Міністерство освіти і науки України
Харківський національний університет радіоелектроніки
Факультет центр післядипломної освіти
(повна назва)
Кафедра програмної інженерії
(повна назва)

КВАЛІФІКАЦІЙНА РОБОТА
Пояснювальна записка

рівень вищої освіти перший (бакалаврський)
Веб-додаток планування домашнього бюджету
(тема)

Виконав:
здобувач ___другого_____ року навчання
групи ПЗПІпз-23-1
Анастасія КУЗНЕЦОВА
(Власне ім'я, ПРІЗВИЩЕ)
Спеціальність 121 – Інженерія програмного забезпечення
(код і повна назва спеціальності)
Тип програми освітньо-професійна
Освітня програма Програмна інженерія
(повна назва освітньої програми)
Керівник ст.викл. кафедри ПІ Катерина ЗИБІНА
(посада, Власне ім'я, ПРІЗВИЩЕ)

Допускається до захисту
Зав. кафедри _____ Кирило СМЕЛЯКОВ
Підпис (Власне ім'я, ПРІЗВИЩЕ)

2025 р.

Харківський національний університет радіоелектроніки

Факультет ЦПО

Кафедра програмної інженерії

Рівень вищої освіти перший (бакалаврський)

Спеціальність 121 – Інженерія програмного забезпечення

Тип програми Освітньо-професійна

Освітня програма Програмна Інженерія

(шифр і назва)

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

ЗАВДАННЯ

НА КВАЛІФІКАЦІЙНУ РОБОТУ

здобувачеві Кузнецовій Анастасії Вячеславівні

(прізвище, ім'я, по батькові)

1. Тема роботи Веб-додаток планування домашнього бюджету

Затверджена наказом по університету від 19.05.2025р. № 86 Стз

2. Термін подання студентом роботи до екзаменаційної комісії 26.06.2022

3. Вихідні дані до роботи Розробити веб-застосунок для планування домашнього бюджету, який дозволятиме користувачеві створювати та керувати записами доходів і витрат. Реалізацію виконати з використанням мов програмування HTML, CSS, JavaScript та фреймворку React.

4. Перелік питань, що потрібно опрацювати в роботі

Вступ, аналіз предметної галузі, формування вимог до програмної системи, архітектура та проектування програмного забезпечення, опис прийнятих програмних рішень, тестування розробленого програмного забезпечення, висновки, додатки.

КАЛЕНДАРНИЙ ПЛАН

№	Назва етапів роботи	Термін виконання етапів роботи	Примітка
1	Аналіз предметної галузі	20.05.2025	*виконано*
2	Створення специфікації ПЗ	23.05.2025	*виконано*
3	Проектування ПЗ	01.06.2025	*виконано*
4	Розробка ПЗ	05.06.2025	*виконано*
5	Тестування ПЗ	07.06.2025	*виконано*
6	Оформлення пояснювальної записки	17.06.2025	*виконано*
7	Підготовка презентації та доповіді	22.06.2025	*виконано*
8	Попередній захист	25.06.2025	*виконано*
9	Нормоконтроль, рецензування	25.06.2025	*виконано*
10	Здача роботи у електронний архів	25.06.2025	*виконано*
11	Допуск до захисту у зав. кафедри	25.06.2025	*виконано*

Дата видачі завдання « 19 » « травня » 2025р.

Здобувач _____ Кузнецова В.А.

(підпис)

Керівник роботи _____ ст.викл. кафедри ПІ Катерина ЗИБІНА

(підпис) (посада, Власне ім’я, ПРІЗВИЩЕ)

РЕФЕРАТ

Звіт містить 90 сторінок основного тексту, 6 рисунків, 1 таблиця та 13 джерел за переліком посилань.

Ключові слова: БЮДЖЕТ, ВИТРАТИ, ВЕБ ЗАСТОСУНОК, ЗВІТИ, ІНТЕРФЕЙС, ОФЛАЙН, ПРОГРАМУВАННЯ, УПРАВЛІННЯ ГРОШИМА, ФІНАНСИ, UX.

У звіті описано процес планування та початку розробки веб-застосунку для ведення домашнього бюджету. Метою роботи є створення інструменту, що допомагає користувачам керувати особистими фінансами без підключення до Інтернету.

У процесі було застосовано методи аналізу потреб користувачів та початок розробки програмного забезпечення з використанням веб-фреймворків та методи тестування інтерфейсу.

Основою роботи є процес створення зручного, безпечного застосунку з можливістю обліку доходів та витрат, встановлення фінансових цілей і формування звітів. Програму планується використовувати серед широкого кола користувачів, зокрема для сімейного та особистого фінансового обліку.

Результати можуть бути застосовані у повсякденному житті, що забезпечує соціально-економічну ефективність. Рекомендовано продовжити розвиток продукту шляхом додавання нових функцій та вдосконалення UX/UI.

ABSTRACT

The report contains 90 pages of the main text, 6 pictures, 1 table, and 13 references.

Keywords: BUDGET, EXPENSES, FINANCE, INTERFACE, MONEY MANAGEMENT, ONLINE, PROGRAMMING, REPORTS, UX, WEB APPLICATION.

The report describes the process of planning and starting to develop a web application for managing a home budget. The aim of the work is to create a tool that helps users manage their personal finances without an Internet connection.

In the process, methods of analyzing user needs and starting software development using web frameworks and interface testing methods were applied.

The basis of the work is the process of creating a convenient, secure application with the ability to record income and expenses, set financial goals and generate reports. The program is planned to be used by a wide range of users, including family and personal financial accounting.

The results can be applied in everyday life, which ensures social and economic efficiency. It is recommended to continue developing the product by adding new features and improving UX/UI.

ЗМІСТ

СКОРОЧЕННЯ ТА УМОВНІ ПОЗНАЧКИ

У звіті використано такі скорочення та умовні позначення:

БД - база даних

GUI - графічний інтерфейс користувача

JSON - JavaScript Object Notation, формат зберігання структурованих даних

SQL - Structured Query Language, мова структурованих запитів до баз даних

UI - User Interface, інтерфейс користувача

UX - User Experience, користувацький досвід

ФФ - фінансова функціональність

MSVS - Microsoft Visual Studio, середовище розробки

API - Application Programming Interface, прикладний інтерфейс програмування

ВСТУП

Вірите в це чи ні, та гроші багато чого змінюють у житті. Фінанси не вирішать всі проблеми, і, звісно, гроші - це ще не все, але фінансові труднощі кидають тінь на ваше щастя. З грішми ви зможете більш вишукано вирішувати інші проблеми і матимете можливість познайомитися з іншими людьми, відвідати чарівні місця, присвятити себе цікавій роботі, бути впевненими в собі, отримувати більше визнання й використовувати інші шанси [1].

У нинішніх умовах світової нестабільності та зростаючої інфляції особливої актуальності набуває питання ефективного управління особистими фінансами. Планування домашнього бюджету є важливою складовою фінансової грамотності, що дозволяє забезпечити контроль над витратами, накопичення коштів і досягнення фінансових цілей. У цьому контексті розвиток інструментів для ефективного управління фінансами став необхідністю.

Провідні наукові установи й компанії, такі як Intuit (розробник Mint), You Need a Budget (YNAB), Microsoft та інші, вже створили низку інструментів і застосунків для особистого фінансового планування. Ці рішення дозволяють користувачам у реальному часі відстежувати доходи та витрати, формувати звіти, аналізувати фінансову активність і планувати бюджет. Світова тенденція спрямована на розробку зручних і персоналізованих програмних рішень, які мають за мету підвищення фінансової грамотності та ефективності управління домашнім бюджетом.

Фінансова грамотність має величезне значення не лише для окремих осіб, але й для суспільства в цілому. Ось декілька причин, чому фінансова грамотність є важливою:

- Запобігання фінансовим кризам: Особи, які розуміють основи управління грошима, можуть уникати ненадійних фінансових рішень, таких як надмірне запозичення або бездумні витрати.
- Підвищення якості життя: Люди з хорошими фінансовими навичками зазвичай мають більший контроль над своїм життям, можуть планувати свій бюджет, заощаджувати на важливі цілі і забезпечувати свої потреби.
- Сприяння розвитку бізнесу: Для підприємців фінансова грамотність є критично важливою. Вміння правильно планувати фінанси дозволяє знижувати ризики, підвищувати рентабельність і забезпечувати сталий розвиток бізнесу [2].

Актуальність цієї роботи полягає в тому, що сучасні цифрові інструменти для планування бюджету, хоча й доступні, не завжди підходять для потреб конкретного користувача або не відповідають вимогам простоти й доступності. Таким чином, створення персоналізованого веб-застосунку для планування домашнього бюджету є важливим кроком для полегшення управління фінансами на індивідуальному рівні.

Ціль роботи - створити веб-застосунок для планування домашнього бюджету на базі JavaScript, що дозволяє користувачам вести облік доходів і витрат, створювати фінансові категорії, генерувати звіти та візуалізувати дані. В процесі розробки будуть використані такі технології, як JavaScript (зокрема, бібліотека React для створення веб-додатків), JSON для зберігання даних, а також інші сучасні веб-технології для побудови зручного та ефективного інтерфейсу користувача.

Робота пов’язана з іншими навчальними проектами з програмування, UI/UX-дизайну та баз даних, а також із загальною підготовкою до професійної діяльності в галузі розробки програмного забезпечення.

1. АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

У даному розділі здійснюється огляд предметної області, до якої належить розроблений програмний продукт. Визначено основні функціональні та нефункціональні вимоги, проведено аналіз аналогічного програмного забезпечення та обґрунтовано доцільність створення нового рішення.

1.1 Загальна характеристика предметної області

Планування домашнього бюджету - це процес управління особистими фінансами з метою контролю доходів, витрат, накопичення коштів та досягнення фінансових цілей. Ця діяльність передбачає регулярне фіксування фінансових операцій, їх аналіз і прийняття обґрунтованих рішень на основі сформованих звітів.

В умовах цифровізації зростає попит на автоматизовані системи, що забезпечують зручне, швидке та безпечне ведення обліку фінансів. Такі системи мають включати механізми категоризації витрат, візуалізації даних, прогнозування та формування фінансових звітів.

1.2 Існуючі програмні рішення

На сучасному ринку існує низка популярних застосунків для ведення домашнього бюджету:

- Mint (Intuit) - онлайн-платформа з широким функціоналом, автоматичним підключенням до банківських рахунків.
- You Need a Budget (YNAB) - програма, орієнтована на навчання користувачів методиці розподілу бюджету.
- HomeBank - десктопний застосунок з базовим функціоналом обліку витрат і доходів.
- Money Manager EX - кросплатформенна програма з відкритим вихідним кодом.

Недоліки деяких існуючих рішень:

- складність у користуванні (особливо для новачків);
- відсутність української мови;
- обмеження безкоштовних версій;
- прив’язаність до інтернет-з’єднання;
- недостатня підтримка локалізації;
- складна установка на різних ОС.

Ці обмеження створюють потребу в простих додатках, які будуть адаптовані до українських користувачів.

1.3 Обґрунтування розробки власного застосунку

Створення власного веб-застосунку дозволяє врахувати потреби цільової аудиторії, забезпечити інтуїтивно зрозумілий інтерфейс, простоту та швидкість використання та можливість роботи в автономному режимі.

Основні переваги запропонованого рішення:

- гнучкість налаштувань;
- простий та зручний інтерфейс;
- можливість локалізації українською мовою;
- відкритість до майбутньої модифікації й розширення.

Завдяки використанню JavaScript та фреймворку React, застосунок буде доступний через мережу Інтернет.

1.4 Вимоги до програмного забезпечення

Вимоги до програмного забезпечення - набір вимог щодо властивостей програми, якості та функцій програмного забезпечення, що буде розроблено, або знаходиться у розробці. Вимоги визначаються на самому початку розробки, в процесі аналізу вимог та обов'язково фіксуються в специфікації вимог, діаграмах прецедентів та інших артефактах процесу аналізу та розробки вимог.

1.4.1 Функціональні вимоги та нефункціональні вимоги

Функціональні вимоги - це вимоги до програмного забезпечення, які описують внутрішню роботу системи та її поведінку, наприклад:

- додавання, редагування та видалення фінансових записів (доходів і витрат);
- категоризація транзакцій;
- формування звітів за вибраний період;
- візуалізація даних у вигляді графіків;
- збереження даних у форматі JSON або в локальній базі даних.

Нефункціональні вимоги - це вимоги до програмного забезпечення, які описують характеристики та обмеження роботи системи, наприклад:

- кросплатформеність;
- локалізація інтерфейсу;
- інтуїтивний UI/UX;
- висока швидкодія та стабільність;
- мінімальні системні вимоги.

Вимоги до програмного забезпечення є ключовим етапом у процесі розробки системи, оскільки вони визначають її функціональність, поведінку та очікувану якість.

2. ОПИС ПРЕДМЕТУ РОЗРОБКИ

Предмет розробки - це веб-застосунок для планування домашнього бюджету, який дозволить користувачам вести свої доходи та витрати, створювати категорії фінансів, аналізувати фінансові потоки та генерувати звіти.

Застосунок передбачає кілька ключових можливостей:

1. Ведення обліку доходів і витрат: Користувач може вводити дані про свої фінансові потоки, категоризувати їх за різними видами витрат та доходів.

   Необхідно подумати та проаналізувати, які статті витрат можна скоротити чи прибрати, а без яких неможливо обійтись.
2. Створення фінансових категорій: Застосунок дозволяє користувачу відстежити та прописати основні статті витрат (наприклад, "Продукти", "Оренда", "Транспорт") [3].
3. Генерація звітів: Застосунок автоматично формує звіти, що дозволяють аналізувати фінансову діяльність за різними періодами.
4. Візуалізація даних: Для зручності користувача застосунок генерує графіки та діаграми, що допомагають оцінити співвідношення витрат та доходів.

Застосунок також пропонує можливість сімейної підписки, що полегшує процес прийняття фінансових рішень.

2.1 Опис теоретичних основ і принципів дії

Веб-додаток - це прикладне програмне забезпечення, яке працює у веб браузері та для своєї функціональності використовує інтернет-з’єднання. Веб-додатки не потребують встановлення на локальний комп’ютер - доступ до них здійснюється через мережу за допомогою URL-адреси.

Основною особливістю сучасних веб-додатків є їх здатність функціонувати як у підключеному до мережі режимі (online), так і частково

в автономному (offline), завдяки використанню технологій локального зберігання даних, таких як LocalStorage, IndexedDB або Service Workers. Це дозволяє забезпечити стабільну роботу додатка навіть при нестабільному або відсутньому підключенні до Інтернету.

На відміну від десктопних програм, веб-додатки не потребують ручного оновлення - всі оновлення виконуються централізовано на сервері, що дозволяє забезпечити актуальність функціоналу для всіх користувачів одразу. Крім того, веб-додатки легко масштабуються, дають змогу підтримувати багато користувачів одночасно та можуть бути доступними з різних пристроїв, зокрема комп’ютерів, планшетів і смартфонів.

Такі додатки часто реалізуються з використанням стеків технологій на основі JavaScript, HTML5, CSS, а для серверної частини - Node.js, Express, баз даних (наприклад, PouchDB, PostgreSQL), що дозволяє досягати високої інтерактивності, швидкодії та зручності користування.

Розробка застосунку базується на використанні сучасних програмних інструментів, які дозволяють реалізувати функціональність, зручну і зрозумілу для користувачів[4].

2.1.1 Програмні інструменти

Програмні інструменти - це спеціалізоване програмне забезпечення, яке використовують для підтримки процесів розробки, тестування, аналізу та супроводу інших програмних систем.

При розробці веб-додатку для планування домашнього бюджету було застосовано:

- Мова програмування JavaScript: Сучасний JavaScript – це “безпечна” мова програмування. Вона не надає низькорівневого доступу до пам’яті чи процесора, оскільки була створена для браузерів, які цього не потребують.

Можливості JavaScript значно залежать від середовища, у якому виконується скрипт. Наприклад, Node.js підтримує функції, які дозволяють JavaScript читати/записувати довільні файли, здійснювати мережеві запити тощо [5].

Це основна мова для розробки логіки застосунку. JavaScript дозволяє працювати з даними в реальному часі, обробляти введення користувача та забезпечувати інтерактивність інтерфейсу.

- Бібліотека React - це JavaScript-бібліотека з відкритим вихідним кодом, розроблена компанією Meta (раніше Facebook), яка призначена для побудови інтерфейсів користувача. Вона дає змогу створювати компонентну архітектуру веб-додатків, що спрощує розробку, підтримку та масштабування складних інтерфейсів.

  React працює на основі концепції віртуального DOM (Virtual DOM), що забезпечує ефективне оновлення лише тих частин інтерфейсу, які змінилися, завдяки чому досягається висока продуктивність навіть у складних односторінкових додатках (SPA - Single Page Application).

  Застосування React дозволяє реалізовувати адаптивні, інтерактивні та динамічні веб інтерфейси, які можуть працювати на всіх сучасних браузерах та пристроях. Завдяки широкій екосистемі додаткових бібліотек (React Router, Redux, Axios тощо), React легко інтегрується з бекендом, API, системами автентифікації, базами даних тощо.

  React є кросплатформним інструментом - код, написаний з його використанням, може застосовуватись у різних середовищах: веб (через ReactDOM), мобільних додатках (через React Native), а також у десктопних додатках у поєднанні з Electron.
- JSON (JavaScript Object Notation): Формат зберігання та обміну даними. У JSON зберігаються всі фінансові операції користувача

(доходи, витрати, категорії), що забезпечує простоту роботи з даними та їх передачу між компонентами застосунку.

Завдяки використанню JavaScript та фреймворку React, застосунок буде доступний через мережу Інтернет.

2.1.2 Принципи дії застосунку

Принципи дії застосунку визначають загальну логіку роботи, взаємодію програми з користувачем та обробку даних, які забезпечують виконання його функціонального призначення.

- Обробка даних в реальному часі: Застосунок автоматично оновлює введені дані, забезпечуючи актуальність інформації для користувача.
- Персоналізація категорій: Користувач може створювати власні категорії витрат та доходів, що дозволяє адаптувати застосунок під свої потреби.
- Генерація звітів: Застосунок автоматично формує звіти по введеним даним. Користувач може обирати різні періоди для аналізу та отримувати докладні графіки.
- Збереження даних у форматі JSON: Це забезпечує зручність і стабільність роботи з даними, а також можливість передачі та збереження інформації між різними компонентами застосунку.

Ці принципи забезпечують ефективну, зручну та безпечну роботу застосунку.

2.1.3 Основні рішення і методи роботи

Головною задачею роботи було створити простий додаток, який буде зручний для українських користувачів.

Основні рішення та методи роботи включають:

- Метод створення інтерфейсу користувача: Для побудови графічного інтерфейсу використовується бібліотека React. Вона дозволяє створювати сучасні, зручні та інтуїтивно зрозумілі інтерфейси за

допомогою веб-технологій (HTML, CSS, JavaScript). Це дає змогу створити єдиний інтерфейс для різних операційних систем і забезпечити високу інтерактивність застосунку.

- Метод збереження і обробки даних: Дані користувача зберігаються у форматі JSON, що є стандартом для зберігання структурованих даних. Кожен запис про дохід чи витрату містить дату, суму, категорію та інші параметри. Це дозволяє користувачам легко переглядати та редагувати свої фінансові дані, а також формувати звіти на основі введених даних.
- Метод візуалізації фінансової інформації: Для відображення фінансових даних використовуються графіки та діаграми, створені за допомогою бібліотеки Chart.js. Це дозволяє користувачам зручно аналізувати співвідношення витрат і доходів.
- Метод генерації звітів: Застосунок генерує різноманітні звіти, які допомагають користувачеві побачити, на що витрачаються кошти і в яких категоріях є можливість заощадити. Звіти можуть бути побудовані за різні періоди (наприклад, щомісяця, щокварталу тощо).

Тож, ці методи та рішення допомогли у створенні власного веб-застосунку, дозволили врахувати потреби цільової аудиторії, забезпечити інтуїтивно зрозумілий інтерфейс, простоту та швидкість використання та можливість роботи в автономному режимі.

3. АРХІТЕКТУРА ТА ПРОЕКТУВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

3.1 Проектування структури зберігання даних

Також, у процесі роботи було створено модель сутностей та міграції для додатку домашнього бюджету.

Модель сутностей загалом описує, які дані зберігаються в системі та як ці дані пов’язані між собою, а також, які атрибути має кожна сутність (наприклад, користувач, транзакція, категорія) - це основа логіки застосунку.

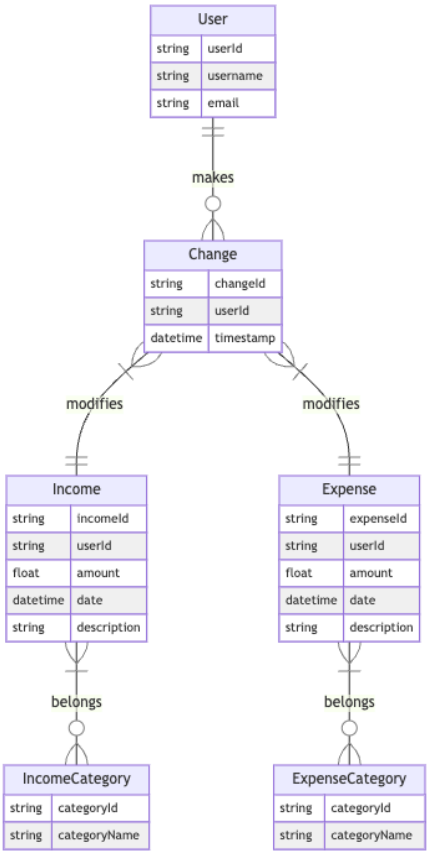


Рисунок 3.1 - Модель сутностей та міграції для додатку Домашнього бюджету (рисунок виконаний самостійно)

За допомогою міграцій ви можете перевести існуючу базу даних в інший стан і навпаки: ці переходи зберігаються у файлах міграції, які описують,

як перейти в новий стан і як відмінити зміни, щоб повернутися до старого стану [7].

3.2 Проектування архітектури ПЗ

3.2.1 Загальна структура

Архітектура програмного забезпечення - це структура програми або обчислювальної системи, яка включає програмні компоненти, видимі зовні властивості цих компонентів, а також відносини між ними [10].

Застосунок складається з трьох основних рівнів:

- Frontend - було реалізовано з використанням React, відповідає за інтерфейс користувача та обробку подій. Бібліотека Chart.js використовується для побудови графіків і діаграм.
- Backend - реалізований на Node.js з фреймворком Express.js, обробляє логіку застосунку та взаємодіє з базою даних.
- База даних - PouchDB, що дозволяє зберігати всю інформацію про користувачів, транзакції, бюджети, налаштування тощо.

Документування архітектури ПЗ спрощує процес комунікації між зацікавленими особами, дозволяє зафіксувати прийняті на ранніх етапах проектування рішення про високорівневий дизайн системи і дозволяє використовувати компоненти цього дизайну і шаблони повторно в інших проектах [10].

3.2.2 Основні компоненти системи

Проектована архітектура забезпечує швидку роботу веб додатку, зручність використання для кінцевого користувача та технічну готовність до подальшого розширення функціоналу проєкту.

Клієнтська сторона (React):

- форми додавання/редагування доходів та витрат;
- сторінки перегляду статистики (графіки, діаграми);
- особистий кабінет користувача;

- компоненти навігації та локалізації інтерфейсу (українська мова).

Серверна частина (Express):

- модулі обробки запитів (контролери);
- маршрути для доступу до API (/api/users, /api/transactions, тощо);
- модулі авторизації/реєстрації;
- логіка валідації та обробки даних.

Цей підхід створює стійкий фундамент для подальшого розвитку програмного продукту та покращення функціоналу загалом, при цьому не втрачаючи орієнтації на потреби користувача та потреби ринку.

3.2.3 Взаємодія компонентів

Комунікація між фронтендом та бекендом здійснюється через RESTful API у форматі JSON. Клієнт надсилає HTTP-запити (GET, POST, PUT, DELETE), на які сервер відповідає відповідними даними або повідомленням про успіх/помилку.

```
// Реалізація виклику API
const api: any = {
 // для роботи з інформацією користувача
 user: {
   get: async () => {
         const response = await fetch(HOST + '/api/user?id=' +
CURRENT_USER_ID);
     return response.json();
   },
   update: (data: { userId: string, username: string, email: string
}) => fetch(HOST + '/api/user', {
     method: 'PUT',
     headers: {
       'Content-Type': 'application/json'
     },
     body: JSON.stringify({ ...data, id: CURRENT_USER_ID })
   })
 }, …
```

Фронтенд, як наочна частина застосунку, відповідає за вивід інформації користувачеві та збір його вхідних даних. Коли людина взаємодіє з інтерфейсом (наприклад, натискає кнопку, заповнює форму), фронтенд надсилає запит на бекенд. Цей запит містить необхідні дані для виконання певної операції.

Бекенд, отримавши запит, обробляє його згідно зі своєю логікою: звертається до бази даних для отримання або оновлення інформації, виконує необхідні обчислення або взаємодіє з іншими сервісами через API.

Після обробки бекенд надсилає відповідь назад на фронтенд. Ця відповідь може містити інформацію, повідомлення про успішне виконання операції або інформацію про помилку, яку фронтенд відображає користувачеві.

Таким чином, фронтенд і бекенд працюють як одне ціле, забезпечуючи повноцінну функціональність сайту чи застосунку [11].

3.2.4 5. Переваги архітектурного рішення

Кросплатформеність: фронтенд працює у будь-якому браузері, бекенд - на будь-якому сервері з Node.js.

Розширюваність: можливість легко додати нові функції (наприклад, підключення банківських API).

3.3 Дизайн системи

Застосунок передбачає декілька ключових можливостей, таких як ведення обліку доходів і витрат, створення фінансових категорій та візуалізація даних.

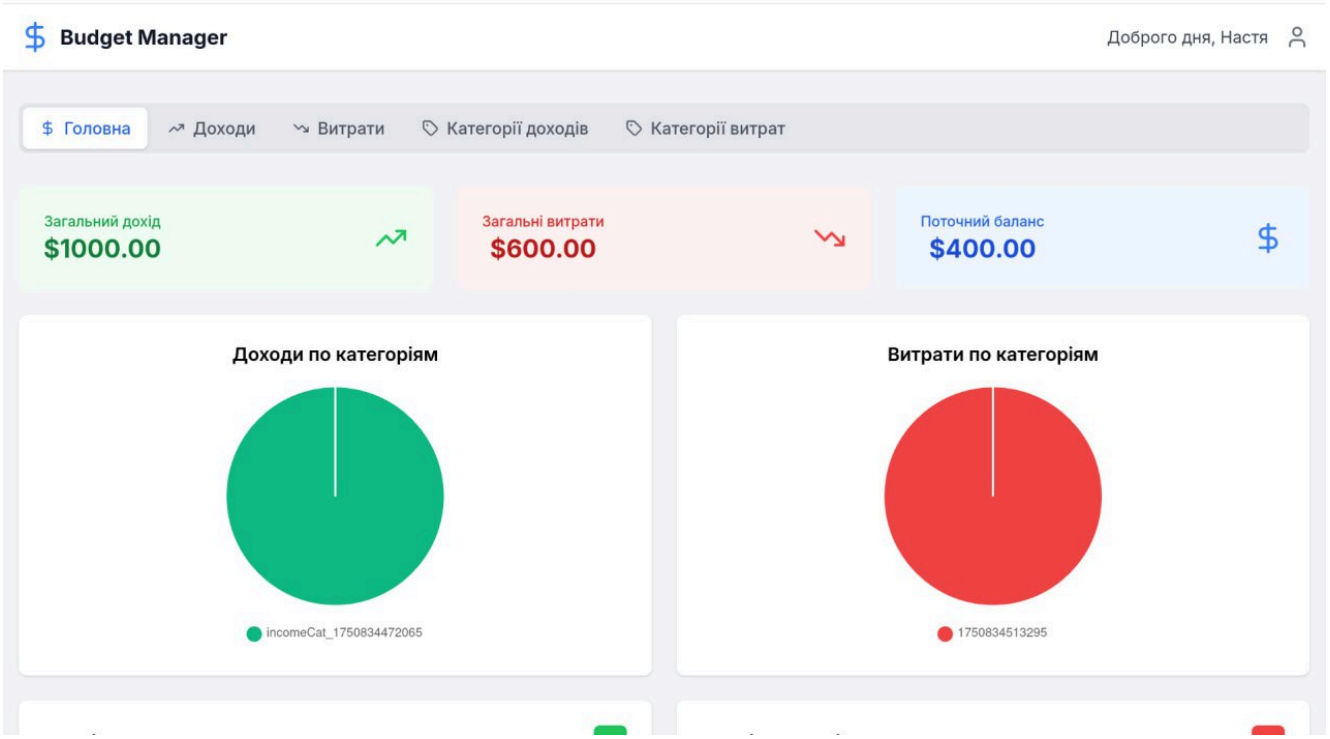


Рисунок 3.2 - Головний екран Веб-додатку для планування домашнього бюджету (рисунок виконаний самостійно)

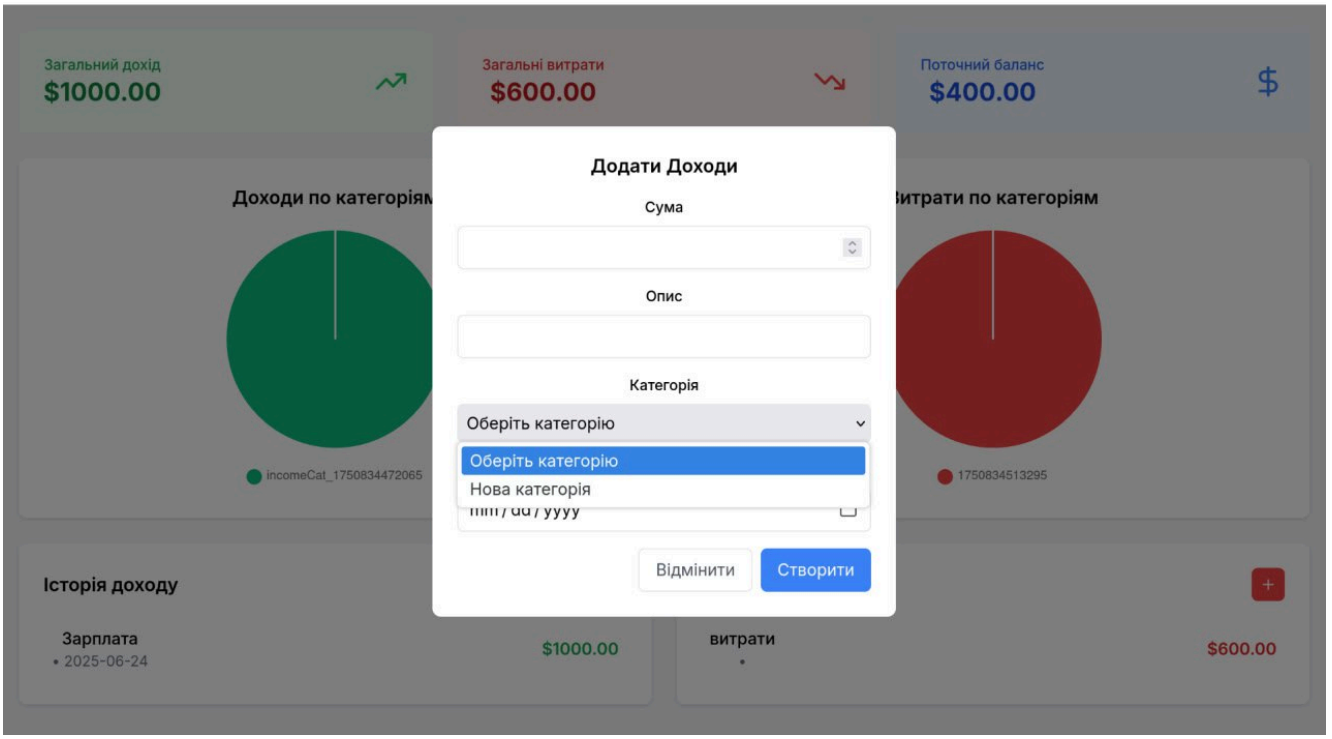


Рисунок 3.3 - Діалог додавання доходів (рисунок виконаний самостійно)

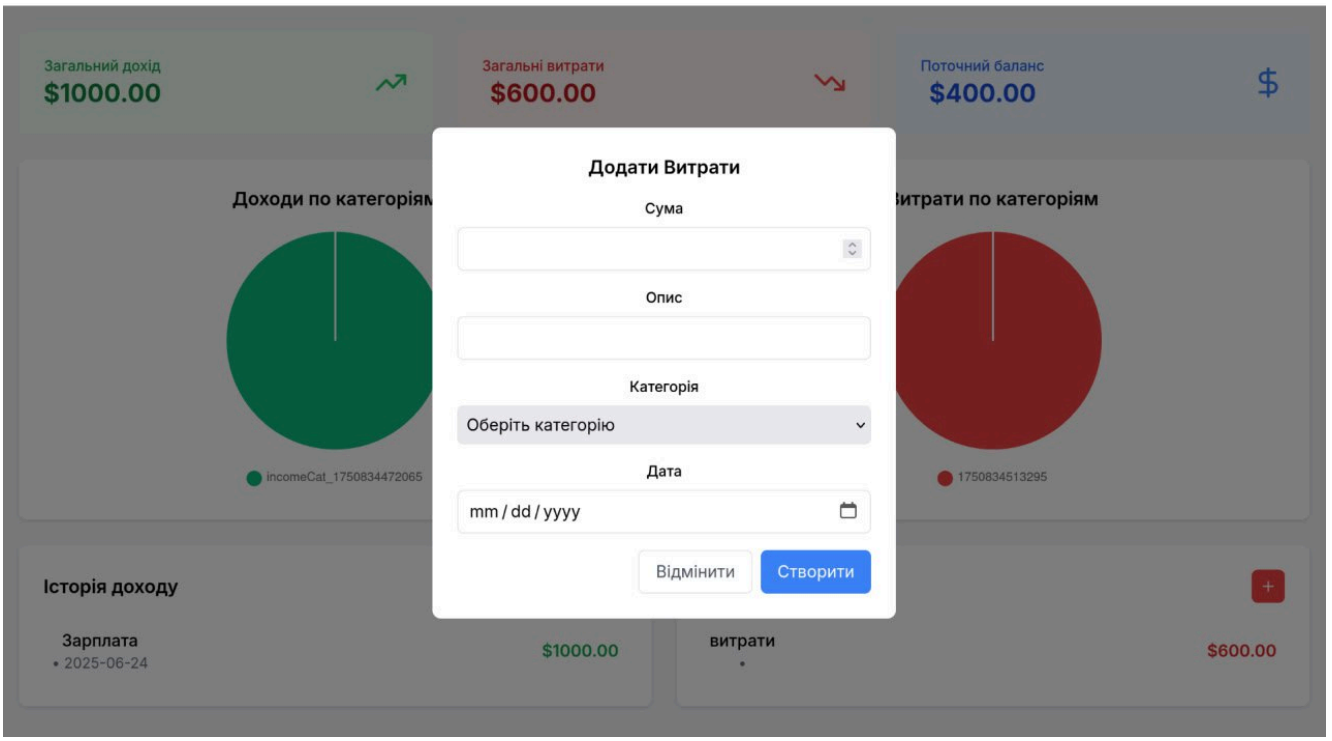


Рисунок 3.4 - Діалог додавання витрат (рисунок виконаний самостійно)

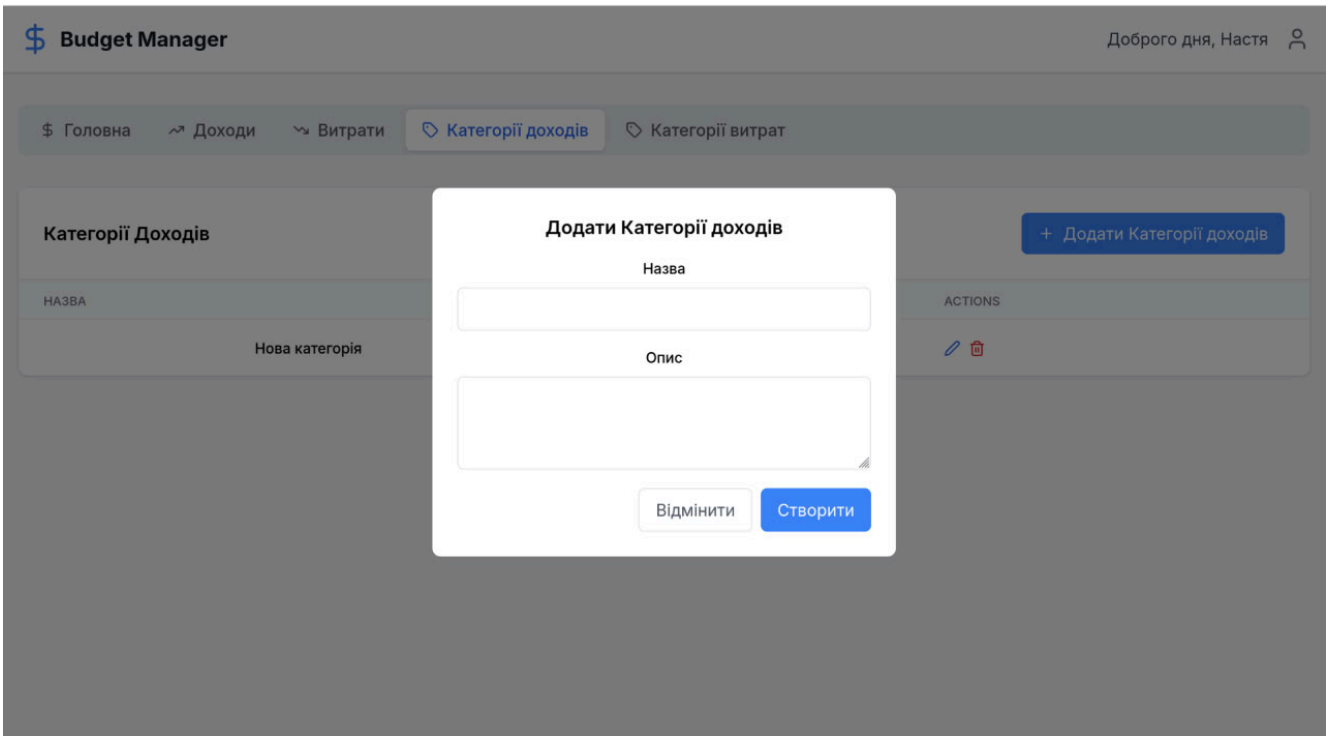


Рисунок 3.5 - Створення нової категорії доходів (рисунок виконаний самостійно)

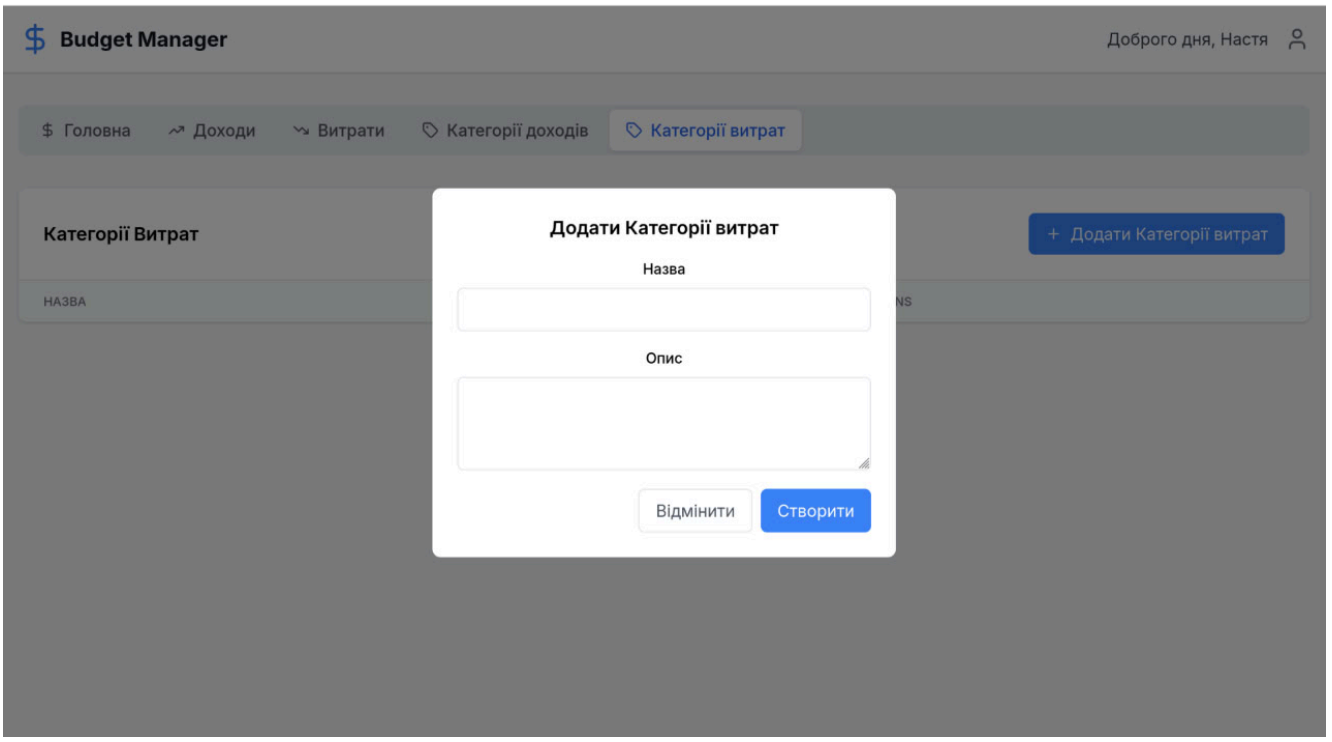


Рисунок 3.6 - Створення нової категорії витрат (рисунок виконаний самостійно)

Створення цього веб-застосунку дозволяє врахувати потреби цільової аудиторії, забезпечити інтуїтивно зрозумілий інтерфейс, простоту та швидкість використання та можливість роботи в автономному режимі.

4. ОПИС ПРИЙНЯТИХ ПРОГРАМНИХ РІШЕНЬ

Під час розробки застосунку для планування домашнього бюджету було прийнято рішення реалізувати проєкт як веб застосунок із чітким поділом на фронтенд і бекенд, що забезпечує масштабованість, зручність підтримки та гнучкість у подальшому розвитку.

4.1 Фронтенд (інтерфейс користувача)

Інтерфейс користувача реалізовано з використанням React.

React - це декларативна, ефективна і гнучка JavaScript-бібліотека, призначена для створення інтерфейсів користувача. Вона дозволяє компонувати складні інтерфейси з невеликих окремих частин коду - “компонентів” [12].

Основні функції фронтенду:

- введення доходів і витрат;
- перегляд статистики та фінансових звітів;
- налаштування бюджету;
- взаємодія з REST API бекенду.

```
// Реалізація виклику API
const api: any = {
 // для роботи з інформацією коритувача
 user: {
   get: async () => {
            const response = await fetch(HOST + '/api/user?id=' +
CURRENT_USER_ID);
     return response.json();
   },
   update: (data: { userId: string, username: string, email: string }) =>
fetch(HOST + '/api/user', {
     method: 'PUT',
     headers: {
       'Content-Type': 'application/json'
     },
     body: JSON.stringify({ ...data, id: CURRENT_USER_ID })
   })
 },
 // для роботи з інформацією про доходи
 income: {
   get: async () => {
         const response = await fetch(HOST + '/api/income?userId=' +
CURRENT_USER_ID);
     return response.json();
   },
   post: (data: any) => fetch(HOST + '/api/income', {
```

```
      method: 'POST',
      headers: {
        'Content-Type': 'application/json'
      },
          body: JSON.stringify({ ...data, incomeId: Date.now(), userId:
CURRENT_USER_ID })
    }),
    update: (data: any) => fetch(HOST + '/api/income', {
      method: 'PUT',
      headers: {
        'Content-Type': 'application/json'
      },
      body: JSON.stringify({ ...data, userId: CURRENT_USER_ID })
    }),
    delete: (id: string) => fetch(HOST + '/api/income?id=' + id, {
      method: 'DELETE',
      headers: {
        'Content-Type': 'application/json'
      }
    })
  },
  // для роботи з інформацією про витрати
  expense: {
    get: async () => {
          const response = await fetch(HOST + '/api/expense?userId=' +
CURRENT_USER_ID);
      return response.json();
    },
    post: (data: any) => fetch(HOST + '/api/expense', {
      method: 'POST',
      headers: {
        'Content-Type': 'application/json'
      },
          body: JSON.stringify({ ...data, expenseId: Date.now(), userId:
CURRENT_USER_ID })
    }),
    update: (data: any) => fetch(HOST + '/api/expense', {
      method: 'PUT',
      headers: {
        'Content-Type': 'application/json'
      },
      body: JSON.stringify({ ...data, userId: CURRENT_USER_ID })
    }),
    delete: (id: string) => fetch(HOST + '/api/expense?id=' + id, {
      method: 'DELETE',
      headers: {
        'Content-Type': 'application/json'
      }
    })
  },
  // для роботи з інформацією про категорії доходів
  incomeCategory: {
    get: async () => {
       const response = await fetch(HOST + '/api/incomeCategory?userId=' +
CURRENT_USER_ID);
```

```
    return response.json();
  },
  post: (data: any) => fetch(HOST + '/api/incomeCategory', {
    method: 'POST',
    headers: {
      'Content-Type': 'application/json'
    },
      body: JSON.stringify({ ...data, categoryId: Date.now(), userId:
CURRENT_USER_ID })
  }),
  update: (data: any) => fetch(HOST + '/api/incomeCategory', {
    method: 'PUT',
    headers: {
      'Content-Type': 'application/json'
    },
    body: JSON.stringify({ ...data, userId: CURRENT_USER_ID })
  }),
  delete: (id: string) => fetch(HOST + '/api/incomeCategory?id=' + id, {
    method: 'DELETE',
    headers: {
      'Content-Type': 'application/json'
    }
  }
  )
 },
 // для роботи з інформацією про категорії витрат
 expenseCategory: {
  get: async () => {
     const response = await fetch(HOST + '/api/expenseCategory?userId=' +
CURRENT_USER_ID);
    return response.json();
  },
  post: (data: any) => fetch(HOST + '/api/expenseCategory', {
    method: 'POST',
    headers: {
      'Content-Type': 'application/json'
    },
      body: JSON.stringify({ ...data, categoryId: Date.now(), userId:
CURRENT_USER_ID })
  }),
  update: (data: any) => fetch(HOST + '/api/expenseCategory', {
    method: 'PUT',
    headers: {
      'Content-Type': 'application/json'
    },
    body: JSON.stringify({ ...data, userId: CURRENT_USER_ID })
  }),
   delete: (id: string) => fetch(HOST + '/api/expenseCategory?id=' + id,
{
    method: 'DELETE',
    headers: {
      'Content-Type': 'application/json'
    }
  }
  )
```

```
  }
};
```

Для візуалізації фінансових даних, графіків та діаграм використовується бібліотека Chart.js, яка дозволяє створювати інтерактивні та інформативні візуальні елементи.

4.2 Бекенд (серверна логіка)

Фактично бекенд – це ядро, або “мозок” будь-якого застосунку. Він відповідає за зберігання, обробку та організацію даних, реалізацію бізнес-логіки та забезпечення безпеки. Серверна частина слугує посередником між користувацьким інтерфейсом (фронтендом) та базами даних, API та іншими сервісами, гарантуючи стабільну та надійну роботу застосунку [11].

```
  const express = require('express');
  const bodyParser = require('body-parser');
  const cors = require('cors')
  const budgetController = require('./controllers/budgetController');
  const app = express();
  const port = 3030;
  // Налаштовуємо Middleware
  app.use(bodyParser.urlencoded({ extended: true }));
  app.use(bodyParser.json());
  app.use(cors());
  // Rest Api
  // API для користувача
  app.post('/api/user', budgetController.addUser);
  app.put('/api/user', budgetController.updateUser);
  app.get('/api/user', budgetController.readUser);
  app.delete('/api/user', budgetController.deleteUser);
  // API для роботи з доходами
  app.post('/api/income', budgetController.addIncome);
  app.put('/api/income', budgetController.updateIncome);
  app.get('/api/income', budgetController.readIncome);
  app.delete('/api/income', budgetController.deleteIncome);
  //API для роботи з витратами
  app.post('/api/expense', budgetController.addExpense);
  app.put('/api/expense', budgetController.updateExpense);
  app.get('/api/expense', budgetController.readExpense);
  app.delete('/api/expense', budgetController.deleteExpense);
  //API для роботи з категоріями доходів
  app.post('/api/incomeCategory', budgetController.addIncomeCategory);
  app.put('/api/incomeCategory', budgetController.updateIncomeCategory);
  app.get('/api/incomeCategory', budgetController.readIncomeCategory);
  app.delete('/api/incomeCategory',
budgetController.deleteIncomeCategory);
  //API для роботи з карегоріями витрат
  app.post('/api/expenseCategory', budgetController.addExpenseCategory);
```

```
  app.put('/api/expenseCategory',
budgetController.updateExpenseCategory);
  app.get('/api/expenseCategory', budgetController.readExpenseCategory);
  app.delete('/api/expenseCategory',
budgetController.deleteExpenseCategory);
  // Start server
  app.listen(port, () => {
     console.log(`Server is running on http://localhost:${port}`);
  });
```

Серверну частину було реалізовано на основі Node.js з використанням фреймворку Express.js. Цей стек дозволяє ефективно обробляти HTTP-запити, реалізовувати логіку застосунку та здійснювати підключення до бази даних [13].

4.3 REST API

Взаємодія між клієнтською та серверною частинами відбувається через RESTful API. JSON - формат обміну даними, що забезпечує простоту, легкість обробки та кросплатформеність.

4.4 База даних

Для зберігання всіх даних було використано PouchDB - документоорієнтована NoSQL база даних, що зберігає інформацію у форматі BSON (аналог JSON). Це дозволяє легко працювати з гнучкою структурою фінансових записів, користувачів та категорій [9].

```
const PouchDB = require('pouchdb');
//додавання плагіну для пошуку в базі
PouchDB.plugin(require('pouchdb-find'));
//створення локальної бази даних
const db = new PouchDB('userDB');
//створення індексу для бази даних
db.createIndex({
   index: { fields: ['userId', 'type'] }
});
//реалізація CRUD операцій для роботи з інформацією користувача
const addUser = function (userId, username, email) {
   return new Promise((resolve, reject) => {
       db.put({
           _id: `user_${userId}`,
           type: "user",
           username: username,
           email: email
       })
           .then(() => {
               resolve();
           })
           .catch(function (err) {
               reject(err);
```

```
            });
    });
}
const readUser = function (userId) {
    return new Promise((resolve, reject) => {
        db.get(`user_${userId}`).then(function (user) {
            resolve(user);
        }).catch(function (err) {
            reject(`User ${userId} doesn't exists.`);
        });
    });
}
const deleteUser = function (userId) {
     return readUser(userId).then(user => new Promise((resolve, reject) =>
{
        const related = db.find({
            selector: {
                userId: user._id
            }
        }).then((data) => {
            if (data.docs.length > 0) {
                      reject(`Can't remove user ${userId}, some data is
associated with the user: \n` + JSON.stringify(data.docs, null, 2));
            } else {
                db.remove(user._id, user._rev).then(function () {
                    resolve();
                }).catch(function () {
                    reject(`Cant remove user ${userId}.`);
                });
            }
        })
    }));
}
const updateUser = function (userId, username, email) {
    return new Promise((resolve, reject) => {
        db.get(`user_${userId}`).then(function (user) {
            return db.put({
                _id: `user_${userId}`,
                _rev: user._rev,
                type: "user",
                username: username,
                email: email
            });
        }).then(function () {
            resolve();
        }).catch(function (err) {
            reject(err);
        });
    });
}
```

База даних - PouchDB, що дозволяє зберігати всю інформацію про користувачів, транзакції, бюджети, налаштування тощо.

5. ТЕСТУВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

Тестування ПЗ - це процес перевірки програмного продукту з метою виявлення дефектів, помилок та недоліків перед його випуском на ринок або в експлуатацію. Цей процес охоплює запуск програми з різними вхідними даними та умовами, а також аналіз реакції програми на ці дані.

Мета тестування - це підтвердження правильності роботи програми відповідно до вимог до неї, а також забезпечення високої якості та надійності програмного продукту [8].

Щоб перевірити валідність та якість розробки веб-застосунку з планування домашнього бюджету використовувалось ручне тестування. Мануальне тестування охоплює як функціональні, так і нефункціональні аспекти, оскільки при розробці додатку було важливо враховувати обидва типи тестування для розробки якісного продукту.

Переваги ручного тестування:

- Гнучкість та можливість адаптації до змін;
- Ефективне для випадків тестування, які складно автоматизувати;
- Легке виявлення нових дефектів, які не були враховані в автоматизованих скриптах.

Тестування - це важлива стадія у створенні програмного продукту, яка спрямована на контроль якості та виявлення можливих збоїв. В процесі тестування програми використовуються різноманітні види тестів, які допомагають перевірити її працездатність та відповідність вимогам. Від модульного тестування окремих компонентів до інтеграційного, системного - кожен вид тестування має свої особливості та важливість у забезпеченні якості продукту [8].

Таблиця 5.1 – Тест-кейс №1 (таблиця виконана самостійно)

№	Назва тест-кейсу	Кроки	Очікуваний результат
1.1	Перегляд профілю користувача	1. Перейти на сторінку профілю користувача.	Відображається актуальна інформація про користувача (ім'я, email тощо).
1.2	Редагування профілю користувача	1. Перейти на сторінку профілю. 2. Натиснути кнопку "Редагувати". 3. Внести зміни в поля (наприклад, ім'я). 4. Зберегти зміни.	Зміни успішно зберігаються та відображаються у профілі. Відображається повідомлення про успішне оновлення.
1.3	Редагування профілю з невалідними даними	1. Перейти на сторінку профілю. 2. Натиснути кнопку "Редагувати". 3. Ввести невалідні дані (наприклад, email без "@", порожнє ім'я). 4. Зберегти зміни.	З'являються повідомлення про помилки валідації для відповідних полів. Зміни не зберігаються.
1.4	Видалення користувача (з підтвердженням)	1. Перейти на сторінку профілю. 2. Натиснути кнопку "Видалити користувача". 3. У вікні підтвердження натиснути "Так" / "Підтвердити".	Користувач успішно видаляється. Користувача перенаправляє на сторінку реєстрації/входу, або відображається повідомлення про успішне видалення облікового запису.
1.5	Видалення користувача (без підтвердження)	1. Перейти на сторінку профілю. 2. Натиснути кнопку "Видалити користувача". 3. У вікні натиснути "Ні" / "Скасувати".	Користувач не видаляється. Користувач залишається на сторінці профілю.

Таблиця 5.1 продовження – Тест-кейс №1 (таблиця виконана самостійно)

№	Назва тест-кейсу	Кроки	Очікуваний результат
2.1	Перегляд списку доходів	1. Перейти на сторінку "Доходи".	Відображається список усіх раніше доданих доходів з деталями (сума, дата, категорія, опис). Можливість сортування та фільтрації (якщо реалізовано).
2.2	Додавання нового доходу	1. Перейти на сторінку "Доходи". 2. Натиснути кнопку "Додати дохід". 3. Заповнити всі обов'язкові поля: сума (позитивне число), дата, обрати категорію, додати опис (необов'язково). 4. Зберегти.	Новий дохід успішно додається до списку та відображається на сторінці. Відображається повідомлення про успіх. Загальний баланс оновлюється.
2.3	Додавання доходу з невалідними даними	1. Перейти на сторінку "Доходи". 2. Натиснути кнопку "Додати дохід". 3. Залишити поле "Сума"порожнім/ввести від'ємне число/букви. 4. Не обрати категорію. 5. Зберегти.	З'являються повідомлення про помилки валідації для відповідних полів. Дохід не додається.
2.4	Редагування існуючого доходу	1. Перейти на сторінку "Доходи". 2. Знайти існуючий дохід. 3. Натиснути "Редагувати". 4. Змінити суму/дату/ категорію. 5. Зберегти.	Зміни зберігаються. Інформація про дохід оновлюється у списку. Відображається повідомлення про успіх. Загальний баланс оновлюється.

Таблиця 5.1 продовження – Тест-кейс №1 (таблиця виконана самостійно)

№	Назва тест-кейсу	Кроки	Очікуваний результат
2.5	Видалення доходу	1. Перейти на сторінку "Доходи". 2. Знайти існуючий дохід. 3. Натиснути кнопку "Видалити" біля нього. 4. Підтвердити видалення.	Дохід успішно видаляється зі списку. Відображається повідомлення про успіх. Загальний баланс оновлюється.
2.6	Пошук/Фільтрація доходів	1. Перейти на сторінку "Доходи". 2. Ввести критерії пошуку/фільтрації (наприклад, за категорією, датою). 3. Застосувати фільтр.	Відображаються лише доходи, які відповідають критеріям пошуку/фільтрації.

Отже, тестування програмного забезпечення відіграє одну з ключових ролей у забезпеченні успішного випуску продукту на ринок, дозволяючи виявити та усунути можливі проблеми ще до того, як вони впливають на користувачів [8].

6. ВПРОВАДЖЕННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

Впровадження програмного забезпечення для ведення та планування домашнього бюджету передбачає обов'язкове послідовне виконання декількох етапів, які спрямовані на забезпечення стабільної роботи системи, зручність для кінцевого користувача та відповідність нашим поставленим спочатку функціональним і нефункціональним вимогам.

5.1 Підготовка середовища

Застосунок було розроблено з урахуванням можливості роботи на популярних настільних операційних системах, таких як Windows та Linux. Після вибору платформи користувач завантажує архів застосунку або виконує клонування репозиторію з GitHub чи іншого джерела. Далі здійснюється встановлення необхідних бібліотек та компонентів відповідно до вказаного у проєкті списку залежностей.

5.2 Інсталяція та первинне налаштування

Після запуску система пропонує ввести початкові дані: джерела доходу, основні категорії витрат, валюту, мову інтерфейсу та фінансові цілі.

5.3 Початкове навчання користувача

Для спрощення адаптації користувача може бути передбачено коротке інструктивне введення або допоміжний розділ з підказками - пояснюється, як додавати транзакції, переглядати звіти, планувати бюджет та отримувати сповіщення про перевищення лімітів. У випадку даного проекту буде створено навчальний посібник користування додатком.

5.4 Тестування в реальних умовах

Щоб перевірити валідність та якість розробки веб-застосунку з планування домашнього бюджету використовувалось ручне тестування.

Застосунок перевіряється мануально в умовах реального використання: з внесенням щоденних витрат і доходів, формуванням щомісячних звітів і динаміки змін бюджету.

Мануальне тестування охоплює як функціональні, так і нефункціональні аспекти, оскільки при розробці додатку було важливо враховувати обидва типи тестування для розробки якісного продукту.

5.5 Отримання зворотного зв’язку від користувачів та покращення продукту

Планується зробити додаткову форму зворотнього зв'язку для користувачів. На основі відгуків буде проводитись подальше вдосконалення функціональності - таких як, додавання нових категорій витрат або можливості синхронізації з банківськими сервісами.

ВИСНОВКИ

Розробка веб-застосунку для планування домашнього бюджету на JavaScript з використанням технології React відповідає сучасним вимогам до зручності та функціональності фінансових інструментів. Відповідно до тенденцій у розробці персоналізованих програм для управління фінансами, створений застосунок дає змогу користувачам інтерактивно управляти своїми фінансами, вести облік доходів і витрат, створювати фінансові категорії та генерувати звіти. Використання технології JSON для зберігання даних забезпечує гнучкість і масштабованість застосунку.

Рішення, прийняті під час розробки (наприклад, використання бібліотеки Chart.js для візуалізації фінансових даних), відповідають сучасним вимогам до користувацького інтерфейсу та зручності роботи з даними. Можливість адаптувати застосунок під різні операційні системи за допомогою React також відповідає сучасним стандартам, забезпечуючи універсальність програмного продукту.

Розроблений застосунок має широкий спектр застосування. Він може бути використаний як для особистого фінансового планування, так і для навчання фінансової грамотності серед учнів та студентів. Застосунок може бути адаптований для різних категорій користувачів завдяки гнучкості налаштувань категорій витрат і доходів.

Можливі сфери використання:

- Особисте фінансове планування: для індивідуальних користувачів, які хочуть вести облік своїх доходів та витрат.
- Навчання фінансовій грамотності: використання застосунку в освітніх закладах для проведення практичних занять з управління фінансами.
- Малий бізнес: застосунок може бути адаптований для невеликих підприємств для обліку фінансових операцій.

Розробка веб-застосунку для планування бюджету є лише одним із етапів дослідження в області фінансових інструментів для особистого використання. Подальші дослідження можуть бути спрямовані на:

- Розширення функціональності застосунку (додавання нових можливостей для аналізу фінансових даних).
- Поглиблену інтеграцію з іншими фінансовими платформами та банківськими сервісами для автоматизації обліку.
- Розробку мобільної версії застосунку для забезпечення доступу до особистих фінансів на всіх пристроях користувача.

Розробка такого програмного забезпечення має значну наукову та соціально-економічну значущість. Це не лише вдосконалює фінансове планування на індивідуальному рівні, а й сприяє підвищенню рівня фінансової грамотності. Застосунок допомагає користувачам розуміти і контролювати свої фінанси, що є важливим аспектом для забезпечення стабільності особистих фінансів у умовах економічної нестабільності.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Шефер Б. Шлях до фінансової свободи. Ваш перший мільйон за сім років. - С. 4.
2. Фінансова грамотність: основи, правила, як навчитися [Електронний ресурс]. - Режим доступу: https://business-broker.com.ua/blog/finansova-hramotnist-osnovy-pravyla-iak-navchytys/.
3. Фінансова грамотність: основи та правила поводження з грошима [Електронний ресурс]. - Режим доступу: https://ideabank.ua/uk/experts/finansova-hramotnist-osnovy-ta-pravyla-povodzhennya-z-hroshyma.
4. Desktop-додаток: переваги та перспективи використання [Електронний ресурс]. - Режим доступу: https://wezom.com.ua/ua/blog/desktop-dodatok.
5. JavaScript – сучасний підручник [Електронний ресурс]. - Режим доступу: https://uk.javascript.info/.
6. About Electron [Електронний ресурс]. - Electron, 2017. - Архів оригіналу за 1 квітня 2017. - Процитовано 31 березня 2017. - [Архівовано 2017-04-01 у Wayback Machine]. - Режим доступу: https://www.electronjs.org/.
7. Migrations [Електронний ресурс]. - Режим доступу: https://sequelize.org/docs/v6/other-topics/migrations/
8. Що таке тестування ПЗ, його етапи, види, інструменти [Електронний ресурс]. - Режим доступу: https://university.sigma.software/what-is-software-testing/
9. Install mongosh [Електронний ресурс]. - https://www.mongodb.com/docs/mongodb-shell/install/

10. Західноукраїнський національний університет. Опорний конспект лекцій [Електронний ресурс]. Режим доступу: https://dspace.wunu.edu.ua/bitstream/316497/24194/1/%D0%BE%D0%BF%D0%BE%D1%80%D0%BD%D0%B8%D0%B9%20%D0%BA%D0%BE%D0%BD%D1%81%D0%BF%D0%B5%D0%BA%D1%82%20%D0%BB%D0%B5%D0%BA%D1%86%D1%96%D0%B9.pdf
11. Що таке Back-end розробка [Електронний ресурс]. - Режим доступу: https://wezom.com.ua/ua/blog/chto-takoe-back-end-razrabotka
12. React [Електронний ресурс]. - Режим доступу: https://uk.legacy.reactjs.org/tutorial/tutorial.html
13. Introducing Node [Електронний ресурс]. - Режим доступу: https://developer.mozilla.org/en-US/docs/Learn_web_development/Extensions/Server-side/Express_Nodejs/Introduction

ДОДАТОК А

Звіт результатів перевірки на унікальність тексту в мережі інтернет та базі

ДОДАТОК Б

Слайди презентації

Мета роботи:

Створити персоналізований веб-застосунок для планування домашнього бюджету, що дозволяє:

- вести облік доходів і витрат;
- створювати фінансові категорії;
- формувати звіти та візуалізувати дані.

Для реалізації використано JavaScript (бібліотека React), JSON Rest API для обміну даних та сучасні веб-технології для зручного інтерфейсу.

Актуальність роботи полягає у потребі ефективного управління особистими фінансами в умовах економічної нестабільності.

Фінансова грамотність допомагає уникати криз, підвищувати якість життя та розвивати підприємницькі навички.

Розробка простого, доступного та персоналізованого веб-застосунку сприяє підвищенню фінансової культури та цифрової адаптації населення.

2

Аналіз проблеми та аналіз існуючих рішень

Суть проблеми:
Планування домашнього бюджету – ключовий елемент фінансової грамотності. Користувачі потребують простих, зручних і гнучких інструментів для обліку доходів та витрат.

Досліджені конкуренти:
- Mint - складний інтерфейс, орієнтований на США
- YNAB - платна модель, не підтримує українську
- HomeBank, Money Manager EX - десктопні, неадаптовані до мобільних платформ

Виявлені проблеми існуючих рішень:
- складність використання для новачків
- обмежений функціонал у безкоштовних версіях
- залежність від інтернету
- відсутність локалізації
  складна установка на різних ОС

Доцільність розробки власного застосунку:
- інтуїтивний інтерфейс
- підтримка української мови
- автономний режим
- простота використання
- адаптивність і відкритість до розширення

3

Постановка задачі та опис системи

Мета: створити веб-додаток для планування домашнього бюджету з функціями обліку доходів і витрат, створення категорій, генерації звітів та візуалізації даних.

Функції системи:
- Облік фінансових операцій
- Категоризація витрат
- Графіки й звіти
- Сімейна підписка

Особливості:
- Працює у браузері
- Підтримка offline-режиму
- Автоматичні оновлення
- Доступ з різних пристроїв

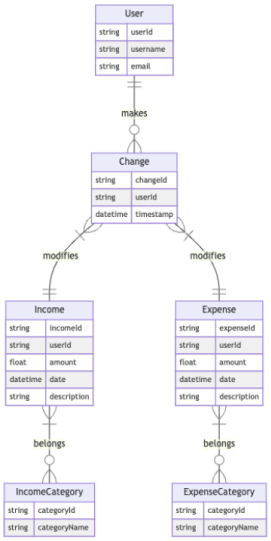
Технології:
React, JavaScript, Node.js, MongoDB, JSON

Вибір технологій розробки

Основні інструменти:
- **JavaScript** – основна мова логіки застосунку, обробка даних у реальному часі
- **React** – побудова динамічного інтерфейсу, компонентна архітектура, Virtual DOM
- **JSON Rest API** – зберігання та обмін даними між компонентами
- **Chart.js** – візуалізація даних (графіки, діаграми)

Принципи роботи застосунку:
- Обробка й оновлення даних у реальному часі
- Персоналізація категорій витрат/доходів
- Автоматична генерація звітів
- Збереження даних у форматі JSON
- Кросплатформеність і доступність з різних пристроїв

Архітектура створеного програмного забезпечення



Модель даних включає основні сутності веб-додатку для планування бюджету:

- **User** – зберігає інформацію про користувача (ID, ім'я, email)
- **Income / Expense** – записи про доходи та витрати з сумою, датою та описом
- **IncomeCategory / ExpenseCategory** – категорії для класифікації трансакцій
- **Change** – фіксує зміни, які користувач вносить у свої фінансові записи

Кожна транзакція (дохід або витрата) належить певній категорії та пов'язана з користувачем. Усі зміни логуються для відстеження активності.

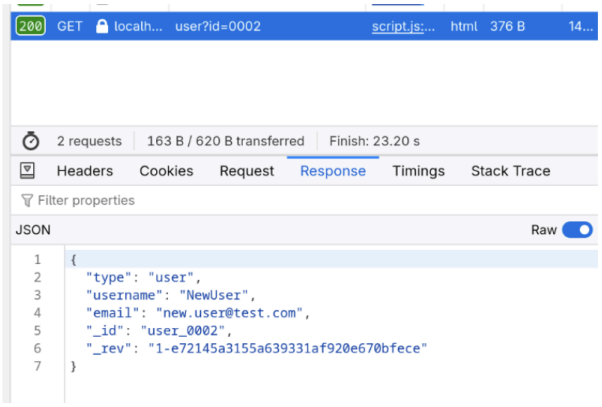
6

Опис програмного забезпечення, що було використано у дослідженні

- **JavaScript** - основна мова логіки, обробка даних у реальному часі
- **React** - побудова інтерфейсу, компонентна структура, Virtual DOM
- **JSON Rest API** - збереження фінансових операцій (доходи, витрати, категорії)
- **Chart.js** - побудова графіків і візуалізація фінансових даних

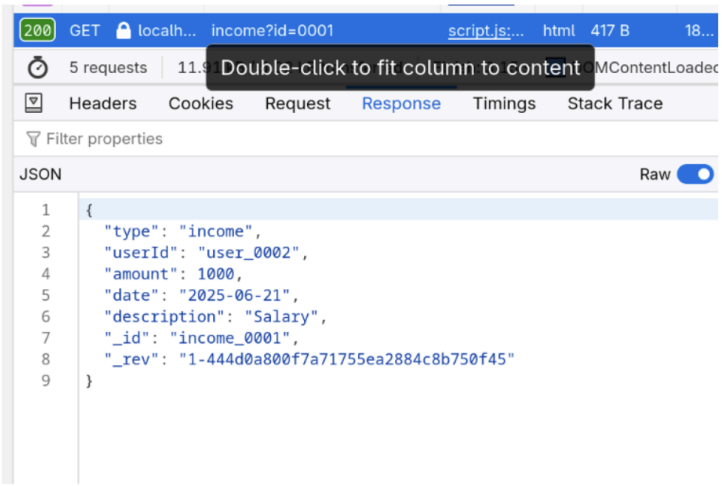
7

Приклад реалізації (Приклад запиту до бази даних (JSON-відповідь))



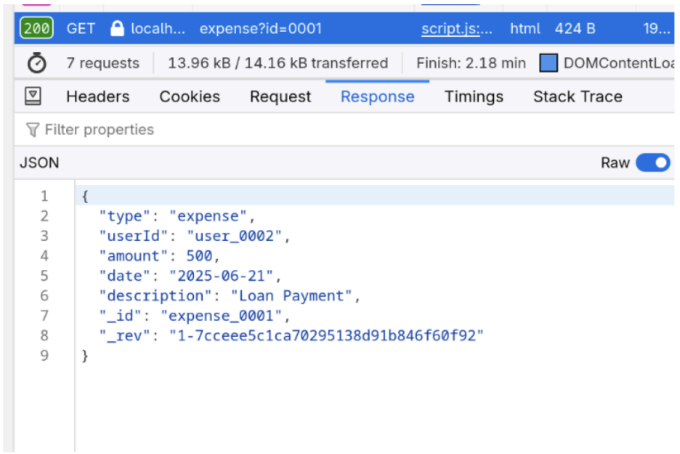
- Успішний GET-запит до локального сервера (user id=0002);
- Сервер повертає дані користувача у форматі JSON;
- Відображено у вкладці **Response** в інструментах розробника браузера;
- Формат JSON використовується для обміну та зберігання даних у застосунку.

Приклад реалізації (Приклад отримання фінансових даних (дохід))



- GET-запит до сервера income id=0001;
- Відповідь у форматі JSON містить дані про дохід користувача;
- Дані використовуються для формування звітів, графіків та аналітики в застосунку;
- Формат JSON забезпечує зручний обмін інформацією між клієнтом і сервером.

Приклад реалізації (Отримання даних витрати з бази (GET-запит))



Повертається JSON-об'єкт з типом "expense", сумою, датою, описом та ідентифікатором користувача.

10

Тестування

№	Назва тест-кейсу	Кроки	Очікуваний результат
1.1	Перегляд профілю користувача	1. Перейти на сторінку профілю користувача.	Відображається актуальна інформація про користувача (ім'я, email тощо).
1.2	Редагування профілю користувача	1. Перейти на сторінку профілю. 2. Натиснути кнопку "Редагувати". 3. Внести зміни в поля (наприклад, ім'я). 4. Зберегти зміни.	Зміни успішно зберігаються та відображаються у профілі. Відображається повідомлення про успішне оновлення.
1.3	Редагування профілю з невалідними даними	1. Перейти на сторінку профілю. 2. Натиснути кнопку "Редагувати". 3. Ввести невалідні дані (наприклад, email без "@", порожнє ім'я). 4. Зберегти зміни.	З'являються повідомлення про помилки валідації для відповідних полів. Зміни не зберігаються.
1.4	Видалення користувача (з підтвердженням)	1. Перейти на сторінку профілю. 2. Натиснути кнопку "Видалити користувача". 3. У вікні підтвердження натиснути "Так" / "Підтвердити".	Користувач успішно видаляється. Користувача перенаправляє на сторінку реєстрації/входу, або відображається повідомлення про успішне видалення облікового запису.

1.1 Перегляд профілю: перевірка відображення актуальної інформації.

1.2 Редагування профілю: перевірка успішного збереження змін.

1.3 Редагування з невалідними даними: перевірка валідації форм.

1.4 Видалення користувача: перевірка підтвердження та перенаправлення після видалення.

11

Підсумки

- **Створено веб-застосунок** для обліку доходів і витрат з використанням JavaScript, React, JSON Rest API
- **Функції**: персоналізовані категорії, генерація звітів, графіки, збереження даних
- **Переваги**: простота, кросплатформеність, адаптивність
- **Сфери застосування**: особисте користування, навчання фінансової грамотності, малий бізнес
- **Перспективи**: мобільна версія, інтеграція з банками, розширення функцій

ДОДАТОК В

Лістинг програмного коду

backend/server.js

```
const express = require('express');
const bodyParser = require('body-parser');
const path = require('path');
const cors = require('cors');
const budgetController = require('./controllers/budgetController');
const app = express();
const port = 3030;
// Налаштування Middleware
app.use(bodyParser.urlencoded({ extended: true }));
app.use(bodyParser.json());
app.use(cors());
// Serve static files from the 'public' folder
app.use(express.static(path.join(__dirname, 'public')));
// Rest Api
// API для користувача
app.post('/api/user', budgetController.addUser);
app.put('/api/user', budgetController.updateUser);
app.get('/api/user', budgetController.readUser);
app.delete('/api/user', budgetController.deleteUser);
// API для роботи з доходами
app.post('/api/income', budgetController.addIncome);
app.put('/api/income', budgetController.updateIncome);
app.get('/api/income', budgetController.readIncome);
app.delete('/api/income', budgetController.deleteIncome);
//API для роботи з витратами
app.post('/api/expense', budgetController.addExpense);
app.put('/api/expense', budgetController.updateExpense);
app.get('/api/expense', budgetController.readExpense);
app.delete('/api/expense', budgetController.deleteExpense);
//API для роботи з категоріями доходів
app.post('/api/incomeCategory', budgetController.addIncomeCategory);
app.put('/api/incomeCategory',
budgetController.updateIncomeCategory);
```

```
app.get('/api/incomeCategory', budgetController.readIncomeCategory);
app.delete('/api/incomeCategory',
budgetController.deleteIncomeCategory);
//API для роботи з карегоріями витрат
app.post('/api/expenseCategory',
budgetController.addExpenseCategory);
app.put('/api/expenseCategory',
budgetController.updateExpenseCategory);
app.get('/api/expenseCategory',
budgetController.readExpenseCategory);
app.delete('/api/expenseCategory',
budgetController.deleteExpenseCategory);
// Start server
app.listen(port, () => {
    console.log(`Server is running on http://localhost:${port}`);
});

backend/controllers/budgetController.js
const budgetApp = require('../repository/BudgetDB');
// Controller methods
const addUser = (req, res) => {
    const { userId, username, email } = req.body;
          budgetApp.addUser(userId,  username,  email).then(()  =>
res.send(`User ${userId} added.`)).catch(err => {
        res.send(`Err: User ${userId} already exists.`);
    });
};
const updateUser = (req, res) => {
    const { id, username, email } = req.body;
           budgetApp.updateUser(id,  username,  email).then(()  =>
res.send(`User ${id} updated.`)).catch(err => {
        res.send(`Err: User ${id} doesn't exists.`);
    });
};
const readUser = (req, res) => {
    const userId = req.query.id;
```

```
                budgetApp.readUser(userId).then((user) =>
res.send(JSON.stringify(user, null, 2))).catch(err => {
            res.send(`Err: Can't get user ${userId}`);
        });
    };
    const deleteUser = (req, res) => {
        const userId = req.query.id;
        budgetApp.deleteUser(userId).then(() => res.send(`User ${userId}
is delete sucessfully.`)).catch(err => {
            res.send(err);
        });
    };
    // Income
    const addIncome = (req, res) => {
         const { incomeId, userId, amount, date, description, categoryId
} = req.body;
        budgetApp.addIncome(incomeId, userId, amount, date, description,
categoryId)
            .then(() => res.send(`Income ${incomeId} added.`)).catch(err
=> {
                res.send(err);
            });
    };
    const updateIncome = (req, res) => {
           const { incomeId, userId, amount, date, description } =
req.body;
             budgetApp.updateIncome(incomeId, userId, amount, date,
description)
                .then(() => res.send(`Income ${incomeId}
updated.`)).catch(err => {
                res.send(err);
            });
    };
    const readIncome = (req, res) => {
        const userId = req.query.userId;
                budgetApp.readIncome(userId).then((income) =>
res.send(JSON.stringify(income, null, 2))).catch(err => {
```

```
            res.send(`Err: Can't get income`);
        });
    };
    const deleteIncome = (req, res) => {
        const incomeId = req.query.id;
           budgetApp.deleteIncome(incomeId).then(() => res.send(`Income
${incomeId} is delete sucessfully.`)).catch(err => {
            res.send(`Err: Can't delete income ${incomeId}`);
        });
    };
    //Expense
    const addExpense = (req, res) => {
        const { expenseId, userId, amount, date, description, categoryId
} = req.body;
              budgetApp.addExpense(expenseId, userId, amount, date,
description, expenseId, categoryId).then(() => {
            res.send(`Expense added.`);
        }).catch(err => {
            res.send(err);
        });
    };
    const updateExpense = (req, res) => {
           const { expenseId, userId, amount, date, description } =
req.body;
            budgetApp.updateExpense(expenseId, userId, amount, date,
description).then(() => {
            es.send(`Expense ${expenseId} updated.`);
        }).catch(err => {
            res.send(err);
        });
    };
    const readExpense = (req, res) => {
        const userId = req.query.userId;
                    budgetApp.readExpense(userId).then((expense)    =>
res.send(JSON.stringify(expense, null, 2))).catch(err => {
            res.send(`Err: Can't get expense`);
        });
```

```
    };
    const deleteExpense = (req, res) => {
        const expenseId = req.query.id;
         budgetApp.deleteExpense(expenseId).then(() => res.send(`Expense
${expenseId} is delete sucessfully.`)).catch(err => {
            res.send(`Err: Can't delete expense ${expenseId}`);
        });
    };
    //Income Categories
    const addIncomeCategory = (req, res) => {
          const { categoryId, userId, categoryName, description } =
req.body;
          budgetApp.addIncomeCategory(categoryId, userId, categoryName,
description).then(() => {
            res.send(`Income category ${categoryId} added.`);
        }).catch(err => {
                res.send(`Err: Income category ${categoryId} already
exists.`);
        });
    };
    const updateIncomeCategory = (req, res) => {
        const { categoryId, name, description } = req.body;
                budgetApp.updateIncomeCategory(categoryId, name,
description).then(() => {
            res.send(`Income category ${categoryId} updated.`);
        }).catch(err => {
                res.send(`Err: Income category ${categoryId} already
exists.`);
        });
    };
    const readIncomeCategory = (req, res) => {
        const userId = req.query.userId;
                budgetApp.readIncomeCategory(userId).then((income) =>
res.send(JSON.stringify(income, null, 2))).catch(err => {
            res.send(`Err: Can't get income category`);
        });
    };
```

```
    const deleteIncomeCategory = (req, res) => {
        const incomeId = req.query.id;
                  budgetApp.deleteIncomeCategory(incomeId).then(()    =>
res.send(`Income category ${incomeId} is delete sucessfully.`)).catch(err
=> {
            res.send(`Err: Can't delete income category ${incomeId}`);
        });
    };

    //Expense categories
    const addExpenseCategory = (req, res) => {
          const  {  categoryId,  userId,  categoryName,  description  }  =
req.body;
         budgetApp.addExpenseCategory(categoryId, userId, categoryName,
description).then(() => {
            res.send(`Expense category ${categoryId} added.`);
        }).catch(err => {
                res.send(`Err:  Expense  category  ${categoryId}  already
exists.`);
        });
    };
    const updateExpenseCategory = (req, res) => {
        const { categoryId, categoryName, description } = req.body;
             budgetApp.updateExpenseCategory(categoryId,  categoryName,
description).then(() => {
            res.send(`Expense category ${categoryId} updated.`);
        }).catch(err => {
                res.send(`Err:  Expense  category  ${categoryId}  already
exists.`);
        });
    };
    const readExpenseCategory = (req, res) => {
        const userId = req.query.userId;
               budgetApp.readExpenseCategory(userId).then((expense)   =>
res.send(JSON.stringify(expense, null, 2))).catch(err => {
            res.send(`Err: Can't get expense category`);
        });
```

```
};
const deleteExpenseCategory = (req, res) => {
    const expenseId = req.query.id;
            budgetApp.deleteExpenseCategory(expenseId).then(() =>
res.send(`Expense category ${expenseId} is delete
sucessfully.`)).catch(err => {
        res.send(`Err: Can't delete expense category ${expenseId}`);
    });
};
module.exports = {
    readUser,
    addUser,
    updateUser,
    deleteUser,
    readIncome,
    addIncome,
    updateIncome,
    deleteIncome,
    readExpense,
    addExpense,
    updateExpense,
    deleteExpense,
    readIncomeCategory,
    addIncomeCategory,
    updateIncomeCategory,
    deleteIncomeCategory,
    readExpenseCategory,
    addExpenseCategory,
    updateExpenseCategory,
    deleteExpenseCategory
};
backend/repository/BudgetDB.js
const PouchDB = require('pouchdb');
//додавання плагіну для пошуку в базі
PouchDB.plugin(require('pouchdb-find'));
//створення локальної бази даних
const db = new PouchDB('userDB');
```

```
    //створення індексу для бази даних
    db.createIndex({
        index: { fields: ['userId', 'type'] }
    });

    //реалізація CRUD операцій для роботи з інформацією користувача
    const addUser = function (userId, username, email) {
        return new Promise((resolve, reject) => {
            db.put({
                _id: `user_${userId}`,
                type: "user",
                username: username,
                email: email
            })
                .then(() => {
                    resolve();
                })
                .catch(function (err) {
                    reject(err);
                });
        });
    }
    const readUser = function (userId) {
        return new Promise((resolve, reject) => {
            db.get(`user_${userId}`).then(function (user) {
                resolve(user);
            }).catch(function (err) {
                reject(`User ${userId} doesn't exists.`);
            });
        });
    }
    const deleteUser = function (userId) {
           return  readUser(userId).then(user  =>  new  Promise((resolve,
reject) => {
            const related = db.find({
                selector: {
                    userId: user._id
```

```
                }
            }).then((data) => {
                if (data.docs.length > 0) {
                       reject(`Can't remove user ${userId}, some data is
associated with the user: \n` + JSON.stringify(data.docs, null, 2));
                } else {
                    db.remove(user._id, user._rev).then(function () {
                        resolve();
                    }).catch(function () {
                        reject(`Cant remove user ${userId}.`);
                    });
                }
            })
        }));
    }
    const updateUser = function (userId, username, email) {
        return new Promise((resolve, reject) => {
            db.get(`user_${userId}`).then(function (user) {
                return db.put({
                    _id: `user_${userId}`,
                    _rev: user._rev,
                    type: "user",
                    username: username,
                    email: email
                });
            }).then(function () {
                resolve();
            }).catch(function (err) {
                reject(err);
            });
        });
    }
    const readIncome = function (userId) {
        return new Promise((resolve, reject) => {
            db.find({
                selector: {
                    type: 'income',
```

```
                    userId: `user_${userId}`
                }
            }).then((incCat) => {
                resolve(incCat.docs);
            })
                .catch(function (err) {
                    reject(err);
                });
        });
    }
    const addIncome = function (incomeId, userId, amount, date,
description, categoryId) {
        return readUser(userId).then(() => new Promise((resolve, reject)
=> {
            db.put({
                _id: `income_${incomeId}`,
                type: "income",
                userId: `user_${userId}`,
                amount: amount,
                date: date,
                description: description,
                categoryId: `${categoryId}`
            })
                .then(() => {
                    resolve();
                })
                .catch(function (err) {
                    reject(`Income ${incomeId} already exists`);
                });
        }));
    }
    const updateIncome = function (incomeId, userId, amount, date,
description) {
        return readUser(userId).then(() => new Promise((resolve, reject)
=> {
            db.get(`income_${incomeId}`).then((income) => {
                return db.put({
```

```
                    _id: `income_${incomeId}`,
                    _rev: income._rev,
                    type: "income",
                    userId: `user_${userId}`,
                    amount: amount,
                    date: date,
                    description: description
                })
            })
                .then(() => {
                    resolve();
                })
                .catch(function (err) {
                    reject(`Income ${incomeId} doesn't exists`);
                });
        }));
    }
    const deleteIncome = function (incomeId) {
        return new Promise((resolve, reject) => {
            db.get(incomeId)
                .then((doc) => {
                    db.remove(doc).then(() => (resolve()))
                        .catch(function (e) {
                                reject(`Can't remove ${incomeId} doesn't
exists`);
                        });
                })
                .catch(function (e) {
                    reject(`Income ${incomeId} doesn't exists`);
                });
        });
    }
    const readExpense = function (userId) {
        return new Promise((resolve, reject) => {
            db.find({
                selector: {
                    type: "expense",
```

```
                    userId: `user_${userId}`
                }
            }).then((incCat) => {
                resolve(incCat.docs);
            })
                .catch(function (err) {
                    reject(err);
                });
        });
    }
    const addExpense = function (expenseId, userId, amount, date,
description, categoryId) {
        return readUser(userId).then(() => new Promise((resolve, reject)
=> {
            db.put({
                _id: `expense_${expenseId}`,
                type: "expense",
                userId: `user_${userId}`,
                amount: amount,
                date: date,
                description: description,
                categoryId: `${categoryId}`
            })
                .then(() => {
                    resolve();
                })
                .catch(function (err) {
                    reject(`Expense ${expenseId} already exists`);
                });
        }));
    }

    const updateExpense = function (expenseId, userId, amount, date,
description) {
        return readUser(userId).then(() => new Promise((resolve, reject)
=> {
            db.get(`expense_${expenseId}`).then((expense) => {
```

```
                return db.put({
                    _id: `expense_${expenseId}`,
                    _rev: expense._rev,
                    type: "expense",
                    userId: `user_${userId}`,
                    amount: amount,
                    date: date,
                    description: description
                });
            })
                .then(() => {
                    resolve();
                })
                .catch(function (err) {
                    reject(`Expense ${expenseId} doesn't exists`);
                });
        }));
    }
    const deleteExpense = function (expenseId) {
        return new Promise((resolve, reject) => {
            db.get(expenseId)
                .then((doc) => {
                    db.remove(doc).then(() => (resolve()))
                        .catch(function (e) {
                            reject(`Can't remove ${expenseId} doesn't
exists`);
                        });
                })
                .catch(function (err) {
                    reject(err);
                });
        });
    }
    const addIncomeCategory = function (categoryId, userId,
categoryName, description) {
        return new Promise((resolve, reject) => {
            db.put({
```

```
                _id: `incomeCat_${categoryId}`,
                type: "incomeCat",
                userId: `user_${userId}`,
                categoryName: categoryName,
                description: description
            })
                .then(() => {
                    resolve();
                })
                .catch(function (err) {
                    reject(err);
                });
        });
    }
    const updateIncomeCategory = function (categoryId, categoryName,
description) {
        return new Promise((resolve, reject) => {
            db.get(`incomeCat_${categoryId}`).then(inCat => {
                return db.put({
                    _id: `incomeCat_${categoryId}`,
                    _rev: inCat._rev,
                    type: "incomeCat",
                    categoryName: categoryName,
                    description: description
                })
            })
                .then(() => {
                    resolve();
                })
                .catch(function (err) {
                    reject(err);
                });
        });
    }
    const readIncomeCategory = function (userId) {
        return new Promise((resolve, reject) => {
            db.find({
```

```
                selector: {
                    type: 'incomeCat',
                    userId: `user_${userId}`
                }
            }).then((incCat) => {
                resolve(incCat.docs);
            })
                .catch(function (err) {
                    reject(err);
                });
        });
    }
    const deleteIncomeCategory = function (categoryId) {
        return new Promise((resolve, reject) => {
            db.get(categoryId)
                .then( (doc) => {
                    db.remove(doc).then(() => (resolve()))
                    .catch(function (e) {
                              reject(`Can't remove ${categoryId} doesn't
exists`);
                    });
                })
                .catch(function (err) {
                    reject(err);
                });
        });
    }

    const   addExpenseCategory   =   function   (categoryId,   userId,
categoryName, description) {
        return new Promise((resolve, reject) => {
            db.put({
                _id: `expenseCat_${categoryId}`,
                type: "expenseCat",
                userId: `user_${userId}`,
                categoryName: categoryName,
                description: description
```

```
            })
                .then(() => {
                    resolve();
                })
                .catch(function (err) {
                    reject(err);
                });
        });
    }
    const updateExpenseCategory = function (categoryId, categoryName,
description) {
        return new Promise((resolve, reject) => {
            db.get(`expenseCat_${categoryId}`).then(expCat => {
                return db.put({
                    _id: `expenseCat_${categoryId}`,
                    _rev: expCat._rev,
                    type: "expenseCat",
                    categoryName: categoryName,
                    description
                });
            })
                .then(() => {
                    resolve();
                })
                .catch(function (err) {
                    reject(err);
                });
        });
    }
    const readExpenseCategory = function (userId) {
        return new Promise((resolve, reject) => {
            db.find({
                selector: {
                    type: 'expenseCat',
                    userId: `user_${userId}`
                }
            }).then((expCat) => {
```

```
                resolve(expCat.docs);
            })
                .catch(function (err) {
                    reject(err);
                });
        });
    }
    const deleteExpenseCategory = function (categoryId) {
        return new Promise((resolve, reject) => {
            db.get(categoryId)
                .then( (doc) => {
                    db.remove(doc).then(() => (resolve()))
                    .catch(function (e) {
                              reject(`Can't remove ${categoryId} doesn't
exists`);
                    });
                }).catch(function (err) {
                reject(err);
            });
        });
    }
    module.exports = {
        readUser,
        addUser,
        updateUser,
        deleteUser,
        readIncome,
        addIncome,
        updateIncome,
        deleteIncome,
        readExpense,
        addExpense,
        updateExpense,
        deleteExpense,
        readIncomeCategory,
        addIncomeCategory,
        updateIncomeCategory,
```

```
    deleteIncomeCategory,
    readExpenseCategory,
    addExpenseCategory,
    updateExpenseCategory,
    deleteExpenseCategory
};

ui/src/index.tsx
import React from 'react';
import ReactDOM from 'react-dom/client';
import './index.css';
import App from './App';
import reportWebVitals from './reportWebVitals';
const root = ReactDOM.createRoot(
  document.getElementById('root') as HTMLElement
);
root.render(
  <React.StrictMode>
    <App />
  </React.StrictMode>
);

// If you want to start measuring performance in your app, pass a
function
// to log results (for example: reportWebVitals(console.log))
//   or   send   to   an   analytics   endpoint.   Learn   more:
https://bit.ly/CRA-vitals
reportWebVitals();
ui/src/App.tsx
import React from 'react';
import logo from './logo.svg';
import './App.css';
import BudgetApp from './BudgetApp';
function App() {
  return (
    <div className="App">
        <BudgetApp/>
```

```
        </div>
      );
    }
    export default App;
    ui/src/BudgetApp.tsx
    import React, { useState, useEffect, useRef } from 'react';
    import { Plus, Edit2, Trash2, DollarSign, TrendingUp, TrendingDown,
User, Tag } from 'lucide-react';
    import * as Chart from 'chart.js';

    const CURRENT_USER_ID = '0004';
    const HOST = 'http://localhost:3030';
    // Реалізація виклику API
    const api: any = {
      // для роботи з інформацією користувача
      user: {
        get: async () => {
              const response = await fetch(HOST + '/api/user?id=' +
CURRENT_USER_ID);
          return response.json();
        },
        update: (data: { userId: string, username: string, email: string
}) => fetch(HOST + '/api/user', {
          method: 'PUT',
          headers: {
            'Content-Type': 'application/json'
          },
          body: JSON.stringify({ ...data, id: CURRENT_USER_ID })
        })
      },
      // для роботи з інформацією про доходи
      income: {
        get: async () => {
            const response = await fetch(HOST + '/api/income?userId=' +
CURRENT_USER_ID);
          return response.json();
        },
```

```
        post: (data: any) => fetch(HOST + '/api/income', {
          method: 'POST',
          headers: {
            'Content-Type': 'application/json'
          },
           body: JSON.stringify({ ...data, incomeId: Date.now(), userId:
CURRENT_USER_ID })
        }),
        update: (data: any) => fetch(HOST + '/api/income', {
          method: 'PUT',
          headers: {
            'Content-Type': 'application/json'
          },
          body: JSON.stringify({ ...data, userId: CURRENT_USER_ID })
        }),
        delete: (id: string) => fetch(HOST + '/api/income?id=' + id, {
          method: 'DELETE',
          headers: {
            'Content-Type': 'application/json'
          }
        })
      },
      // для роботи з інформацією про витрати
      expense: {
        get: async () => {
           const response = await fetch(HOST + '/api/expense?userId=' +
CURRENT_USER_ID);
          return response.json();
        },
        post: (data: any) => fetch(HOST + '/api/expense', {
          method: 'POST',
          headers: {
            'Content-Type': 'application/json'
          },
          body: JSON.stringify({ ...data, expenseId: Date.now(), userId:
CURRENT_USER_ID })
        }),
```

```
        update: (data: any) => fetch(HOST + '/api/expense', {
          method: 'PUT',
          headers: {
            'Content-Type': 'application/json'
          },
          body: JSON.stringify({ ...data, userId: CURRENT_USER_ID })
        }),
        delete: (id: string) => fetch(HOST + '/api/expense?id=' + id, {
          method: 'DELETE',
          headers: {
            'Content-Type': 'application/json'
          }
        })
      },
      // для роботи з інформацією про категорії доходів
      incomeCategory: {
        get: async () => {
                        const response = await fetch(HOST +
'/api/incomeCategory?userId=' + CURRENT_USER_ID);
          return response.json();
        },
        post: (data: any) => fetch(HOST + '/api/incomeCategory', {
          method: 'POST',
          headers: {
            'Content-Type': 'application/json'
          },
              body: JSON.stringify({ ...data, categoryId: Date.now(),
userId: CURRENT_USER_ID })
        }),
        update: (data: any) => fetch(HOST + '/api/incomeCategory', {
          method: 'PUT',
          headers: {
            'Content-Type': 'application/json'
          },
          body: JSON.stringify({ ...data, userId: CURRENT_USER_ID })
        }),
```

```
          delete: (id: string) => fetch(HOST + '/api/incomeCategory?id=' +
id, {
            method: 'DELETE',
            headers: {
              'Content-Type': 'application/json'
            }
          }
          )
        },
        // для роботи з інформацією про категорії витрат
        expenseCategory: {
          get: async () => {
                          const   response   =   await   fetch(HOST   +
'/api/expenseCategory?userId=' + CURRENT_USER_ID);
            return response.json();
          },
          post: (data: any) => fetch(HOST + '/api/expenseCategory', {
            method: 'POST',
            headers: {
              'Content-Type': 'application/json'
            },
                body: JSON.stringify({ ...data, categoryId: Date.now(),
userId: CURRENT_USER_ID })
          }),
          update: (data: any) => fetch(HOST + '/api/expenseCategory', {
            method: 'PUT',
            headers: {
              'Content-Type': 'application/json'
            },
            body: JSON.stringify({ ...data, userId: CURRENT_USER_ID })
          }),
           delete: (id: string) => fetch(HOST + '/api/expenseCategory?id='
+ id, {
            method: 'DELETE',
            headers: {
              'Content-Type': 'application/json'
            }
```

```
        }
        )
      }
    };
    const BudgetApp = () => {
      const [activeTab, setActiveTab] = useState('dashboard');
      const [user, setUser] = useState<any | null>(null);
      const [income, setIncome] = useState<any>([]);
      const [expenses, setExpenses] = useState<any>([]);
      const [incomeCategories, setIncomeCategories] = useState<any>([]);
            const    [expenseCategories,    setExpenseCategories]    =
useState<any>([]);
      const [loading, setLoading] = useState<boolean>(true);


      // посилання на графік
      const incomeChartRef: any = useRef(null);
      const expenseChartRef: any = useRef(null);
      const incomeChartInstance: any = useRef(null);
      const expenseChartInstance: any = useRef(null);
      // стейт для діалогового вікна
      const [showModal, setShowModal] = useState<boolean>(false);
      const [modalType, setModalType] = useState<string>('');
      const [editingItem, setEditingItem] = useState<any | null>(null);
      interface FormData {
        name?: string
        categoryName?: string
        username?: string
        description?: string
        email?: string
        amount?: number
        categoryId?: string
        date?: string
        _id?: string
      }
      // Form state
      const [formData, setFormData] = useState<FormData>({});
```

```
        const renderCharts = () => {
          // Register Chart.js components
          Chart.Chart.register(...Chart.registerables);
          // Destroy existing charts
          if (incomeChartInstance.current) {
            incomeChartInstance.current.destroy();
          }
          if (expenseChartInstance.current) {
            expenseChartInstance.current.destroy();
          }
          // Prepare income data by category
          const incomeByCategory: any = {};
          income.forEach((item: {categoryId: string, amount: number}) => {
            if (incomeByCategory[item.categoryId]) {
              incomeByCategory[item.categoryId] += item.amount;
            } else {
              incomeByCategory[item.categoryId] = item.amount;
            }
          });
          // Prepare expense data by category
          const expensesByCategory: any = {};
          expenses.forEach((item: {categoryId: string, amount: number}) =>
{
            if (expensesByCategory[item.categoryId]) {
              expensesByCategory[item.categoryId] += item.amount;
            } else {
              expensesByCategory[item.categoryId] = item.amount;
            }
          });
          // Income chart colors
          const incomeColors = [
            '#10B981', '#059669', '#047857', '#065F46', '#064E3B',
            '#6EE7B7', '#34D399', '#10B981', '#059669', '#047857'
          ];
          // Expense chart colors
          const expenseColors = [
            '#EF4444', '#DC2626', '#B91C1C', '#991B1B', '#7F1D1D',
```

```
            '#FCA5A5', '#F87171', '#EF4444', '#DC2626', '#B91C1C'
          ];
          // Render income chart
                                    if       (incomeChartRef.current       &&
Object.keys(incomeByCategory).length > 0) {
            const ctx = incomeChartRef.current.getContext('2d');
            incomeChartInstance.current = new Chart.Chart(ctx, {
              type: 'pie',
              data: {
                labels: Object.keys(incomeByCategory),
                datasets: [{
                  data: Object.values(incomeByCategory),
                                     backgroundColor:  incomeColors.slice(0,
Object.keys(incomeByCategory).length),
                  borderWidth: 2,
                  borderColor: '#ffffff'
                }]
              },
              options: {
                responsive: true,
                maintainAspectRatio: false,
                plugins: {
                  legend: {
                    position: 'bottom',
                    labels: {
                      padding: 20,
                      usePointStyle: true
                    }
                  },
                  tooltip: {
                    callbacks: {
                      label: function(context) {
                        const label = context.label || '';
                        const value = context.parsed;
                          const total = context.dataset.data.reduce((a, b)
=> a + b, 0);
```

```
                                const percentage = ((value / total) *
100).toFixed(1);
                                  return `${label}: ${value.toFixed(2)}
(${percentage}%)`;
                    }
                  }
                }
              }
            });
          }
          // відображаємо графік розходів
                              if (expenseChartRef.current &&
Object.keys(expensesByCategory).length > 0) {
            const ctx = expenseChartRef.current.getContext('2d');
            expenseChartInstance.current = new Chart.Chart(ctx, {
              type: 'pie',
              data: {
                labels: Object.keys(expensesByCategory),
                datasets: [{
                  data: Object.values(expensesByCategory),
                                  backgroundColor: expenseColors.slice(0,
Object.keys(expensesByCategory).length),
                  borderWidth: 2,
                  borderColor: '#ffffff'
                }]
              },
              options: {
                responsive: true,
                maintainAspectRatio: false,
                plugins: {
                  legend: {
                    position: 'bottom',
                    labels: {
                      padding: 20,
                      usePointStyle: true
                    }
```

```
                },
                tooltip: {
                  callbacks: {
                    label: function(context) {
                      const label = context.label || '';
                      const value = context.parsed;
                        const total = context.dataset.data.reduce((a, b)
=> a + b, 0);
                                const percentage = ((value / total) *
100).toFixed(1);
                                  return `${label}: ${value.toFixed(2)}
(${percentage}%)`;
                    }
                  }
                }
              }
            }
          });
        }
      };
      useEffect(() => {
        loadData();
      }, []);
      useEffect(() => {
        if (!loading && activeTab === 'dashboard') {
          renderCharts();
        }
        return () => {
            // видаляєм графіки якщо користувач переходить з головного
екрану
          if (incomeChartInstance.current) {
            incomeChartInstance.current.destroy();
          }
          if (expenseChartInstance.current) {
            expenseChartInstance.current.destroy();
          }
        };
```

```
        }, [loading, activeTab, income, expenses]);

        const loadData = async () => {
          try {
            setLoading(true);
                const [userData, incomeData, expenseData, incomeCatData,
expenseCatData] = await Promise.all([
              api.user.get(),
              api.income.get(),
              api.expense.get(),
              api.incomeCategory.get(),
              api.expenseCategory.get()
            ]);
            setUser(userData);
            setIncome(incomeData);
            setExpenses(expenseData);
            setIncomeCategories(incomeCatData);
            setExpenseCategories(expenseCatData);
          } catch (error) {
            console.error('Error loading data:', error);
          } finally {
            setLoading(false);
          }
        };
        const openModal = (type: any, item: any = null) => {
          setModalType(type);
          setEditingItem(item);
          setFormData(item || {});
          setShowModal(true);
        };
        const closeModal = () => {
          setShowModal(false);
          setModalType('');
          setEditingItem(null);
          setFormData({});
        };
        const handleSubmit = async (e: any) => {
```

```
        if (e) e.preventDefault();
        try {
          if (editingItem) {
               await api[modalType].update({ ...editingItem, ...formData
});
          } else {
            await api[modalType].post(formData);
          }
          await loadData();
          closeModal();
        } catch (error) {
          console.error('Помилка зберігання:', error);
        }
      };
      const handleDelete = async (type: string, id: any) => {
        if (window.confirm('Ви впевненні, що хочете видалити ці дані?'))
{
          try {
            await api[type].delete(id);
            await loadData();
          } catch (error) {
            console.error('Помилка видалення:', error);
          }
        }
      };
       const totalIncome = income.reduce((sum: number, item: { amount:
number }) => sum + item.amount, 0);
        const totalExpenses = expenses.reduce((sum: number, item: {
amount: number }) => sum + item.amount, 0);
      const balance = totalIncome - totalExpenses;
      const renderModal = () => {
        if (!showModal) return null;

        const isCategory = modalType.includes('Category');
        const isIncome = modalType === 'income';
        const isExpense = modalType === 'expense';
```

```
          const categories = isIncome ? incomeCategories : isExpense ? expenseCategories : [];

         return (
              <div className="fixed inset-0 bg-black bg-opacity-50 flex items-center justify-center z-50">
             <div className="bg-white rounded-lg p-6 w-full max-w-md">
               <h3 className="text-lg font-semibold mb-4">
                 {editingItem ? 'Змінити' : 'Додати'} {[
                   { key: 'income', label: 'Доходи', icon: TrendingUp },
                   { key: 'expense', label: 'Витрати', icon: TrendingDown },
                   { key: 'user', label: 'Користувач', icon: TrendingDown },
                    { key: 'incomeCategory', label: 'Категорії доходів', icon: Tag },
                    { key: 'expenseCategory', label: 'Категорії витрат', icon: Tag }
                 ].filter(e => e.key === modalType)[0].label}
               </h3>
               <div onSubmit={handleSubmit}>
                 {isCategory ? (
                   <>
                     <div className="mb-4">
                             <label className="block text-sm font-medium mb-2">Назва</label>
                       <input
                         type="text"
                         value={formData.categoryName || ''}
                              onChange={(e) => setFormData({ ...formData, categoryName: e.target.value })}
                         className="w-full p-2 border rounded-md"
                         required
                       />
                     </div>
                     <div className="mb-4">
```

```
                            <label className="block text-sm font-medium
mb-2">Опис</label>
                        <textarea
                          value={formData.description || ''}
                               onChange={(e) => setFormData({ ...formData,
description: e.target.value })}
                          className="w-full p-2 border rounded-md"
                          rows={3}
                        />
                      </div>
                    </>
                  ) : modalType === 'user' ? (
                    <>
                      <div className="mb-4">
                            <label className="block text-sm font-medium
mb-2">Назва</label>
                        <input
                          type="text"
                          value={formData.username || ''}
                               onChange={(e) => setFormData({ ...formData,
username: e.target.value })}
                          className="w-full p-2 border rounded-md"
                          required
                        />
                      </div>
                      <div className="mb-4">
                            <label className="block text-sm font-medium
mb-2">Email</label>
                        <input
                          type="email"
                          value={formData.email || ''}
                               onChange={(e) => setFormData({ ...formData,
email: e.target.value })}
                          className="w-full p-2 border rounded-md"
                          required
                        />
                      </div>
```

```
                    </>
                  ) : (
                    <>
                      <div className="mb-4">
                              <label  className="block  text-sm  font-medium
mb-2">Сума</label>
                        <input
                          type="number"
                          step="0.01"
                          value={formData.amount || ''}
                               onChange={(e)  =>  setFormData({  ...formData,
amount: parseFloat(e.target.value) })}
                          className="w-full p-2 border rounded-md"
                          required
                        />
                      </div>
                      <div className="mb-4">
                              <label  className="block  text-sm  font-medium
mb-2">Опис</label>
                        <input
                          type="text"
                          value={formData.description || ''}
                               onChange={(e)  =>  setFormData({  ...formData,
description: e.target.value })}
                          className="w-full p-2 border rounded-md"
                          required
                        />
                      </div>
                      <div className="mb-4">
                              <label  className="block  text-sm  font-medium
mb-2">Категорія</label>
                        <select
                          value={formData.categoryId || ''}
                               onChange={(e)  =>  setFormData({  ...formData,
categoryId: e.target.value })}
                          className="w-full p-2 border rounded-md"
                          required
```

```
                        >
                          <option value="">Оберіть категорію</option>
                                    {categories.map((cat: { _id: string,
categoryName: string }) => (
                                                          <option  key={cat._id}
value={cat._id}>{cat.categoryName || ''}</option>
                          ))}
                        </select>
                      </div>
                      <div className="mb-4">
                              <label  className="block  text-sm  font-medium
mb-2">Дата</label>
                        <input
                          type="date"
                          value={formData.date || ''}
                               onChange={(e)  =>  setFormData({  ...formData,
date: e.target.value })}
                          className="w-full p-2 border rounded-md"
                          required
                        />
                      </div>
                    </>
                  )}
                  <div className="flex gap-2 justify-end">
                    <button
                      type="button"
                      onClick={closeModal}
                      className="px-4 py-2 text-gray-600 border rounded-md
hover:bg-gray-50"
                    >
                      Відмінити
                    </button>
                    <button
                      type="button"
                      onClick={handleSubmit}
                              className="px-4  py-2  bg-blue-500  text-white
rounded-md hover:bg-blue-600"
```

```
                    >
                      {editingItem ? 'Оновити' : 'Створити'}
                    </button>
                  </div>
                </div>
              </div>
            </div>
          );
        };
        const renderDashboard = () => (
          <div className="space-y-6">
            <div className="grid grid-cols-1 md:grid-cols-3 gap-6">
              <div className="bg-green-50 p-6 rounded-lg">
                <div className="flex items-center justify-between">
                  <div>
                                    <p  className="text-green-600  text-sm
font-medium">Загальний дохід</p>
                                      <p  className="text-2xl  font-bold
text-green-700">${totalIncome.toFixed(2)}</p>
                  </div>
                  <TrendingUp className="h-8 w-8 text-green-500" />
                </div>
              </div>
              <div className="bg-red-50 p-6 rounded-lg">
                <div className="flex items-center justify-between">
                  <div>
                                    <p  className="text-red-600  text-sm
font-medium">Загальні витрати</p>
                                      <p  className="text-2xl  font-bold
text-red-700">${totalExpenses.toFixed(2)}</p>
                  </div>
                  <TrendingDown className="h-8 w-8 text-red-500" />
                </div>
              </div>
                    <div  className={`${balance  >=  0  ?  'bg-blue-50'  :
'bg-orange-50'} p-6 rounded-lg`}>
                <div className="flex items-center justify-between">
```

```
                <div>
                     <p className={`${balance >= 0 ? 'text-blue-600' :
'text-orange-600'} text-sm font-medium`}>Поточний баланс</p>
                    <p className={`text-2xl font-bold ${balance >= 0 ?
'text-blue-700' : 'text-orange-700'}`}>
                      ${balance.toFixed(2)}
                    </p>
                </div>
                     <DollarSign className={`h-8 w-8 ${balance >= 0 ?
'text-blue-500' : 'text-orange-500'}`} />
              </div>
            </div>
          </div>
               {/* Charts Section */}
          <div className="grid grid-cols-1 lg:grid-cols-2 gap-6">
            <div className="bg-white p-6 rounded-lg shadow">
                 <h3 className="text-lg font-semibold mb-4">Доходи по
категоріям</h3>
              <div className="h-64 flex items-center justify-center">
                {income.length > 0 ? (
                                      <canvas   ref={incomeChartRef}
className="max-h-full"></canvas>
                ) : (
                  <p className="text-gray-500">Даних немає</p>
                )}
              </div>
            </div>
            <div className="bg-white p-6 rounded-lg shadow">
                 <h3 className="text-lg font-semibold mb-4">Витрати по
категоріям</h3>
              <div className="h-64 flex items-center justify-center">
                {expenses.length > 0 ? (
                                     <canvas   ref={expenseChartRef}
className="max-h-full"></canvas>
                ) : (
                  <p className="text-gray-500">Даних немає</p>
                )}
```

```
            </div>
          </div>
        </div>
        <div className="grid grid-cols-1 lg:grid-cols-2 gap-6">
          <div className="bg-white p-6 rounded-lg shadow">
            <div className="flex items-center justify-between mb-4">
                        <h3  className="text-lg  font-semibold">Історія
доходу</h3>
              <button
                onClick={() => openModal('income')}
                     className="bg-green-500 text-white p-2 rounded-md
hover:bg-green-600"
              >
                <Plus className="h-4 w-4" />
              </button>
            </div>
            <div className="space-y-2">
                        {income.slice(0,  5).map((item:  {  id:  number,
description: string, category: string, date: string, amount: number }) =>
(
                  <div key={item.id} className="flex justify-between
items-center p-2 hover:bg-gray-50 rounded">
                    <div>
                      <p className="font-medium">{item.description}</p>
                                                <p   className="text-sm
text-gray-500">{item.category} • {item.date}</p>
                    </div>
                                            <p   className="font-semibold
text-green-600">${item.amount.toFixed(2)}</p>
                  </div>
                ))}
            </div>
          </div>
        </div>
        <div className="bg-white p-6 rounded-lg shadow">
          <div className="flex items-center justify-between mb-4">
                        <h3  className="text-lg  font-semibold">Історія
розходів</h3>
```

```
                <button
                  onClick={() => openModal('expense')}
                        className="bg-red-500 text-white p-2 rounded-md
hover:bg-red-600"
                >
                  <Plus className="h-4 w-4" />
                </button>
              </div>
              <div className="space-y-2">
                        {expenses.slice(0, 5).map((item: { id: number,
description: string, category: string, date: string, amount: number }) =>
(
                    <div key={item.id} className="flex justify-between
items-center p-2 hover:bg-gray-50 rounded">
                      <div>
                        <p className="font-medium">{item.description}</p>
                                                <p  className="text-sm
text-gray-500">{item.category} • {item.date}</p>
                      </div>
                                            <p  className="font-semibold
text-red-600">${item.amount.toFixed(2)}</p>
                    </div>
                  ))}
                </div>
              </div>
            </div>
          </div>
        );
        const renderDataTable = (data: any, type: string, categories = [])
=> (
          <div className="bg-white rounded-lg shadow overflow-hidden">
            <div className="p-6 border-b">
              <div className="flex items-center justify-between">
                <h3 className="text-lg font-semibold capitalize">{[
                    { key: 'dashboard', label: 'Головна', icon: DollarSign
},
                    { key: 'income', label: 'Доходи', icon: TrendingUp },
```

```
                { key: 'expense', label: 'Витрати', icon: TrendingDown
},
                  { key: 'incomeCategory', label: 'Категорії доходів',
icon: Tag },
                  { key: 'expenseCategory', label: 'Категорії витрат',
icon: Tag }
                ].filter(e => e.key === type)[0].label}</h3>
              <button
                onClick={() => openModal(type)}
                  className="bg-blue-500 text-white px-4 py-2 rounded-md
hover:bg-blue-600 flex items-center gap-2"
              >
                <Plus className="h-4 w-4" />
                Додати {[
                  { key: 'dashboard', label: 'Головна', icon: DollarSign
},
                  { key: 'income', label: 'Доходи', icon: TrendingUp },
                  { key: 'expense', label: 'Витрати', icon: TrendingDown
},
                  { key: 'incomeCategory', label: 'Категорії доходів',
icon: Tag },
                  { key: 'expenseCategory', label: 'Категорії витрат',
icon: Tag }
                ].filter(e => e.key === type)[0].label}
              </button>
            </div>
          </div>
          <div className="overflow-x-auto">
            <table className="w-full">
              <thead className="bg-gray-50">
                <tr>
                  {type.includes('Category') ? (
                    <>
                              <th className="px-6 py-3 text-left text-xs
font-medium text-gray-500 uppercase">Назва</th>
                              <th className="px-6 py-3 text-left text-xs
font-medium text-gray-500 uppercase">Опис</th>
```

```
                    </>
                  ) : type === 'user' ? (
                    <>
                              <th className="px-6 py-3 text-left text-xs
font-medium text-gray-500 uppercase">Назва</th>
                              <th className="px-6 py-3 text-left text-xs
font-medium text-gray-500 uppercase">Email</th>
                    </>
                  ) : (
                    <>
                              <th className="px-6 py-3 text-left text-xs
font-medium text-gray-500 uppercase">Опис</th>
                              <th className="px-6 py-3 text-left text-xs
font-medium text-gray-500 uppercase">Сума</th>
                              <th className="px-6 py-3 text-left text-xs
font-medium text-gray-500 uppercase">Категорія</th>
                              <th className="px-6 py-3 text-left text-xs
font-medium text-gray-500 uppercase">Дата</th>
                    </>
                  )}
                  <th className="px-6 py-3 text-left text-xs font-medium
text-gray-500 uppercase">Actions</th>
                </tr>
              </thead>
              <tbody className="divide-y divide-gray-200">
                   {data.map((item: { _id: string, description: string,
name: string, categoryName: string, email: string, category: string,
date: string, amount: number }) => (
                  <tr key={item._id} className="hover:bg-gray-50">
                    {type.includes('Category') ? (
                      <>
                                        <td className="px-6 py-4 text-sm
font-medium">{item.categoryName}</td>
                                        <td className="px-6 py-4 text-sm
text-gray-500">{item.description}</td>
                      </>
                    ) : type === 'user' ? (
```

```
                        <>
                                                <td  className="px-6  py-4  text-sm
font-medium">{item.name}</td>
                                                <td  className="px-6  py-4  text-sm
text-gray-500">{item.email}</td>
                        </>
                      ) : (
                        <>
                                                <td  className="px-6  py-4  text-sm
font-medium">{item.description}</td>
                          <td className="px-6 py-4 text-sm font-semibold">
                                    <span  className={type  ===  'income'  ?
'text-green-600' : 'text-red-600'}>
                              ${item.amount.toFixed(2)}
                            </span>
                          </td>
                                                <td  className="px-6  py-4  text-sm
text-gray-500">{item.category}</td>
                                                <td  className="px-6  py-4  text-sm
text-gray-500">{item.date}</td>
                        </>
                      )}
                      <td className="px-6 py-4 text-sm">
                        <div className="flex gap-2">
                          <button
                            onClick={() => openModal(type, item)}
                            className="text-blue-600 hover:text-blue-800"
                          >
                            <Edit2 className="h-4 w-4" />
                          </button>
                          <button
                            onClick={() => handleDelete(type, item._id)}
                            className="text-red-600 hover:text-red-800"
                          >
                            <Trash2 className="h-4 w-4" />
                          </button>
                        </div>
```

```
                    </td>
                  </tr>
                ))}
              </tbody>
            </table>
          </div>
        </div>
      );
      if (loading) {
        return (
          <div className="min-h-screen bg-gray-100 flex items-center justify-center">
            <div className="text-center">
              <div className="animate-spin rounded-full h-12 w-12 border-b-2 border-blue-500 mx-auto"></div>
              <p className="mt-4 text-gray-600">Завантаження...</p>
            </div>
          </div>
        );
      }
      return (
        <div className="min-h-screen bg-gray-100">
          <nav className="bg-white shadow">
            <div className="max-w-7xl mx-auto px-4">
              <div className="flex justify-between h-16">
                <div className="flex items-center">
                  <DollarSign className="h-8 w-8 text-blue-500" />
                  <span  className="ml-2  text-xl  font-bold text-gray-900">Budget Manager</span>
                </div>
                <div className="flex items-center space-x-4">
                  <span  className="text-gray-700">Доброго  дня, {user?.username}</span>
                  <button
                    onClick={() => openModal('user', user)}
                    className="text-gray-500 hover:text-gray-700"
                  >
```

```
                    <User className="h-6 w-6" />
                  </button>
                </div>
              </div>
            </div>
          </nav>
          <div className="max-w-7xl mx-auto px-4 py-8">
            <div classNamе="mb-8">
                        <div  classNamе="flex  space-x-1  bg-gray-200  p-1
rounded-lg">
                {[
                  { key: 'dashboard', label: 'Головна', icon: DollarSign
},
                  { key: 'income', label: 'Доходи', icon: TrendingUp },
                  { key: 'expense', label: 'Витрати', icon: TrendingDown
},
                    { key: 'incomeCategory', label: 'Категорії доходів',
icon: Tag },
                    { key: 'expenseCategory', label: 'Категорії витрат',
icon: Tag }
                ].map(tab => {
                  const Icon = tab.icon;
                  return (
                    <button
                      key={tab.key}
                      onClick={() => setActiveTab(tab.key)}
                          classNamе={`flex items-center gap-2 px-4 py-2
rounded-md font-medium transition-colors ${activeTab === tab.key
                        ? 'bg-white text-blue-600 shadow'
                        : 'text-gray-600 hover:text-gray-900'
                        }`}
                    >
                      <Icon classNamе="h-4 w-4" />
                      {tab.label}
                    </button>
                  );
                })}
```

```
              </div>
            </div>
            {activeTab === 'dashboard' && renderDashboard()}
            {activeTab === 'income' && renderDataTable(income, 'income',
incomeCategories)}
                {activeTab === 'expense' && renderDataTable(expenses,
'expense', expenseCategories)}
                                {activeTab === 'incomeCategory' &&
renderDataTable(incomeCategories, 'incomeCategory')}
                               {activeTab === 'expenseCategory' &&
renderDataTable(expenseCategories, 'expenseCategory')}
          </div>
          {renderModal()}
        </div>
      );
    };
    export default BudgetApp;
```