </div>

        <div className={styles.resultSection}>

          <h2 className={styles.resultTitle}>Результат конвертації</h2>

          <div className={styles.conversionDisplay}>

            <div className={styles.conversionItem}>

              <span className={styles.amount}>{formData.amount || '0.00'}</span>

              <span className={styles.currency}>{formData.fromCurrency}</span>

            </div>

            <div className={styles.conversionArrow}>→</div>

            <div className={styles.conversionItem}>

              <span className={styles.amount}>{convertedAmount}</span>

              <span className={styles.currency}>{formData.toCurrency}</span>

            </div>

          </div>

          <div className={styles.exchangeRate}>

            <div className={styles.rateInfo}>

              <span className={styles.rateLabel}>Курс обміну:</span>

              <span className={styles.rateValue}>

                1 {formData.fromCurrency} = {getCurrentRate().toFixed(4)} {formData.toCurrency}

              </span>

            </div>

            {lastUpdated && (

              <p className={styles.lastUpdated}>Останнє оновлення: {lastUpdated}</p>

            )}

          </div>

          <div className={styles.rateStatus}>

            {loading ? (

              <div className={styles.loadingIndicator}>

                <span className={styles.spinner}>⏳</span>

                <span>Отримуємо актуальні курси...</span>

              </div>

            ) : exchangeRates[formData.toCurrency] ? (

              <div className={styles.statusSuccess}>

                <span className={styles.statusIcon}>✅</span>

                <span>Курси успішно оновлено</span>

              </div>

            ) : (

              <div className={styles.statusWarning}>

                <span className={styles.statusIcon}>⚠️</span>

                <span>Оберіть валюти для отримання курсів</span>

              </div>

            )}

          </div>

          {(!formData.amount && !loading) && (

            <div className={styles.placeholder}>

              <p>💱 Введіть суму та оберіть валюти для конвертації</p>

              <p className={styles.placeholderNote}>

                Працює на FreeCurrencyAPI для актуальних курсів валют

              </p>

            </div>

          )}

        </div>

      </div>

    </div>

  )

}

export default CurrencyConverter

**3.3.5. Сервісний модуль «currencyApi.js»**

Модуль currencyApi.js відповідає за всю логіку взаємодії з зовнішнім API FreeCurrencyAPI. Він містить три основні методи:

* getLatestRates — отримання актуального курсу обраних валют відносно базової (за замовчуванням — USD). Метод підтримує передачу базової валюти та масиву цільових валют.
* getSupportedCurrencies — повертає перелік доступних для обміну валют.
* convertAmount — допоміжна функція для обчислення суми після конвертації (множення на курс із точністю до двох знаків після коми).

Усі методи побудовані на fetch, мають обробку помилок, вивід повідомлень у консоль у разі збоїв і забезпечують стабільну взаємодію з API навіть при частковій недоступності сервісу.

Код до файлу «currency.Api»

const API\_KEY = 'fca\_live\_PFGIuIHkPZ8urUCaWuEEkebAKWZbdWKdA0Yjvm5y'

const BASE\_URL = 'https://api.freecurrencyapi.com/v1'

export const currencyApi = {

  // Отримати останні курси валют

  async getLatestRates(baseCurrency = 'USD', currencies = []) {

    try {

      let url = `${BASE\_URL}/latest?apikey=${API\_KEY}`

      if (baseCurrency && baseCurrency !== 'USD') {

        url += `&base\_currency=${baseCurrency}`

      }

      if (currencies.length > 0) {

        url += `&currencies=${currencies.join(',')}`

      }

      const response = await fetch(url)

      if (!response.ok) {

        throw new Error(`API Error: ${response.status}`)

      }

      const data = await response.json()

      return data

    } catch (error) {

      console.error('Error fetching exchange rates:', error)

      throw error

    }

  },

  // Отримати список підтримуваних валют

  async getSupportedCurrencies() {

    try {

      const response = await fetch(`${BASE\_URL}/currencies?apikey=${API\_KEY}`)

      if (!response.ok) {

        throw new Error(`API Error: ${response.status}`)

      }

      const data = await response.json()

      return data

    } catch (error) {

      console.error('Error fetching currencies:', error)

      throw error

    }

  },

  // Конвертувати суму з однієї валюти в іншу

  convertAmount(amount, rate) {

    if (!amount || !rate) return 0

    return (parseFloat(amount) \* rate).toFixed(2)

  }

}

**3.3.6. Сервісний модуль «localStorage.js»**

Клас LocalStorageService реалізує повноцінну систему збереження та обробки історії розрахунків у localStorage. Його функціонал охоплює:

* Додавання нових записів до історії розрахунків для кожного з калькуляторів (loan, interest, deposit).
* Обмеження кількості записів до 10 (найновіші).
* Можливість видалити окремий запис за ID або повністю очистити історію.
* Експорт історії у JSON-файл (з іменем, типом і датою).
* Імпорт історії з файлу з підтримкою стратегій "merge" або "replace" та перевіркою на відповідність типу калькулятора.
* Глобальна очистка історії та перевірка доступності localStorage.

Модуль реалізований у вигляді сінглтона, що дозволяє централізовано керувати збереженням даних із будь-якого компонента програми. Завдяки інкапсульованій логіці та валідованим методам він гарантує стабільність та безпечність роботи з локальним сховищем.

Код до файлу «localStorage.js»

const STORAGE\_KEYS = {

  LOAN\_CALCULATOR\_HISTORY: 'loanCalculatorHistory',

  INTEREST\_CALCULATOR\_HISTORY: 'interestCalculatorHistory',

  DEPOSIT\_CALCULATOR\_HISTORY: 'depositCalculatorHistory',

}

class LocalStorageService {

  // Основні методи для роботи з localStorage

  setItem(key, value) {

    try {

      const serializedValue = JSON.stringify(value)

      localStorage.setItem(key, serializedValue)

      return true

    } catch (error) {

      console.error(`Error saving to localStorage with key "${key}":`, error)

      return false

    }

  }

  getItem(key) {

    try {

      const item = localStorage.getItem(key)

      return item ? JSON.parse(item) : null

    } catch (error) {

      console.error(`Error reading from localStorage with key "${key}":`, error)

      return null

    }

  }

  removeItem(key) {

    try {

      localStorage.removeItem(key)

      return true

    } catch (error) {

      console.error(`Error removing from localStorage with key "${key}":`, error)

      return false

    }

  }

  // Кредитний калькулятор

  getLoanCalculatorHistory() {

    return this.getItem(STORAGE\_KEYS.LOAN\_CALCULATOR\_HISTORY) || []

  }

  saveLoanCalculatorHistory(history) {

    return this.setItem(STORAGE\_KEYS.LOAN\_CALCULATOR\_HISTORY, history)

  }

  addLoanCalculation(calculation) {

    const currentHistory = this.getLoanCalculatorHistory()

    const newHistoryItem = {

      id: Date.now(),

      date: new Date().toLocaleDateString(),

      timestamp: Date.now(),

      ...calculation

    }

    const updatedHistory = [newHistoryItem, ...currentHistory.slice(0, 9)] // Осьтанні 10 записів

    this.saveLoanCalculatorHistory(updatedHistory)

    return updatedHistory

  }

  deleteLoanCalculationById(id) {

    const currentHistory = this.getLoanCalculatorHistory()

    const updatedHistory = currentHistory.filter(item => item.id !== id)

    this.saveLoanCalculatorHistory(updatedHistory)

    return updatedHistory

  }

  clearLoanCalculatorHistory() {

    return this.removeItem(STORAGE\_KEYS.LOAN\_CALCULATOR\_HISTORY)

  }

  // Відсотковий калькулятор

  getInterestCalculatorHistory() {

    return this.getItem(STORAGE\_KEYS.INTEREST\_CALCULATOR\_HISTORY) || []

  }

  saveInterestCalculatorHistory(history) {

    return this.setItem(STORAGE\_KEYS.INTEREST\_CALCULATOR\_HISTORY, history)

  }

  addInterestCalculation(calculation) {

    const currentHistory = this.getInterestCalculatorHistory()

    const newHistoryItem = {

      id: Date.now(),

      date: new Date().toLocaleDateString(),

      timestamp: Date.now(),

      ...calculation

    }

    const updatedHistory = [newHistoryItem, ...currentHistory.slice(0, 9)]

    this.saveInterestCalculatorHistory(updatedHistory)

    return updatedHistory

  }

  deleteInterestCalculationById(id) {

    const currentHistory = this.getInterestCalculatorHistory()

    const updatedHistory = currentHistory.filter(item => item.id !== id)

    this.saveInterestCalculatorHistory(updatedHistory)

    return updatedHistory

  }

  clearInterestCalculatorHistory() {

    return this.removeItem(STORAGE\_KEYS.INTEREST\_CALCULATOR\_HISTORY)

  }

  // Депозитний калькулятор

  getDepositCalculatorHistory() {

    return this.getItem(STORAGE\_KEYS.DEPOSIT\_CALCULATOR\_HISTORY) || []

  }

  saveDepositCalculatorHistory(history) {

    return this.setItem(STORAGE\_KEYS.DEPOSIT\_CALCULATOR\_HISTORY, history)

  }

  addDepositCalculation(calculation) {

    const currentHistory = this.getDepositCalculatorHistory()

    const newHistoryItem = {

      id: Date.now(),

      date: new Date().toLocaleDateString(),

      timestamp: Date.now(),

      ...calculation

    }

    const updatedHistory = [newHistoryItem, ...currentHistory.slice(0, 9)]

    this.saveDepositCalculatorHistory(updatedHistory)

    return updatedHistory

  }

  deleteDepositCalculationById(id) {

    const currentHistory = this.getDepositCalculatorHistory()

    const updatedHistory = currentHistory.filter(item => item.id !== id)

    this.saveDepositCalculatorHistory(updatedHistory)

    return updatedHistory

  }

  clearDepositCalculatorHistory() {

    return this.removeItem(STORAGE\_KEYS.DEPOSIT\_CALCULATOR\_HISTORY)

  }

  // Методи

  clearAllHistory() {

    this.clearLoanCalculatorHistory()

    this.clearInterestCalculatorHistory()

    this.clearDepositCalculatorHistory()

  }

  exportHistory(calculatorType) {

    let history = []

    let filename = ''

    switch (calculatorType) {

      case 'loan':

        history = this.getLoanCalculatorHistory()

        filename = 'loan-calculator-history'

        break

      case 'interest':

        history = this.getInterestCalculatorHistory()

        filename = 'interest-calculator-history'

        break

      case 'deposit':

        history = this.getDepositCalculatorHistory()

        filename = 'deposit-calculator-history'

        break

      default:

        console.error('Invalid calculator type for export')

        return

    }

    if (history.length === 0) {

      console.warn('No history to export')

      return

    }

    const exportData = {

      exportDate: new Date().toISOString(),

      calculatorType,

      history

    }

    const blob = new Blob([JSON.stringify(exportData, null, 2)], { type: 'application/json' })

    const url = URL.createObjectURL(blob)

    const a = document.createElement('a')

    a.href = url

    a.download = `${filename}-${Date.now()}.json`

    a.click()

    URL.revokeObjectURL(url)

  }

  // Імпорт історії з json файлу

  importHistory(calculatorType, importedData, mergeStrategy = 'merge') {

    try {

      // Валідація даних

      if (!importedData || !importedData.history || !Array.isArray(importedData.history)) {

        throw new Error('Invalid import file format')

      }

      if (importedData.calculatorType !== calculatorType) {

        throw new Error('Import file is for a different calculator type')

      }

      let currentHistory = []

      let saveMethod = null

      switch (calculatorType) {

        case 'loan':

          currentHistory = this.getLoanCalculatorHistory()

          saveMethod = this.saveLoanCalculatorHistory.bind(this)

          break

        case 'interest':

          currentHistory = this.getInterestCalculatorHistory()

          saveMethod = this.saveInterestCalculatorHistory.bind(this)

          break

        case 'deposit':

          currentHistory = this.getDepositCalculatorHistory()

          saveMethod = this.saveDepositCalculatorHistory.bind(this)

          break

        default:

          throw new Error('Invalid calculator type')

      }

      let mergedHistory = []

      if (mergeStrategy === 'replace') {

        // Заміна теперешньої історії на імпортовану

        mergedHistory = importedData.history

      } else {

        // Збереження імпортованої історії та відсіювання дублікацій

        const existingTimestamps = new Set(currentHistory.map(item => item.timestamp))

        const newItems = importedData.history.filter(item => !existingTimestamps.has(item.timestamp))

        mergedHistory = [...currentHistory, ...newItems].sort((a, b) => b.timestamp - a.timestamp)

        // Останні 20 записів

        mergedHistory = mergedHistory.slice(0, 20)

      }

      // Зберегти об'єднану історію

      saveMethod(mergedHistory)

      return {

        success: true,

        imported: importedData.history.length,

        total: mergedHistory.length,

        mergedHistory

      }

    } catch (error) {

      console.error('Error importing history:', error)

      return {

        success: false,

        error: error.message

      }

    }

  }

  isAvailable() {

    try {

      const test = '\_\_localStorage\_test\_\_'

      localStorage.setItem(test, test)

      localStorage.removeItem(test)

      return true

    } catch (error) {

      return false

    }

  }

}

const localStorageService = new LocalStorageService()

export default localStorageService

export { STORAGE\_KEYS }

# ВИСНОВКИ

У межах даної дипломної роботи було успішно розроблено функціональний веб-застосунок — «Калькулятор для фінансових обрахунків», який дозволяє здійснювати точні розрахунки депозитів, відсоткових ставок, кредитних платежів, а також конвертацію валют у зручному інтерфейсі. Проєкт реалізовано як SPA-застосунок із використанням сучасного JavaScript-стеку (React + Vite), що забезпечує високу швидкодію, адаптивність і простоту масштабування.

На етапі аналізу вимог (розділ 2) було чітко сформульовано як функціональні вимоги (реалізація окремих модулів для кожного типу фінансового розрахунку, інтерактивні графіки, підтримка кількох валют, історія розрахунків), так і нефункціональні (швидкодія, модульність, адаптивність до різних пристроїв, відокремлення логіки від інтерфейсу). Це дозволило ще на ранніх етапах задати точний напрямок технічної реалізації.

У процесі проєктування архітектури застосунку (розділи 2.2–2.5) було створено гнучку модульну структуру, в якій кожен калькулятор реалізовано як окремий React-компонент із власною логікою, стилем та інтеграцією з сервісними модулями. Такий підхід дозволив досягти високої масштабованості та спростити подальше розширення проєкту. Особливу увагу приділено користувацькому інтерфейсу — дизайн орієнтований на практичність, читабельність і мінімалізм, а інтерфейс побудований так, щоб користувач без спеціальної підготовки міг легко працювати з будь-яким модулем.

У розділі реалізації (розділ 3) детально описано структуру проєкту, включаючи компоненти, сторінки, логіку сервісів (наприклад, збереження в localStorage, робота з FreeCurrencyAPI) та особливості кожного калькулятора. Візуальна частина була реалізована з використанням CSS-модулів, а результати розрахунків виводяться у вигляді тексту й інтерактивних графіків, що значно підвищує зручність сприйняття даних. Кожен модуль також підтримує збереження історії обчислень із можливістю експорту та імпорту у форматі JSON.

Таким чином, поставлена мета була повністю досягнута — створено інтуїтивний, функціональний та гнучкий вебзастосунок, який виконує широкий спектр фінансових розрахунків та відповідає сучасним вимогам до зручності, надійності та візуальної подачі. Проєкт є прикладом застосування модульного підходу, відокремлення бізнес-логіки від представлення, а також ефективної інтеграції клієнтської логіки з зовнішніми API, що робить його придатним до подальшого розширення та підтримки.

# СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. React Documentation. (2025). Retrieved from https://reactjs.org/docs/getting-started.html
2. MDN Web Docs. (2025). JavaScript Guide. Retrieved from <https://developer.mozilla.org/en-US/docs/Web/JavaScript>
3. Vite Documentation. (2025). Retrieved from https://vitejs.dev/guide/
4. React Router Documentation. (2025). Retrieved from https://reactrouter.com/en/main/start/tutorial
5. Recharts Documentation. (n.d.). Retrieved from https://recharts.org/en-US
6. CSS Modules. (n.d.). Retrieved from <https://github.com/css-modules/css-modules>
7. FreeCurrencyAPI Documentation. (2025). Retrieved from https://freecurrencyapi.com/docs
8. W3Schools. (2025). HTML Form Validation. Retrieved from https://www.w3schools.com/html/html\_forms.asp
9. Crockford, D. (2008). *JavaScript: The Good Parts*. O'Reilly Media.
10. Banks, E. (2020). *React Design Patterns and Best Practices*. Packt Publishing.
11. Sanderson, M. (2021). *Modern Front-End Development for Beginners: Learn React, CSS, and JavaScript from Scratch*. Apress.
12. Krug, S. (2014). *Don't Make Me Think: A Common Sense Approach to Web Usability*. New Riders.
13. Nielsen Norman Group. (n.d.). UX Guidelines for Financial Interfaces. Retrieved from https://www.nngroup.com/articles/financial-ux/
14. Khan Academy. (n.d.). Finance and Capital Markets. Retrieved from <https://www.khanacademy.org/economics-finance-domain/core-finance>
15. Investopedia. (n.d.). Financial Calculator Formulas. Retrieved from <https://www.investopedia.com/>
16. React Hook Form. (n.d.). Retrieved from <https://react-hook-form.com/>
17. JSON Formatter & Validator. (n.d.). Retrieved from <https://jsonformatter.org/>
18. LocalStorage API. (n.d.). MDN Web Docs. Retrieved from <https://developer.mozilla.org/en-US/docs/Web/API/Window/localStorage>
19. Freecurrencyapi. (2025) from https://freecurrencyapi.com/docs/#official-libraries

# ДОДАТКИ

«src/main.jsx»

import React from 'react'

import ReactDOM from 'react-dom/client'

import App from './App.jsx'

import './index.css'

ReactDOM.createRoot(document.getElementById('root')).render(

  <React.StrictMode>

    <App />

  </React.StrictMode>,

)

«src/index.css»

:root {

  --primary-color: #3b82f6;

  --primary-hover: #2563eb;

  --secondary-color: #64748b;

  --background-color: #f8fafc;

  --card-background: #ffffff;

  --border-color: #e2e8f0;

  --text-primary: #1e293b;

  --text-secondary: #64748b;

  --success-color: #10b981;

  --warning-color: #f59e0b;

  --error-color: #ef4444;

  --shadow-sm: 0 1px 2px 0 rgb(0 0 0 / 0.05);

  --shadow-md: 0 4px 6px -1px rgb(0 0 0 / 0.1), 0 2px 4px -2px rgb(0 0 0 / 0.1);

  --border-radius: 8px;

}

input, select, button {

  font-family: inherit;

}

button {

  cursor: pointer;

}

button:disabled {

  cursor: not-allowed;

  opacity: 0.6;

}

«src/App.jsx»

import React from 'react'

import { BrowserRouter as Router, Routes, Route, Navigate } from 'react-router-dom'

import HomeLayout from './layout/HomeLayout'

import LoanCalculator from './pages/LoanCalculator'

import DepositCalculator from './pages/DepositCalculator'

import InterestCalculator from './pages/InterestCalculator'

import CurrencyConverter from './pages/CurrencyConverter'

import './App.css'

function App() {

  return (

    <Router>

      <Routes>

        <Route path="/" element={<HomeLayout />}>

          <Route index element={<Navigate to="/loan-calculator" replace />} />

          <Route path="loan-calculator" element={<LoanCalculator />} />

          <Route path="deposit-calculator" element={<DepositCalculator />} />

          <Route path="interest-calculator" element={<InterestCalculator />} />

          <Route path="currency-converter" element={<CurrencyConverter />} />

        </Route>

      </Routes>

    </Router>

  )

}

export default App

«src/App.css»

\* {

  margin: 0;

  padding: 0;

  box-sizing: border-box;

}

body {

  font-family: -apple-system, BlinkMacSystemFont, 'Segoe UI', 'Roboto', 'Oxygen',

    'Ubuntu', 'Cantarell', 'Fira Sans', 'Droid Sans', 'Helvetica Neue',

    sans-serif;

  -webkit-font-smoothing: antialiased;

  -moz-osx-font-smoothing: grayscale;

  background-color: #f8fafc;

  color: #334155;

}

#root {

  min-height: 100vh;

  display: flex;

  flex-direction: column;

}