Міністерство освіти і науки України

Чернівецький національний університету імені Юрія Федьковича»

Відокремлений структурний підрозділ «Фаховий коледж Чернівецького національного університету імені Юрія Федьковича»

Природниче відділення

Циклова комісія комп'ютерної інженерії

КВАЛІФІКАЦІЙНА РОБОТА

на здобуття освітньо-професійного ступеня «фаховий молодший бакалавр»

зі спеціальності *123 Комп'ютерна інженерія*

підготовки за освітньо-професійною програмою «*Комп'ютерна інженерія*»

на тему: «**Трекер персонального бюджету»**

Виконав (ла):

студент (ка) 4-го курсу *Панчук Дмитро Вікторович*

(прізвище, ім'я та по батькові) (підпис)

Керівник:

Мельничук А.Ю. _____

(науковий ступінь, вчене звання, прізвище та ініціали) (підпис)

Рецензент:

к. ф.-м. н., Тащук О.Ю. _____

(науковий ступінь, вчене звання, прізвище та ініціали) (підпис)

До захисту допущено:

Протокол засідання циклової комісії № _

від „___"_____ 2024 р.

Голова циклової комісії _____ Олександр ТАЩУК

Чернівці, 2024

Завдання та календарний план роботи

Міністерство освіти і науки України
Чернівецький національний університету імені Юрія Федьковича»
Відокремлений структурний підрозділ «Фаховий коледж
Чернівецького національного університету імені Юрія Федьковича»

Затверджую
Голова циклової комісії,
Олександр ТАЩУК.

_____

«____» _____2023 р.

Завдання

На кваліфікаційну роботу	Дмитро Вікторович
Тема кваліфікаційної роботи	Трекер персонального бюджету
Постановка завдання в короткій формі	Розробка та програмування програмної частини трекера персонального бюджету.
Вихідні дані (2-3 адреси сайтів, матеріали яких рекомендує керівник кваліфікаційної роботи)	https://dribbble.com/tags/expense-tracker https://www.behance.net/search/projects/expense%20tracker%20-

Календарний план роботи

Дата отримання версії звіту керівником	Етап виконання роботи	Виконання (зазначає керівник)
03.10.23	Отримання завдання до кваліфікаційної роботи.	
04.11.23	Огляд літератури, присвяченої розробці ПЗ автономних мобільних роботів.	

05.12.23	Аналіз існуючих конструкторських рішень. Оформлення розділу 1.	
03.01.24	Вибір мови програмування, узгодження модулів системи.	
27.01.24	Написання теоретичного матеріалу по використовуваних модулях.	
10.03.24	Проектування програмної частини трекеру персонального бюджету. Оформлення розділу 2.	
10.04.24	Тестування та перевірка, працездатності ПЗ. Оформлення розділу 3.	
27.04.24	Оформлення та редагування кваліфікаційної роботи.	
за графіком	Подання роботи на перевірку Інтернет-сервісом Unicheck.	
за графіком	Подання роботи на рецензію	
за графіком	Представлення роботи на засіданні Циклової комісії	
за графіком	Захист кваліфікаційної роботи	

Дата видачі завдання _____.

Студент Дмитро ПАНЧУК

Керівник (П.І.Б) Христина МЕЛЬНИЧУК

Андрій МЕЛЬНИЧУК

АНОТАЦІЯ

У даній кваліфікаційній роботі проведено аналіз теоретичних основ, інструментів та технологій, що використовуються при створенні програмного забезпечення для трекеру особистого бюджету. Виконано аналіз вимог до проектування програмної частини, реалізацію та інтеграцію основних модулів системи, а також проведено тестування та налаштування трекера.

Актуальність роботи зумовлена тим, що контроль особистих фінансів є важливою складовою сучасного життя. У зв'язку з зростанням кількості фінансових операцій, стає важче відстежувати витрати та доходи без відповідного інструменту. Трекер бюджету допомагає організувати фінанси, контролювати витрати та планувати майбутні фінансові цілі, що дозволяє покращити фінансове благополуччя та уникнути нераціональних витрат.

Програмне забезпечення для трекеру бюджету має бути зручним у використанні, надійним та адаптивним до різних потреб користувачів.

Метою роботи є розробка програмної частини трекера персонального бюджету, яка включає в себе створення основних модулів для обліку доходів і витрат а також тестування і налагодження програмного продукту. Для зберігання даних використовується база даних MongoDB, а сам додаток є веб-застосунком, розгорнутим на платформі Vercel

Об'єктом дослідження є програмна частина трекера.

Робота складається з чотирьох розділів, у яких розглянуто теоретичні основи, інструменти та технології, що використовуються при розробці програмного забезпечення. Проведено аналіз вимог до проектування програмної частини, її реалізацію та інтеграцію з іншими сервісами, а також здійснено тестування та налагодження трекера.

Кваліфікаційна робота містить 83 сторінок, 18 рисунків та 2 посилань на літературні джерела.

Ключові слова: *MongoDB, Vercel, фінансовий менеджмент, апробація, фронтенд, бекенд, трекер бюджету, персональний бюджет.*

ЗМІСТ

СПИСОК ТЕРМІНІВ, СКОРОЧЕНЬ ТА ПОЗНАЧЕНЬ

MongoDB - документо-орієнтована система керування базами даних з відкритим вихідним кодом, яка не потребує опису схеми таблиць. MongoDB займає нішу між швидкими і масштабованими системами, що оперують даними у форматі ключ/значення, і реляційними СКБД, функціональними і зручними у формуванні запитів.

Vercel - це платформа, яка допомагає розробникам швидко та ефективно створювати та розгортати веб-додатки. Вона надає фреймворки, робочі процеси та інфраструктуру для створення швидших та більш персоналізованих веб-сайтів.

Фінансовий менеджмент - система принципів, методів, засобів і форм організації грошових відносин. Основна суть полягає в управлінні фінансами з метою підвищення ефективності виробництва та розподілу продукту.

Апробація - офіційне схвалення, затвердження чого-небудь після випробування, перевірки.

Фронтенд - відображення функціональних завдань призначеного для користувача інтерфейсу, що виконуються на стороні клієнта, а також обробка запитів користувачів.

Бекенд - відповідає за створення програмного забезпечення сайту або застосунків, що працюють на сервері. Крім того, він займається програмуванням баз даних, відповідальних за зберігання інформації на сервері та безпеку сайтів або програм.

Трекер бюджету - додаток для відстеження витрат і доходів, який допоможе вам взяти під контроль свій бюджет, гроші та фінанси без зайвих витрат часу.

Персональний бюджет - розрахунок доходів та витрат за певний проміжок часу.

ВСТУП

Актуальність теми. Сучасне життя вимагає все більш ретельного контролю над особистими фінансами. Зростання кількості фінансових операцій, цифрових сервісів і різних методів оплати ускладнює процес відстеження витрат та доходів. Ефективне управління бюджетом допомагає уникнути фінансових проблем, планувати майбутні витрати і заощадження, а також досягати фінансових цілей. Тому розробка трекерів персонального бюджету є актуальною темою.

Метою даної роботи є розробка та впровадження веб-застосунку для управління особистими фінансами, який дозволить користувачам зручно відстежувати доходи та витрати, планувати бюджет і контролювати фінансовий стан. В процесі роботи використовуватимуться сучасні технології, такі як React для розробки інтерфейсу користувача та MongoDB для зберігання даних, а сам додаток буде розгорнутий на платформі Vercel.

Об’єктом дослідження є програмна частина трекера персонального бюджету.

Предметом дослідження є процес розробки конструкції та програмного забезпечення для трекера персонального бюджету.

Мета роботи: розробка та впровадження веб-застосунку для управління особистими фінансами. Для досягнення цієї мети необхідно виконати наступні задачі:

-Аналіз існуючих систем для управління персональними фінансами та визначення їх переваг і недоліків.

-Визначення вимог до системи та проектування архітектури трекера бюджету.

-Розробка бази даних на MongoDB для зберігання фінансових даних.

-Розробка інтерфейсу користувача за допомогою React.

-Розгортання веб-застосунку на платформі Vercel.

-Тестування та налагодження системи для забезпечення її надійності та зручності у використанні.

Методи дослідження. Дослідження базується на методах аналізу та проектування програмного забезпечення, веб-розробки, роботи з базами даних, а також використанні засобів розробки для веб-додатків. Використовуються методи тестування та апробації програмного забезпечення для підтвердження його функціональності та ефективності.

Практична цінність отриманих результатів полягає у можливості створення ефективного та зручного веб-застосунку для управління особистими фінансами. Такий додаток може бути корисним для широкого кола користувачів, включаючи студентів, працівників, сім'ї, які бажають мати кращий контроль над своїм бюджетом, планувати витрати та заощадження, а також уникати фінансових труднощів.

1 ПОСТАНОВКА ЗАДАЧІ

Трекер персонального бюджету – це веб-додаток, що дозволяє користувачам ефективно керувати своїми фінансами за допомогою сучасних технологій. Сьогодні контроль власних фінансів є невід'ємною частиною повсякденного життя кожної людини. Вміння правильно планувати витрати та доходи, уникати непередбачених боргів і досягати фінансових цілей є надзвичайно важливим для забезпечення стабільного і впевненого майбутнього.

Розробка трекера має на меті створення зручного і ефективного інструменту, який дозволить користувачам вести облік своїх фінансів, аналізувати витрати, планувати бюджет і контролювати фінансовий стан.

Однією з ключових переваг трекера є його зручність. Користувачі можуть вводити дані безпосередньо на своєму пристрої, будь то комп'ютер чи смартфон. Це дозволяє зберігати всю інформацію в одному місці і мати до неї доступ у будь-який час. Трекер персонального бюджету може бути економічно вигідним рішенням, оскільки він дозволяє користувачам більш ефективно планувати свої витрати і заощадження. Це, у свою чергу, може призвести до зменшення непередбачених витрат і кращого розподілу фінансових ресурсів.

Загалом, розробка трекера персонального бюджету є актуальною задачею, яка може значно покращити фінансову дисципліну та стабільність користувачів. Він стане потужним інструментом для управління фінансами, що допоможе користувачам контролювати свої фінансові потоки, досягати фінансових цілей і уникати фінансових проблем. Враховуючи всі зазначені аспекти, розробка трекера є важливим кроком на шляху до забезпечення фінансового благополуччя користувачів.

1.1 Огляд існуючих систем для управління персональними фінансами

Розробка трекера персонального бюджету є актуальним завданням, і на ринку вже існує декілька популярних рішень, які допомагають користувачам ефективно управляти своїми фінансами.

Mint - Mint є одним із найпопулярніших трекерів бюджету. Він пропонує широкий набір функцій для відстеження витрат, створення бюджетів і отримання фінансових звітів.

Переваги: Інтуїтивно зрозумілий інтерфейс, автоматичне підключення до банківських рахунків, категоризація транзакцій, сповіщення про перевищення бюджету.

Недоліки: Обмежені можливості для налаштування, висока залежність від автоматизації, що може призводити до помилок у категоризації транзакцій.

YNAB (You Need a Budget) - YNAB пропонує унікальний підхід до бюджетування, фокусуючись на чотирьох основних правилах, що допомагають користувачам управляти своїми фінансами більш свідомо.

Переваги: Чітка методологія бюджетування, акцент на плануванні майбутніх витрат, зручний мобільний додаток.

Недоліки: Висока вартість підписки, необхідність ручного введення деяких даних, складний інтерфейс для новачків.

Personal Capital - Personal Capital надає інструменти для управління бюджетом і інвестиціями, що робить його привабливим для тих, хто хоче контролювати всі свої фінансові аспекти в одному місці. [4]

Переваги: Потужні інструменти для аналізу інвестицій, інтеграція з банківськими рахунками, детальні звіти про витрати та доходи.

Недоліки: Складність інтерфейсу, орієнтація більше на інвесторів, а не на звичайних користувачів.

Goodbudget - Goodbudget базується на методі конвертів, де користувачі розподіляють свої доходи по різним категоріям витрат.

Переваги: Проста концепція конвертів, можливість синхронізації з іншими користувачами (наприклад, сім'єю), легкість у використанні.

Недоліки: Відсутність автоматичного підключення до банківських рахунків, обмежені можливості для аналізу даних.

1.2 Аналіз технічного завдання

Технічне завдання (ТЗ) є основою будь-якого проєкту розробки програмного забезпечення, оскільки воно визначає вимоги до функціональності, архітектури, технологій і етапів реалізації.

Функціональні вимоги:

- Реєстрація та авторизація користувачів - можливість створювати облікові записи та входити в систему.
- Введення фінансових операцій - можливість додавати, редагувати та видаляти записи про доходи та витрати.
- Категоризація транзакцій: можливість відносити транзакції до різних категорій (наприклад, "Їжа", "Транспорт", "Розваги").

Нефункціональні вимоги:

- Додаток повинен швидко обробляти запити користувачів і забезпечувати високий рівень продуктивності.
- Інтерфейс користувача повинен бути інтуїтивно зрозумілим та зручним для використання на різних пристроях.

Технологічні вимоги:

- **Фронтенд:** React для створення інтерфейсу користувача. Ця бібліотека дозволяє створювати динамічні та інтерективні веб-додатки.

- **Бекенд:** Використання Node.js для серверної частини, яка забезпечує логіку обробки даних та взаємодію з базою даних.
- **База даних:** MongoDB для зберігання фінансових даних користувачів. MongoDB забезпечує високу продуктивність та гнучкість у роботі з неструктурованими даними.
- **Хостинг:** Платформа Vercel для розгортання та хостингу веб-додатку. Vercel забезпечує зручне середовище для безперервної інтеграції та розгортання додатків.

Вимоги до інтерфейсу користувача

- Інтерфейс повинен бути простим і зрозумілим, забезпечуючи легкий доступ до всіх основних функцій.
- Інтерфейс повинен коректно відображатися на різних пристроях та екранах різних розмірів.

Етапи реалізації

Розробка трекера персонального бюджету буде поділена на кілька основних етапів:

1. Визначення повного переліку вимог до системи та розробка технічного завдання.
2. Проектування архітектури додатку, створення макетів інтерфейсу користувача.
3. Написання коду для фронтенду та бекенду, інтеграція з базою даних.
4. Проведення модульного та інтеграційного тестування для виявлення та виправлення помилок.
5. Запуск додатку на платформі Vercel та налаштування середовища для його стабільної роботи.
6. Регулярне оновлення додатку, впровадження нових функцій та виправлення виявлених проблем.

2. ТЕХНОЛОГІЇ ТА ІНСТРУМЕНТИ ДЛЯ РЕАЛІАЦІЇ ПРОЄКТУ

У даному розділі ми визначимо архітектурні концепції, структуру програмного забезпечення та виберемо необхідні технології та інструменти для реалізації проекту.

2.1 Вибір технологій

Для розробки трекера персонального бюджету було прийнято рішення використовувати React. Це одна з найпопулярніших та найефективніших бібліотек для створення користувацьких інтерфейсів веб-додатків. Вона має компонентну архітектуру, яка дозволяє легко розділити інтерфейс на невеликі самостійні компоненти, що спрощує розробку та підтримку коду.

Banks та Porcello у книзі "Learning React: Modern Patterns for Developing React Apps" зазначають, що використання React значно полегшує процес розробки завдяки компонентному підходу та можливості швидкого оновлення інтерфейсу[2]

2.1.1 Вибір фронтенд технологій

У розділі про вибір фронтенд технологій, визначено чотири ключових інструменти, які допоможуть забезпечити ефективну розробку інтерфейсу додатка.

Для стилізації інтерфейсу, буде використовуватись Tailwind CSS. Його концепція "utility-first" дозволить швидко та зручно задавати стилі, використовуючи готові класи.

Formik став вибором для керування формами в додатку. Він надає зручний спосіб валідації, обробки подій та управління станом форм.

Для відображення анімаційних спінерів під час завантаження даних буде використовуватись бібліотека react-spinners. Вона дозволяє забезпечити користувачеві відчуття, що додаток працює, коли дані завантажуються.

Аксіос є інструментом взаємодії з сервером. Його простий інтерфейс дозволяє легко виконувати HTTP-запити та обробляти відповіді сервера.

Ці технології будуть використовуватись в даній роботі через їхню ефективність, зручність у використанні та популярність у розробницькій спільноті. Вони сприяють швидкій та зручній розробці функціонального фронтенду для майбутнього додатку.

2.1.2 Вибір бекенд технологій

Бекенд програми для керування особистим бюджетом забезпечує основну логіку та функціонал, необхідний для роботи з фінансовими даними користувачів. Він реалізований за допомогою фреймворку NestJS, який надає ефективні засоби для побудови серверної частини додатка.

- **NestJS** - Обрано фреймворк з метою створення розширюваних та надійних серверних додатків на Node.js. NestJS пропонує концепцію модульності, використання декораторів та інжекцію залежностей для зручного розробки та тестування.
- **MongoDB** - В якості бази даних використовується MongoDB, NoSQL система збереження даних, що пропонує гнучкість та швидкодію для зберігання фінансових даних користувачів. [5]
- **Passport.js** - Для аутентифікації користувачів використовується Passport.js, модульний пакет для аутентифікації на Node.js. Використовується локальна стратегія для перевірки логіна та пароля користувача.
- **Bcrypt** - Для безпечного зберігання та перевірки паролів використовується бібліотека bcrypt, яка забезпечує хешування паролів перед зберіганням у базі даних.
- Розділення логіки на контролери та сервіси для кращої організації та розподілу обов'язків.

2.1.3 Вибір бази даних

Після аналізу потреб додатку та порівняння різних варіантів, було вирішено використовувати MongoDB. Нижче подано докладний огляд причин цього вибору та переваг цієї бази даних.

Причини вибору MongoDB:

Гнучкість та масштабованість - це документ-орієнтована база даних, що дозволяє зберігати дані у вигляді JSON-подібних документів. Ця модель даних дозволяє зберігати дані гнучко та легко розширювати їхню структуру в майбутньому, що особливо корисно для додатків зі змінними вимогами.

Висока швидкодія - пропонує дуже ефективну швидкодію завдяки своїй архітектурі. Вона використовує індексацію для швидкого пошуку даних та розподілені обчислення для оптимізації операцій.

Горизонтальне масштабування - підтримує горизонтальне масштабування, що дозволяє розподілити навантаження на декілька серверів. Це дозволяє забезпечити високу доступність та стійкість до великого обсягу даних.

Підтримка складних операцій - надає потужні можливості для роботи з даними, включаючи агрегаційні операції, текстовий пошук, геопросторові запити та багато іншого. Це дозволяє ефективно виконувати різноманітні завдання обробки даних у додатку.

Переваги MongoDB:

MongoDB має дружній для розробника інтерфейс, що дозволяє швидко розгорнути та працювати з базою даних.

Можливість зберігання даних у вигляді документів JSON-подібних документів робить MongoDB ідеальним вибором для додатків зі змінними вимогами. [1]

Пропонує прості механізми горизонтального масштабування, що дозволяє додавати нові сервери та розподіляти навантаження.

Надає багато корисних функцій, таких як резервне копіювання, реплікація даних та керування доступом, що забезпечують безпеку та надійність даних.

2.1.4 Вибір платформи для запуску додатка

При виборі платформи для розгортання цього веб-додатку було обрано Vercel, і ось чому:

Vercel надає інтуїтивний та дружній для користувача інтерфейс, що дозволяє швидко розгорнути та керувати веб-додатком без зайвих складнощів.

Платформа автоматично налаштовує потрібні середовища для запуску веб-додатків, включаючи CDN, SSL-сертифікати та інші параметри.

Пропонує швидке розгортання веб-додатків за допомогою простих команд або інтеграції з репозиторіями Git. Це дозволяє швидко впроваджувати зміни та тестувати їх у продакшн-середовищі.

Платформа Vercel забезпечує гнучке масштабування веб-додатків, дозволяючи легко розширювати потужності за потребою. Це особливо важливо для додатків зі зростаючою або непередбачуваною кількістю користувачів.

Vercel має багато інтеграцій з популярними сервісами розробки, такими як GitHub, GitLab, Bitbucket та інші. Це дозволяє автоматизувати процеси розгортання та спростити роботу розробників. [7]

Переваги Vercel [7]:

- Платформа пропонує високу швидкість розгортання та відповіді веб-додатків, що забезпечує користувачам приємний досвід використання додатків.
- Vercel забезпечує високий рівень безпеки для розгорнутих додатків, включаючи захист від атак та регулярне оновлення безпеки.

- Надає інструменти для моніторингу та аналітики роботи веб-додатків, що дозволяє виявляти та виправляти проблеми швидко та ефективно.
- Добре документовані ресурси та активну спільноту користувачів, що дозволяє швидко отримати допомогу та вирішити будь-які питання.
- Різні плани тарифів, включаючи безкоштовний план для невеликих проектів, а також платні плани з розширеними можливостями для великих підприємств.

2.2. Архітектура системи

2.2.1. Загальна архітектура системи

У системі використовується мікросервісна архітектура, що полягає в розділенні функціональності на невеликі, незалежні сервіси. Кожен сервіс відповідає за свою конкретну функціональність, що дозволяє гнучко реагувати на зміни та легко масштабувати систему.

Використання JWT (JSON Web Tokens) у системі дозволяє забезпечити безпеку та автентифікацію користувачів. JWT - це стандарт відкритих токенів, які використовуються для передачі даних між сторонами в форматі JSON. Вони підписуються цифровим ключем і можуть бути перевірені для цілісності даних.

Паролі користувачів зберігаються у хешованому форматі за допомогою bcrypt. Bcrypt - це криптографічний хеш-функція, яка забезпечує надійний захист паролів шляхом генерації унікального хешу для кожного пароля.

MongoDB обрана як база даних для зберігання транзакцій. Її гнучка схема дозволяє ефективно зберігати дані про фінансові операції користувачів.

У системі встановлено різні ролі та права доступу, які дозволяють обмежити доступ до конфіденційної інформації. Це допомагає забезпечити конфіденційність та безпеку даних в системі.

Обрано платформу Vercel для запуску веб-додатку. Вона дозволяє швидко запускати та масштабувати додатки без складностей налаштування серверів.

2.2.2. Взаємодія компонентів фронтенду та бекенду

В додатку для керування особистим бюджетом фронтенд і бекенд тісно співпрацюють для забезпечення всіх функцій. Фронтенд, створений на базі React, взаємодіє з й на NestJS через HTTP-запити. Для обміну даними використовується бібліотека Axios.

Як це працює?

Спочатку фронтенд відправляє HTTP-запити до бекенду. Це можуть бути запити на реєстрацію, логін, отримання або збереження даних. Бекенд, отримавши ці запити, обробляє їх і повертає відповіді, які фронтенд використовує для оновлення інтерфейсу.

Аутентифікація користувачів

У додатку користувачі можуть реєструватися та авторизуватися. Коли користувач заповнює форму реєстрації або логіну на фронтенді, ці дані (email і пароль) відправляються на бекенд. Там NestJS перевіряє дані (наприклад, паролі хешуються за допомогою bcrypt) і створює нового користувача або перевіряє існуючого. Якщо логін успішний, бекенд генерує JWT-токен, який потім використовує фронтенд для авторизації подальших запитів.

Робота з даними користувача

Фронтенд може відправляти запити на отримання або оновлення даних користувача. Наприклад, коли користувач оновлює своє ім'я, фронтенд відправляє PUT-запит на бекенд з новими даними. Бекенд обробляє цей запит, оновлює інформацію в MongoDB і повертає оновлені дані на фронтенд.

Керування транзакціями

Дуже важливою частиною розробки додатку є керування фінансовими транзакціями. Користувач може додавати нові транзакції через форми на

фронтенді. Дані про транзакцію (сума, дата, тип, опис) відправляються на бекенд, де зберігаються в MongoDB. Фронтенд також може запитувати список транзакцій для відображення їх у інтерфейсі або видаляти транзакції за допомогою відповідних запитів на бекенд.

Обробка помилок

Щоб користувачі розуміли, що відбувається, коли виникають помилки, бекенд повертає відповіді з повідомленнями про помилки. Наприклад, якщо під час логіну введено неправильний пароль, бекенд відправить відповідне повідомлення, яке фронтенд відобразить користувачеві.

Запуск та масштабування

В даній роботі фронтенд запущений на Vercel, що дуже зручно, адже ця платформа автоматично запускає додаток, забезпечує безпечне з'єднання через HTTPS і дозволяє легко масштабувати додаток. Бекенд, завдяки NestJS, теж легко масштабувати, додаючи нові мікросервіси або розширюючи існуючі.

2.2.3. Структура бази даних

У MongoDB було створено декілька основних колекцій, але головна з них — це колекція користувачів. Ось як вона структурована:

Колекція користувачів (Users)

Кожен документ у колекції Users представляє собою одного користувача і має такі поля:

- email: Це унікальний ідентифікатор кожного користувача. Він зберігається у нижньому регістрі для забезпечення унікальності та зручності порівняння.
- password: Хеш пароля користувача, створений за допомогою бібліотеки bcrypt. Це забезпечує безпеку даних користувачів.
- name: Ім'я користувача, яке використовується для персоналізації інтерфейсу.

- transactions: Масив транзакцій, що належать користувачеві.

Кожна транзакція містить такі поля:

- _id: Унікальний ідентифікатор транзакції, який генерується автоматично.
- date: Дата транзакції.
- amount: Сума транзакції.
- type: Тип транзакції, який може бути "Витрата" або "Надходження".
- description: Опис транзакції, який користувач додає для кращого розуміння витрат або доходів.

Чому така структура?

Використовуючи MongoDB, можна легко змінювати структуру документів, додаючи нові поля або змінюючи існуючі без потреби у складних міграціях бази даних.

JSON-подібний формат документів робить роботу з даними інтуїтивно зрозумілою, особливо при розробці на JavaScript.

База даних MongoDB забезпечує високу швидкодію при операціях запису та читання, що важливо для додатку, де користувачі постійно взаємодіють з даними. БД надає потужні інструменти для агрегації, фільтрації та сортування даних, що дозволяє ефективно обробляти великі обсяги інформації.

3. ПРОЕКТУВАННЯ ПРОГРАМНОЇ ЧАСТИНИ

В даному розділі розглянемо проектування бази даних, інтерфейсу користувача, забезпечення безпеки, а також стратегії тестування та налагодження для забезпечення якості та надійності програмного продукту.

3.1 Проектування інтерфейсу користувача

3.1.1 Реалізація інтерфейсу користувача на React

Для створення інтерфейсу користувача в додатку для керування персональним бюджетом було обрано React.

Основною перевагою React є його компонентний підхід. В процесі роботи було створено інтерфейс на невеликі, повторно використовувані компоненти, що спрощує розробку та підтримку коду. Наприклад, я створив окремі компоненти для форм реєстрації та логіну, списку транзакцій, форми додавання нових транзакцій, і т.д.

На рис. 3.1 показано приклад компоненту для кнопки, яка використовується в різних частинах додатку.

```
function Button({
  children,
  type = "button",
  className = "",
  onClick,
  isLoading = false,
  isDisabled = false,
}) {
  return (
    <button
      type={type}
      disabled={isDisabled}
      onClick={onClick}
      className={`bg-formItemsBackground rounded-xl xs:text-sm border-0 disabled:cursor-not-allowed outline-none
        text-textColor px-3 py-1 h-10 text-xl cursor-pointer hover:bg-[#150C16] ${className}`}>
      {isLoading ? "Обробка..." : children}
    </button>
  );
}

export default Button;
```

Рис. 3.1. Компонент для кнопки

На рис. 3.2 видно, як в роботі використовується Tailwind CSS для стилізації компонентів.

```
<div className='w-full h-60 xs:h-36 md:h-44 bg-[#704E71] rounded-xl flex'>
  <div className='h-4/6 w-10/12 m-auto flex flex-col'>
    <div className='h-3/6 w-full flex justify-between xs:mb-3'>
      <div className='flex flex-col '>
        <Label className='!text-xl xs:!text-base md:!text-base'>
          Баланс
        </Label>
        <Amount className='!text-4xl xs:!text-xl md:!text-2xl'>
          {formatNumber(balance)}
        </Amount>
      </div>
    </div>
  </div>
</div>
```

Рис. 3.2. Використання Tailwind CSS для стилізації компонентів

Управління станом компонентів є важливою частиною будь-якого додатку. У додатку я використовую хуки React, такі як useState та useEffect, для керування станом компонентів. Це дозволяє зберігати та оновлювати дані у реальному часі.

На рис. 3.3. наведено приклад використання хука useState для зберігання стану транзакцій.

```
import React, { useState, useEffect } from "react";

function TransactionsList({ transactions }) {
  const [sortedTransactions, setSortedTransactions] = useState([]);

  useEffect(() => {
    const sorted = [...transactions].sort(
      (a, b) => new Date(b.date) - new Date(a.date)
    );
    setSortedTransactions(sorted);
  }, [transactions]);

  return (
    <div>
      {sortedTransactions.map((transaction) => (
        <TransactionItem key={transaction.id} transaction={transaction} />
      ))}
    </div>
  );
}

export default TransactionsList;
```

Рис. 3.3. Використання useState для зберігання стану транзакцій.

Взаємодія з бекендом за допомогою Axios

Для взаємодії з бекендом використовується бібліотека Axios. Вона дозволяє легко відправляти HTTP-запити на сервер і отримувати відповіді. Це особливо важливо для операцій, таких як реєстрація, логін, отримання та збереження транзакцій.

На рис. 3.4 наведено приклад використання Axios для реєстрації нового користувача.

```
import axios from "axios";

export const apiClient = axios.create({
  baseURL: import.meta.env.VITE_API_URL,
  responseType: "json",
});

async function registerUser(email, password, name) {
  try {
    const response = await apiClient.post("/auth/signup", {
      email,
      password,
      name,
    });
    return response.data;
  } catch (error) {
    console.error("Error registering user:", error);
    throw error;
  }
}
```

Рис. 3.4. Використання Axios для реєстрації нового користувача

3.1.2. Адаптивність та кросплатформеність

У сучасному світі дуже важливо, щоб веб-додатки були доступними на різних пристроях, таких як смартфони, планшети, ноутбуки та настільні комп'ютери. Тому, при розробці додатку для керування особистим бюджетом, особливу увагу було приділено його адаптивності та кросплатформеності.

Використання Tailwind CSS

Для забезпечення адаптивності додатку використовується Tailwind CSS. Tailwind надає велику кількість утилітарних класів, які дозволяють легко створювати адаптивні дизайни. За допомогою медіа-запитів можна змінювати стилі елементів в залежності від розміру екрана. Це дозволяє забезпечити зручний вигляд додатку на будь-якому пристрої.

Кросплатформеність

React, як бібліотека для побудови інтерфейсів, забезпечує кросплатформенність завдяки своїй універсальності. Можемо створювати компоненти, які працюють однаково добре на різних платформах, включаючи веб-додатки, мобільні додатки (за допомогою React Native) та навіть десктопні додатки (за допомогою Electron).

3.2. Реалізація бекенд частини

3.2.1. Створення REST API на Node.js

Створення REST API на Node.js є популярним підходом для розробки серверних додатків завдяки його асинхронній природі, легкості використання та потужним можливостям. У даному проекті для керування особистим бюджетом створено REST API з використанням фреймворку NestJS, який побудований на базі Node.js. [3] Ось деякі деталі про те, як це було реалізовано.

Чому REST API?

REST (Representational State Transfer) API є стандартом для побудови веб-сервісів, що забезпечують взаємодію між клієнтами та серверами через HTTP-запити. Основні причини вибору REST API включають його простоту, легкість у використанні та сумісність з різними клієнтами (веб, мобільні додатки, десктопні додатки).

Використання Node.js

Node.js був обраний для розробки REST API через його асинхронну, неблокуючу модель вводу/виводу, яка дозволяє ефективно обробляти велику кількість одночасних запитів. Крім того, Node.js має велику кількість бібліотек і фреймворків, що полегшують розробку, таких як Express.js і NestJS.[3]

Бекенд програми для керування бюджетом забезпечує основну логіку та функціонал, необхідний для роботи з фінансовими даними користувачів. Він реалізований за допомогою фреймворку NestJS, який надає ефективні засоби для побудови серверної частини додатка.

Casciaro та Mammino у своїй книзі "Node.js Design Patterns" підкреслюють важливість використання ефективних патернів проектування для створення масштабованих і надійних бекенд систем на базі Node.js[2]

Використання NestJS

NestJS - це фреймворк для Node.js, який надає потужні можливості для створення масштабованих та легко підтримуваних серверних додатків.

NestJS дозволяє організувати код у вигляді модулів, що сприяє розділенню відповідальностей та покращує підтримуваність коду.

Використання декораторів спрощує визначення маршрутів, контролерів і сервісів, а інжекція залежностей полегшує управління залежностями між компонентами.

Легко інтегрується з різними базами даних (наприклад, MongoDB) та іншими бібліотеками для забезпечення аутентифікації, авторизації та обробки запитів.

Основні компоненти REST API

У цьому проекті REST API складається з наступних основних компонентів:

1. Контролери - відповідають за обробку HTTP-запитів і відправку відповідей клієнтам. Наприклад, AuthController обробляє запити на реєстрацію та логін користувачів (див. рис. 3.5).

```
import { Body, Controller, Post, HttpException, HttpStatus } from '@nestjs/common';
import { AuthService } from './auth.service';

@Controller('auth')
export class AuthController {
  constructor(private readonly authService: AuthService) {}

  @Post('/login')
  async login(@Body('email') userEmail: string, @Body('password') userPassword: string) {
    try {
      const user = await this.authService.validateUser(userEmail, userPassword);
      return { msg: 'User successfully logged in', id: user.id, email: user.email, name: user.name };
    } catch (error) {
      throw new HttpException('Invalid login or password', HttpStatus.BAD_REQUEST);
    }
  }
}
```

Рис. 3.5. AuthController

2. Сервіси - містять основну бізнес-логіку додатка. Наприклад, AuthService містить методи для аутентифікації користувачів та управління їх даними (див. рис. 3.6).

```
import { Injectable, NotAcceptableException } from '@nestjs/common';
import { UsersService } from 'src/users/users.service';
import * as bcrypt from 'bcrypt';

@Injectable()
export class AuthService {
  constructor(private readonly usersService: UsersService) {}

  async validateUser(email: string, password: string): Promise<any> {
    const user = await this.usersService.getUser(email);
    if (!user) {
      throw new NotAcceptableException('User not found');
    }
    const passwordValid = await bcrypt.compare(password, user.password);
    if (user && passwordValid) {
      return { id: user.id, email: user.email, name: user.name };
    }
    return null;
  }
}
```

Рис. 3.6. AuthService

3.Моделі - визначають структуру даних у базі даних. Наприклад, модель User містить поля для email, пароля, ім'я користувача та транзакцій (дивю рис. 3.7).

```
import * as mongoose from 'mongoose';

export enum TransactionType {
  EXPENSE = 'Expense',
  INCOME = 'Income',
}

export const UserSchema = new mongoose.Schema({
  email: { type: String, required: true, unique: true },
  password: { type: String, required: true },
  name: { type: String, required: true },
  transactions: [{
    _id: { type: mongoose.Schema.Types.ObjectId, default: mongoose.Types.ObjectId },
    date: { type: Date, required: true },
    amount: { type: Number, required: true },
    type: { type: String, enum: Object.values(TransactionType), required: true },
    description: { type: String, required: true },
  }],
}, { timestamps: true });
```

Рис. 3.7. Модель User

4.NestJS дозволяє легко налаштовувати роутинг та використовувати middleware для обробки запитів, логування, перевірки авторизації тощо.

3.2.2. Обробка запитів та відповідей

Обробка запитів та відповідей є важливою частиною функціональності бекенду. Використовуючи фреймворк NestJS, реалізовано ефективну систему для обробки HTTP-запитів та повернення відповідей клієнту.

Як це працює?

Запити від клієнта

Клієнт (фронтенд) надсилає HTTP-запити до сервера (бекенду) для виконання різних операцій, таких як реєстрація користувача, логін, додавання транзакцій, отримання інформації про користувача тощо. Запити можуть бути різних типів: GET, POST, PUT, DELETE, кожен з яких відповідає певній дії.

Обробка запитів у контролерах

Контролери в NestJS відповідають за обробку запитів. Вони визначають маршрути та методи, які викликаються при надходженні запитів. Контролери отримують дані із запиту, викликають відповідні сервіси для обробки бізнес-логіки і повертають відповідь клієнту.

Приклад контролера для обробки запитів наведений на рис. 3.8.

```
import { Body, Controller, Post, HttpException, HttpStatus } from '@nestjs/common';
import { AuthService } from './auth.service';

@Controller('auth')
export class AuthController {
  constructor(private readonly authService: AuthService) {}

  @Post('/login')
  async login(@Body('email') userEmail: string, @Body('password') userPassword: string) {
    try {
      const user = await this.authService.validateUser(userEmail, userPassword);
      return { msg: 'User successfully logged in', id: user.id, email: user.email, name: user.name };
    } catch (error) {
      throw new HttpException('Invalid login or password', HttpStatus.BAD_REQUEST);
    }
```

Рис. 3.8. Контролер для обробки запитів

У цьому прикладі метод login обробляє POST-запит до маршруту auth/login. Він отримує дані з тіла запиту (@Body('email') userEmail) і передає їх у сервіс AuthService для перевірки. У разі успішного логіну повертається відповідь з інформацією про користувача, у разі помилки - відповідь з повідомленням про помилку.

Сервіси для обробки бізнес-логіки

Сервіси містять основну бізнес-логіку додатка. Вони виконують операції з базою даних, перевірки та інші необхідні дії. Контролери викликають методи сервісів для обробки запитів.

Приклад сервісу для аутентифікації наведений на рис. 3.9.

```
import { Injectable, NotAcceptableException } from '@nestjs/common';
import { UsersService } from 'src/users/users.service';
import * as bcrypt from 'bcrypt';

@Injectable()
export class AuthService {
  constructor(private readonly usersService: UsersService) {}

  async validateUser(email: string, password: string): Promise<any> {
    const user = await this.usersService.getUser(email);
    if (!user) {
      throw new NotAcceptableException('User not found');
    }
    const passwordValid = await bcrypt.compare(password, user.password);
    if (user && passwordValid) {
      return { id: user.id, email: user.email, name: user.name };
    }
    return null;
  }
}
```

Рис. 3.9. Сервіс для аутентифікації

У цьому прикладі сервіс AuthService містить метод validateUser, який перевіряє існування користувача та валідність пароля. Якщо користувача знайдено і пароль вірний, метод повертає інформацію про користувача.

Обробка відповідей

Після обробки запиту контролер повертає відповідь клієнту. Відповідь може містити дані, підтвердження успішного виконання операції або повідомлення про помилку. У NestJS для цього використовуються об'єкти HttpException для повернення HTTP-статусів та повідомлень про помилки.

Приклад обробки відповідей наведений на рис. 3.10.

```
@Post('/signup')
@HttpCode(HttpStatus.CREATED)
async addUser(
  @Body('password') userPassword: string,
  @Body('email') userEmail: string,
  @Body('name') userName: string,
) {
  try {
    const saltOrRounds = 10;
    const hashedPassword = await bcrypt.hash(userPassword, saltOrRounds);
    const result = await this.usersService.insertUser(
      userEmail,
      hashedPassword,
      userName,
    );
    return {
      msg: 'User successfully registered',
      userId: result.id,
      email: result.email,
      name: result.name,
    };
  } catch (error) {
    if (error.status === 409) {
      throw new HttpException(
        'User with this email already exists',
        HttpStatus.CONFLICT,
      );
    } else {
      throw new HttpException(
        'Internal Server Error',
        HttpStatus.INTERNAL_SERVER_ERROR,
      );
    }
  }
}
```

Рис. 3.10. Реалізація обробки відповідей

У цьому прикладі метод addUser обробляє POST-запит на реєстрацію користувача. У разі успішної реєстрації повертається відповідь з

повідомленням про успіх та інформацією про нового користувача. У разі помилки повертається відповідний статус HTTP та повідомлення про помилку.

Використання middleware та інтерсепторів

NestJS також дозволяє використовувати middleware та інтерсептори для обробки запитів і відповідей. Middleware може використовуватися для логіну, перевірки авторизації та інших дій перед обробкою запиту контролером. Інтерсептори можуть змінювати або доповнювати відповіді перед їх відправкою клієнту.

3.2.3. Взаємодія з базою даних MongoDB

Взаємодія з MongoDB здійснюється через Mongoose, об'єктно-документний меппер (ODM) для Node.js, що надає зручний спосіб роботи з документами в MongoDB, використовуючи JavaScript-об'єкти.[6]

Використання Mongoose

Mongoose дозволяє визначати схеми даних, створювати моделі та виконувати CRUD (Create, Read, Update, Delete) операції з документами в MongoDB. У моєму додатку я використовую Mongoose для взаємодії з колекцією користувачів та їх транзакцій.[6]

Створення моделей

Модель користувача визначається за допомогою схеми Mongoose. Схема визначає структуру документів, які зберігаються в колекції.

Приклад схеми користувача наведений на рис. 3.11.

```
import * as mongoose from 'mongoose';

export enum TransactionType {
  EXPENSE = 'Витрата',
  INCOME = 'Надходження',
}

export const UserSchema = new mongoose.Schema({
  email: { type: String, required: true, unique: true },
  password: { type: String, required: true },
  name: { type: String, required: true },
  transactions: [{
    _id: { type: mongoose.Schema.Types.ObjectId, default: mongoose.Types.ObjectId },
    date: { type: Date, required: true },
    amount: { type: Number, required: true },
    type: { type: String, enum: Object.values(TransactionType), required: true },
    description: { type: String, required: true },
  }],
}, { timestamps: true });

export interface User extends mongoose.Document {
  _id: string;
  email: string;
  password: string;
  name: string;
  transactions: Transaction[];
}
```

Рис. 3.11. Схема користувача

Виконання CRUD операцій

Моделі Mongoose дозволяють легко виконувати CRUD(Create, Read, Update, Delete) операції з документами. Наприклад, у сервісі UsersService я використовую методи Mongoose для створення, читання, оновлення та видалення документів користувачів і транзакцій.

Приклад сервісу для роботи з користувачами:

```
import { Injectable } from '@nestjs/common';
import { InjectModel } from '@nestjs/mongoose';
import { Model } from 'mongoose';
import { User, Transaction } from './users.model';
import mongoose from 'mongoose';

@Injectable()
export class UsersService {
  constructor(@InjectModel('user')    private    readonly    userModel:
Model<User>) {}

  async insertUser(userEmail: string, password: string, name: string) {
    const email = userEmail.toLowerCase();

    const existingUser = await this.userModel.findOne({ email });
    if (existingUser) {
      throw new Error('User with this email already exists');
    }

    const newUser = new this.userModel({ email, password, name });
    await newUser.save();
    return newUser;
  }

  async getUser(userEmail: string) {
    const email = userEmail.toLowerCase();
    return await this.userModel.findOne({ email });
  }

  async getUserById(userId: string) {
    return await this.userModel.findById(userId);
  }

  async updateUserName(userId: string, newName: string) {
    return await this.userModel.findByIdAndUpdate(userId, { name: newName
}, { new: true });
  }

  async addTransaction(userId: string, newTransaction: Transaction) {
    const transactionId = new mongoose.Types.ObjectId();
    const result = await this.userModel.findByIdAndUpdate(
      userId,
      { $push: { transactions: { _id: transactionId, ...newTransaction }
} },
      { new: true },
    );
    if (!result) {
      throw new Error('User not found');
    }
    return { ...newTransaction, id: transactionId };
  }
```

```
  async deleteTransaction(userId: string, transactionId: string) {
    return await this.userModel.findByIdAndUpdate(
      userId,
      { $pull: { transactions: { _id: transactionId } } },
      { new: true },
    );
  }
}
```

3.3. Інтеграція фронтенду з бекендом

3.3.1. Використання Axios для HTTP-запитів

Чому Axios?

Axios надає простий і зрозумілий синтаксис для виконання запитів, що дозволяє легко надсилати запити і обробляти відповіді.

Використовує проміси (Promises), що дозволяє зручно працювати з асинхронними операціями та спрощує обробку результатів запитів.

Автоматично перетворює дані в формат JSON при надсиланні запитів та обробці відповідей, що спрощує роботу з даними.

Дозволяє налаштовувати інтерсептори, встановлювати заголовки та інші параметри запитів, що дає більше контролю над HTTP-запитами.

Налаштування клієнта Axios

Створено окремий файл для налаштування клієнта Axios (див. рис. 3.12). Це дозволяє легко змінювати базові налаштування та використовувати один клієнт для всіх запитів у додатку.

```
    import axios from "axios";

export const apiClient = axios.create({
  baseURL: import.meta.env.VITE_API_URL, // Базовий URL API
  responseType: "json", // Тип відповіді
});
```

Рис. 3.12. Налаштування клієнта Axios

Виконання запитів

За допомогою налаштованого клієнта Axios можна виконувати різні запити, такі як реєстрація користувача, логін, отримання даних про транзакції тощо. Ось приклади деяких з них:

Реєстрація користувача ()

```
async function registerUser(email, password, name) {
  try {
    const response = await apiClient.post("/auth/signup", {
      email,
      password,
      name,
    });
    return response.data;
  } catch (error) {
    console.error("Error registering user:", error);
    throw error;
  }
}
```

Рис. 3.13. Реєстрація користувача

У цьому прикладі використовується метод post для надсилання даних реєстрації користувача на бекенд. У разі успіху повертаються дані користувача, у разі помилки виводиться повідомлення в консоль (див. рис. 3.13).

Отримання транзакцій користувача

```
async function getUserTransactions(userId) {
  try {
    const response = await apiClient.get(`/users/${userId}/transactions`);
    return response.data;
  } catch (error) {
    console.error("Error fetching transactions:", error);
    throw error;
  }
}
```

Рис. 3.14. Отримання транзакцій користувача

У цьому прикладі використовується метод get для отримання списку транзакцій користувача. Запит надсилається з параметром userId, який ідентифікує користувача (див. рис. 3.14).

Обробка помилок

Axios дозволяє легко обробляти помилки, що виникають під час виконання запитів. Використовуючи блок try-catch, можна обробити помилки і відповідним чином реагувати на них.

```
async function loginUser(email, password) {
  try {
    const response = await apiClient.post("/auth/login", {
      email,
      password,
    });
    return response.data;
  } catch (error) {
    if (error.response && error.response.status === 401) {
      console.error("Unauthorized: Invalid login credentials");
    } else {
      console.error("Error logging in:", error);
    }
    throw error;
  }
}
```

Рис. 3.15. Обробка помилок

У цьому прикладі, якщо сервер повертає статус 401 (Unauthorized), виводиться відповідне повідомлення про помилку (див. рис. 3.15).

3.3.2. Обробка даних на фронтенді

Вона включає кілька ключових етапів: отримання даних від сервера, маніпулювання цими даними, відображення їх у інтерфейсі та забезпечення інтерактивності для користувача.

Отримання даних від сервера

Перший крок в обробці даних — це їх отримання з сервера. Коли користувач заходить на сторінку, наприклад, зі списком транзакцій, додаток надсилає запит до сервера, отримує дані і зберігає їх у стані компонента за допомогою хуків React.

Маніпулювання даними

Після отримання даних з сервера часто необхідно їх обробити перед відображенням. Це може включати фільтрацію, сортування, обчислення підсумкових значень тощо. Наприклад, у додатку я обчислюю загальний баланс користувача, суму доходів та витрат. Для цього використовуються вбудовані методи масивів JavaScript, такі як map, filter і reduce.

Відображення даних

Відображення даних у інтерфейсі є кінцевим етапом обробки даних. Використовуючи JSX, можна створювати динамічні ітерації елементів, які відображають дані користувачів. Наприклад, список транзакцій може бути відображений у вигляді таблиці або карток, кожна з яких містить деталі про конкретну транзакцію. React автоматично оновлює ці елементи при зміні стану, забезпечуючи динамічний і реактивний інтерфейс.

Валідація

Валідація введення даних користувача є важливою частиною обробки даних на фронтенді. Для цього в роботі використовується бібліотека Formik, яка спрощує створення та управління формами, а також їх валідацію. Це дозволяє переконатися, що дані, введені користувачем, відповідають необхідним критеріям перед їх відправкою на сервер.

Взаємодія з сервером

Після валідації дані відправляються на сервер для збереження або обробки. Це може бути додавання нової транзакції, оновлення даних користувача або видалення існуючої транзакції. Використовуючи Axios, надсилаються відповідні HTTP-запити (POST, PUT, DELETE) і обробляються відповіді сервера, оновлюючи стан додатку відповідно до отриманих даних.

Обробка помилок

При роботі з даними завжди є ймовірність виникнення помилок. Це включає відображення повідомлень про помилки, коли запити до сервера не вдаються, або коли введені дані не відповідають вимогам. Це допомагає користувачам розуміти, що саме пішло не так і як це виправити.

3.3.3. Відображення отриманих даних

Відображення отриманих даних включає представлення їх у зручному та зрозумілому форматі для користувачів, забезпечення динамічних оновлень інтерфейсу та створення інтерактивного досвіду.

Динамічне відображення даних

Однією з основних переваг React є його здатність динамічно оновлювати інтерфейс при зміні стану. Це дозволяє відображати отримані дані у реальному часі, без необхідності перезавантаження сторінки. Наприклад, коли користувач додає нову транзакцію, список транзакцій автоматично оновлюється, відображаючи новий запис.

Компонентний підхід

React використовує компонентний підхід, що дозволяє розбити інтерфейс на невеликі, повторно використовувані компоненти. Це спрощує розробку та підтримку коду. Кожен компонент відповідає за відображення певної частини даних. Наприклад, можна створити окремі компоненти для відображення списку транзакцій, загального балансу, форми додавання нової транзакції тощо (див. рис. 3.16).

```
import React from 'react';

function TransactionItem({ transaction }) {
  return (
    <div className="transaction-item">
      <p>{transaction.description}</p>
      <p>{transaction.amount}</p>
      <p>{transaction.date}</p>
      <p>{transaction.type}</p>
    </div>
  );
}

function TransactionsList({ transactions }) {
  return (
    <div className="transactions-list">
      {transactions.map(transaction => (
        <TransactionItem key={transaction.id} transaction={transaction} />
      ))}
    </div>
  );
}

export default TransactionsList;
```

Рис. 3.16. Приклад компонента для відображення транзакцій

У цьому прикладі компонент TransactionsList отримує список транзакцій як пропс і відображає кожну транзакцію, використовуючи компонент TransactionItem.

Використання JSX

JSX (JavaScript XML) дозволяє писати HTML-подібний код у JavaScript, що робить створення та відображення елементів інтерфейсу зручнішим та інтуїтивно зрозумілим. З допомогою JSX можна легко створювати структури даних і відображати їх у вигляді таблиць, списків, карток тощо.

Форматування даних

Отримані дані часто потребують форматування перед відображенням. Наприклад, дати можуть бути відформатовані у зручний для користувача вигляд, а числа - округлені або доповнені символами валюти. Для цього використовуються різні бібліотеки та вбудовані методи JavaScript (див. рис. 3.17).

```
import React from 'react';

function formatCurrency(amount) {
  return new Intl.NumberFormat('uk-UA', {
    style: 'currency',
    currency: 'UAH'
  }).format(amount);
}

function TransactionItem({ transaction }) {
  return (
    <div className="transaction-item">
      <p>{transaction.description}</p>
      <p>{formatCurrency(transaction.amount)}</p>
      <p>{new Date(transaction.date).toLocaleDateString()}</p>
      <p>{transaction.type}</p>
    </div>
  );
}

export default TransactionItem;
```

Рис. 3.17. Приклад форматування даних

У цьому прикладі функція formatCurrency використовується для форматування суми транзакції у валютний формат, а метод toLocaleDateString для форматування дати.

3.4 Запуск додатку

3.4.1 Запуск додатку на платформі Vercel

Чому Vercel?

Vercel надає зручний інтерфейс та інструменти, які дозволяють легко розгортати додаток всього за кілька кроків.

Кожен раз, коли є потреба робити зміни у своєму репозиторії, Vercel автоматично розгортає нову версію додатку, що забезпечує безперервну інтеграцію та доставку (CI/CD).

Він використовує глобальну мережу CDN (Content Delivery Network), що забезпечує швидке завантаження додатку для користувачів з будь-якої точки світу.

Платформа автоматично масштабує ресурси залежно від навантаження, забезпечуючи стабільну роботу додатку навіть при високих пікових навантаженнях.

Надає автоматичне налаштування HTTPS, що забезпечує безпечне з'єднання між користувачами та сервером.

Процес запуску додатку

Перед розгортанням потрібно переконатися, що проект готовий до публікації. Це включає перевірку залежностей, тестування та побудову продакшн-версії додатку.

Спершу потрібно підключити свій репозиторій на GitHub до Vercel. Це дозволяє автоматично розгортати додаток при кожному пуші в основну гілку. Потім, створити новий проект на Vercel, вибрати репозиторій, який хочемо розгорнути, та налаштувати основні параметри, такі як середовище (production), змінні оточення та налаштування збірки.

Vercel автоматично виконує побудову проекту, використовуючи визначені налаштування збірки. Після успішної побудови додаток автоматично розгортається і стає доступним за наданою URL-адресою.

Після розгортання необхідно перевірити роботу додатку, щоб переконатися, що все працює правильно. Vercel також надає можливість переглядати логи та діагностувати можливі проблеми.

Переваги використання Vercel у поточному проекті

Завдяки автоматичній побудові та розгортанню, є змога швидко вносити зміни у додаток та публікувати нові версії.

Глобальна мережа CDN забезпечує швидке завантаження додатку для користувачів з різних регіонів.

Автоматичне налаштування HTTPS забезпечує захищене з'єднання, що є важливим для додатків, які обробляють особисті дані користувачів.

Vercel автоматично масштабує ресурси, що забезпечує стабільну роботу додатку навіть при високих навантаженнях.

3.4.2. Запуск проекту та виправлення помилок

Під час тестування було виявлено кілька помилок, які потребували виправлення. Деякі з них стосувалися неправильного відображення даних, а інші були пов'язані з некоректною обробкою подій.

Окрім цього, було помічено, що текст у мобільній версії додатку відображався не коректно. Виправлення цієї проблеми стало одним з найбільш пріорітетніших.

Оперативно виправивши всі знайдені помилки, були внесені необхідні зміни в код та повторно протестовано додаток. Для покращення відображення тексту у мобільній версії адаптовано стилі та налаштовано відповідні розміри шрифтів і елементів інтерфейсу.

Після виправлення помилок додаток почав працювати стабільно та коректно. Тепер він правильно відображає всі дані, коректно обробляє події користувача та виконує всі заплановані функції. Текст у мобільній версії тепер також відображається належним чином, що значно покращує користувацький досвід.

4. Демонстрація інтерфейсу та функціональних можливостей програми

Екран реєстрації

Після запуску програми користувач бачить екран реєстрації, який вітає його та пропонує створити новий акаунт або увійти у вже існуючий . На цьому екрані розміщено форму для введення необхідних даних для реєстрації (див. рис. 4.1).

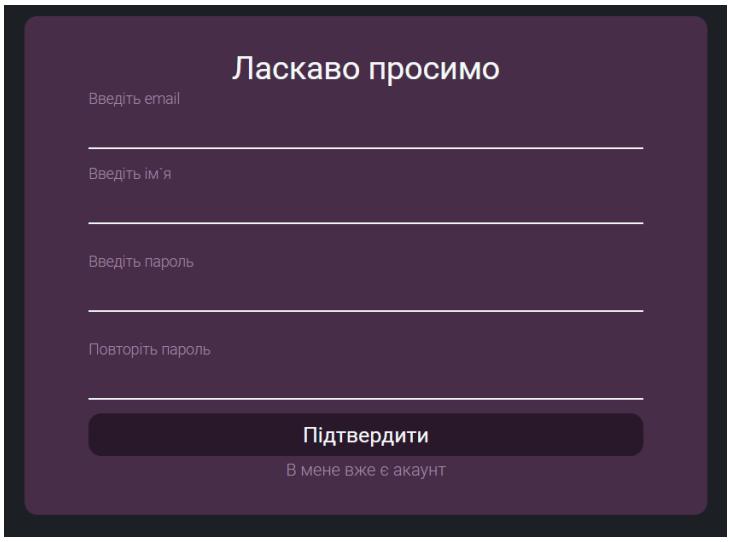

Рис. 4.1. Екран реєстрації програми

Екран входу

Після вибору опції "в мене вже є акаунт " в програмі, користувач бачить екран входу, який вітає його з поверненням і пропонує авторизуватися вже існуючим акаунтом. На цьому екрані розміщено форму для введення email та паролю, необхідних для входу (див. рис. 4.2).

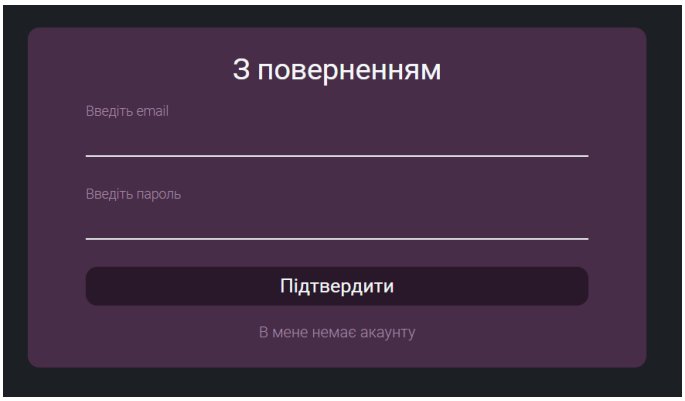

Рис. 4.2. Екран авторизації

Головний екран

Після успішної реєстрації/входу користувач потрапляє на головний екран програми, який вітає його персональним привітанням. Нижче розташований інформаційний блок з балансом коштів та окремими показниками доходів та витрат, що дозволяє користувачу швидко оцінити своє фінансове становище. Основна частина екрана включає форму для введення даних про нові фінансові операції, де можна вказати опис, суму та дату транзакції (див. рис. 4.3). Кнопки вибору дозволяють легко переключатися між доходами та витратами, а також забезпечують зручність підтвердження або відміни введеної інформації.

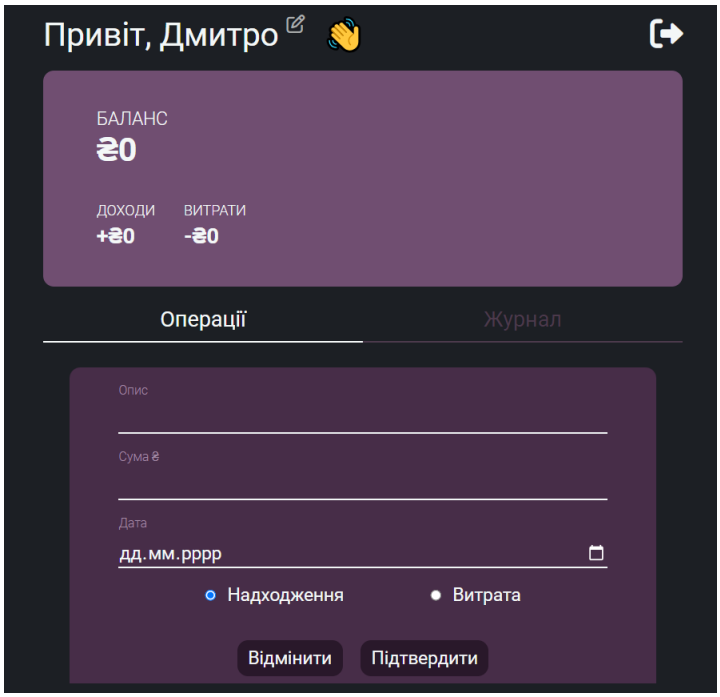

Рис. 4.3 Головний екран

Журнал операцій

На вкладці "Журнал" відображається деталізований список усіх фінансових операцій користувача з хронологічним впорядкуванням. Кожна операція супроводжується датою та описом, як, наприклад, "Зарплата" із зазначенням суми доходу або витрати (див. рис. 4.4). Це дозволяє користувачам легко переглядати свої фінансові транзакції за певний період і

сприяє кращому управлінню їхнім бюджетом. Значок кошика біля кожної операції дозволяє швидко видалити запис, якщо це необхідно

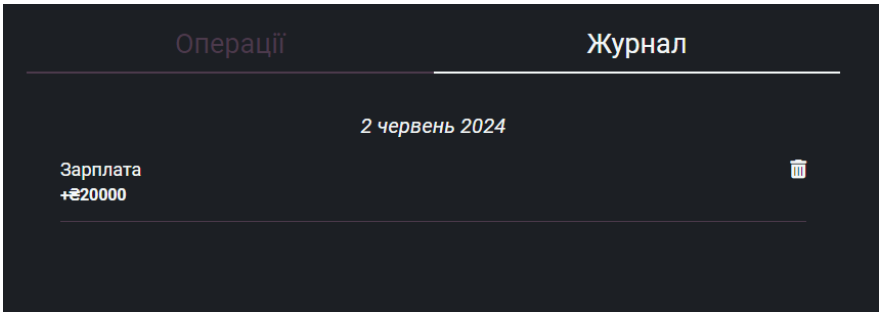

Рис. 4.4 Журнал операцій

ВИСНОВКИ

У рамках цієї кваліфікаційної роботи було розроблено додаток для керування персональним бюджетом, який дозволяє користувачам ефективно відслідковувати свої фінанси, додавати транзакції та аналізувати витрати і доходи. Основною метою роботи було створення зручного, функціонального та надійного інструменту для управління особистими фінансами.

Процес розробки включав кілька ключових етапів, починаючи з аналізу існуючих систем та формування технічного завдання. Було обрано найбільш кращі технології для реалізації проекту, серед яких React для фронтенду, Node.js та NestJS для бекенду, MongoDB для зберігання даних та Vercel для розгортання додатку. Такий вибір технологій забезпечив високу продуктивність, масштабованість та надійність системи.

Розробка цього додатку дозволила отримати цінний досвід у використанні сучасних технологій для створення веб-додатків, а також навички у тестуванні та налагодженні програмного забезпечення. Завдяки цьому проекту було продемонстровано, як можна ефективно вирішити проблему управління особистими фінансами за допомогою зручного та надійного інструменту.

Результати роботи можуть бути корисними як для подальшого розвитку додатку, так і для інших проектів, пов'язаних з управлінням даними та фінансами.

СПИСКИ ВИКОРИСТАНИХ ДЖЕРЕЛ

1. "MongoDB: The Definitive Guide" by Kristina Chodorow and Michael Dirolf. URL: https://pepa.holla.cz/wp-content/uploads/2016/11/MongoDB-The-Definitive-Guide.pdf (дата звернення: 14.01.24)
2. "Learning React: Modern Patterns for Developing React Apps" by Alex Banks and Eve Porcello, 2020. 307 с.
3. "Node.js Design Patterns" by Mario Casciaro and Luciano Mammino. URL: https://github.com/kpyszkowski/kpyszkowski/blob/main/books/Mario%20Casciaro%20%26%20Luciano%20Mammino%20-%20Node.js%20Design%20Patterns%3A%20Design%20and%20implement%20production-grade%20Node.js%20applications%20using%20proven%20patterns%20and%20techniques%20-%20Packt%20Publishing%20(2020).pdf (дата звернення: 15.02.24)
4. "Financial Management: Theory & Practice" by Eugene F. Brigham and Michael C. Ehrhardt
5. MongoDB. URL: https://www.mongodb.com/ (дата звернення: 20.02.24)
6. W3Schools. URL: https://www.w3schools.com/ (дата звернення: 25.02.24)
7. Vercel. URl: https://vercel.com/ (дата звернення: 23.03.24)

ДОДАТКИ

Додаток А. Лістинг програми

Бекенд

Код контролера авторизації (AuthController) у бекенді (NestJS)

```
import {
  Body,
  Controller,
  HttpException,
  HttpStatus,
  Post,
} from '@nestjs/common';
import { AuthService } from './auth.service';

@Controller('auth')
export class AuthController {
  constructor(private readonly authService: AuthService) {}

  @Post('/login')
  async login(
    @Body('password') userPassword: string,
    @Body('email') userEmail: string,
  ) {
    try {
      const user = await this.authService.validateUser(userEmail,
userPassword);
```

					Код контролера авторизації	Літ.	Маса	Масштаб
Змн.	Арк.	№ докум.	Підпис	Дата				
Розроб.		Панчук Д.В.						
Перевір.		Мельничук А.Ю.				Арк.	Аркушів 1	
Т. Контр.								
Реценз.						Циклова комісія Комп'ютерної інженерії		
Н. Контр.								
Затверд.		Тащук О.Ю.						

```
        return {
          msg: 'User successfully login',
          id: user.id,
          email: user.email,
          name: user.name,
        };
      } catch (error) {
        throw new HttpException(
          'Неправильний логін або пароль',
          HttpStatus.BAD_REQUEST,
        );
      }
    }
  }
```

Змн.	Арк.	№ докум.	Підпис	Дата	**Код контролера авторизації**	Літ.	Маса	Масштаб
Розроб.		Панчук Д.В.						
Перевір.		Мельничук А.Ю.				Арк.	Аркушів	1
Т. Контр.								
Реценз.						Циклова комісія Комп'ютерної інженерії		
Н. Контр.								
Затверд.		Тащук О.Ю.						

Код модуля авторизації (AuthModule) у бекенді (NestJS)

```
import { Module } from '@nestjs/common';
import { PassportModule } from '@nestjs/passport';
import { UsersModule } from 'src/q/users.module';
import { AuthController } from './auth.controller';
import { AuthService } from './auth.service';
import { LocalStrategy } from './local.strategy';
import { SessionSerializer } from './session.serializer';

@Module({
  imports: [UsersModule, PassportModule.register({ session: true })],
  providers: [AuthService, LocalStrategy, SessionSerializer],
  controllers: [AuthController],
})
export class AuthModule {}
```

Змн.	Арк.	№ докум.	Підпис	Дата	**Код модуля авторизації**	Літ.	Маса	Масштаб
Розроб.		Панчук Д.В.						
Перевір.		Мельничук А.Ю.						
Т. Контр.						Арк.	Аркушів	1
Реценз.								
Н. Контр.								
Затверд.		Тащук О.Ю.						

Циклова комісія
Комп'ютерної інженерії

Тестування AuthService у бекенді (NestJS)

```
import { Test, TestingModule } from '@nestjs/testing';
import { AuthService } from './auth.service';

describe('AuthService', () => {
  let service: AuthService;

  beforeEach(async () => {
    const module: TestingModule = await Test.createTestingModule({
      providers: [AuthService],
    }).compile();

    service = module.get<AuthService>(AuthService);
  });

  it('should be defined', () => {
    expect(service).toBeDefined();
  });
});
```

					Тестування AuthService у бекенді	Літ.	Маса	Масштаб
Змн.	Арк.	№ докум.	Підпис	Дата				
Розроб.		Панчук Д.В.						
Перевір.		Мельничук А.Ю.				Арк.	Аркушів	1
Т. Контр.								
Реценз.						Циклова комісія Комп'ютерної інженерії		
Н. Контр.								
Затверд.		Тащук О.Ю.						

Код сервісу авторизації (AuthService) у бекенді (NestJS)

```
import { Injectable, NotAcceptableException } from '@nestjs/common';
import { UsersService } from 'src/users/users.service';
import * as bcrypt from 'bcrypt';

@Injectable()
export class AuthService {
  constructor(private readonly usersService: UsersService) {}

  async validateUser(email: string, password: string): Promise<any> {
    const user = await this.usersService.getUser(email);

    const passwordValid = await bcrypt.compare(password, user.password);

    if (!user) {
      throw new NotAcceptableException('could not find the user');
    }
    if (user && passwordValid) {
      return {
        id: user.id,
        email: user.email,
        name: user.name,
      };
    }
    return null;
  }
}
```

Змн.	Арк.	№ докум.	Підпис	Дата	Код сервісу авторизації	Літ.	Маса	Масштаб
Розроб.		Панчук Д.В.						
Перевір.		Мельничук А.Ю.						
Т. Контр.						Арк.	Аркушів	1
Реценз.						Циклова комісія Комп'ютерної інженерії		
Н. Контр.								
Затверд.		Тащук О.Ю.						

Код захисника авторизації (AuthenticatedGuard) у бекенді (NestJS)

```
import { CanActivate, ExecutionContext, Injectable } from '@nestjs/common';

@Injectable()
export class AuthenticatedGuard implements CanActivate {
  async canActivate(context: ExecutionContext) {
    const request = context.switchToHttp().getRequest();
    return request.isAuthenticated();
  }
}
```

Змн.	Арк.	№ докум.	Підпис	Дата	**Код захисника авторизації**	Літ.	Маса	Масштаб
Розроб.		Панчук Д.В.						
Перевір.		Мельничук А.Ю.						
Т. Контр.						Арк.	Аркушів	1
Реценз.								
Н. Контр.						Циклова комісія Комп'ютерної інженерії		
Затверд.		Тащук О.Ю.						

Код локального захисника авторизації (LocalAuthGuard) у бекенді (NestJS)

```
import { ExecutionContext, Injectable } from '@nestjs/common';
import { AuthGuard } from '@nestjs/passport';

@Injectable()
export class LocalAuthGuard extends AuthGuard('local') {
  async canActivate(context: ExecutionContext) {
    const result = (await super.canActivate(context)) as boolean;
    const request = context.switchToHttp().getRequest();
    await super.logIn(request);
    return result;
  }
}
```

					Код локального захисника авторизації	*Літ.*	*Маса*	*Масштаб*
Змн.	*Арк.*	*№ докум.*	*Підпис*	*Дата*				
Розроб.		*Панчук Д.В.*						
Перевір.		*Мельничук А.Ю.*						
Т. Контр.						*Арк.*	*Аркушів* 1	
Реценз.						*Циклова комісія Комп'ютерної інженерії*		
Н. Контр.								
Затверд.		*Тащук О.Ю.*						

Код стратегії локальної авторизації (LocalStrategy) у бекенді (NestJS)

```
import { Injectable, UnauthorizedException } from '@nestjs/common';
import { PassportStrategy } from '@nestjs/passport';
import { Strategy } from 'passport-local';
import { AuthService } from './auth.service';

@Injectable()
export class LocalStrategy extends PassportStrategy(Strategy) {
  constructor(private readonly authService: AuthService) {
    super();
  }

  async validate(email: string, password: string): Promise<any> {
    const userEmail = email.toLowerCase();
    const user = await this.authService.validateUser(userEmail,
password);
    if (!user) {
      throw new UnauthorizedException();
    }
    return user;
  }
}
```

Змн.	Арк.	№ докум.	Підпис	Дата	**Код стратегії локальної авторизації**	Літ.	Маса	Масштаб
Розроб.		Панчук Д.В.						
Перевір.		Мельничук А.Ю.						
Т. Контр.						Арк.	Аркушів	1
Реценз.						Циклова комісія Комп'ютерної інженерії		
Н. Контр.								
Затверд.		Тащук О.Ю.						

Код серіалізатора

сесій (SessionSerializer) у бекенді (NestJS)

```
import { Injectable } from '@nestjs/common';
import { PassportSerializer } from '@nestjs/passport';

@Injectable()
export class SessionSerializer extends PassportSerializer {
  serializeUser(user: any, done: (err: Error, user: any) => void): any {
    done(null, user);
  }

  deserializeUser(
    payload: any,
    done: (err: Error, payload: string) => void,
  ): any {
    done(null, payload);
  }
}
```

					Код серіалізатора сесій	Літ.	Маса	Масштаб
Змн.	Арк.	№ докум.	Підпис	Дата				
Розроб.		Панчук Д.В.						
Перевір.		Мельничук А.Ю.				Арк.	Аркушів 1	
Т. Контр.								
Реценз.						Циклова комісія Комп'ютерної інженерії		
Н. Контр.								
Затверд.		Тащук О.Ю.						

Тестування UsersController у бекенді (NestJS)

```
import { Test, TestingModule } from '@nestjs/testing';
import { UsersController } from './users.controller';

describe('UsersController', () => {
  let controller: UsersController;

  beforeEach(async () => {
    const module: TestingModule = await Test.createTestingModule({
      controllers: [UsersController],
    }).compile();

    controller = module.get<UsersController>(UsersController);
  });

  it('should be defined', () => {
    expect(controller).toBeDefined();
  });
});
```

Змн.	Арк.	№ докум.	Підпис	Дата	Тестування UsersController	Літ.	Маса	Масштаб
Розроб.		Панчук Д.В.						
Перевір.		Мельничук А.Ю.				Арк.	Аркушів 1	
Т. Контр.						Циклова комісія Комп'ютерної інженерії		
Реценз.								
Н. Контр.								
Затверд.		Тащук О.Ю.						

Код контролера користувачів (UsersController) у бекенді (NestJS)

```
import {
  Body,
  Controller,
  Get,
  Post,
  Request,
  HttpCode,
  HttpStatus,
  HttpException,
  Param,
  Put,
  Delete,
} from '@nestjs.common';
import * as bcrypt from 'bcrypt';
import { UsersService } from './users.service';
import { TransactionType } from './users.model';

@Controller('users')
export class UsersController {
  constructor(private readonly usersService: UsersService) {}

  // Signup
  @Post('/signup')
  @HttpCode(HttpStatus.CREATED)
  async addUser(
    @Body('password') userPassword: string,
    @Body('email') userEmail: string,
    @Body('name') userName: string,
  ) {
    try {
      const saltOrRounds = 10;
      const hashedPassword = await bcrypt.hash(userPassword,
saltOrRounds);
```

Змн.	Арк.	№ докум.	Підпис	Дата	**Код контролера користувачів**	Літ.	Маса	Масштаб
Розроб.		Панчук Д.В.						
Перевір.		Мельничук А.Ю.						
Т. Контр.						Арк.	Аркушів	1
Реценз.								
Н. Контр.								
Затверд.		Тащук О.Ю.				Циклова комісія Комп'ютерної інженерії		

```
      const result = await this.usersService.insertUser(
        userEmail,
        hashedPassword,
        userName,
      );
      return {
        msg: 'User successfully registered',
        userId: result.id,
        email: result.email,
        name: result.name,
      };
    } catch (error) {
      if (error.status === 409) {
        throw new HttpException(
          'Користувач з таким email вже існує',
          HttpStatus.CONFLICT,
        );
      } else {
        throw new HttpException(
          'Internal Server Error',
          HttpStatus.INTERNAL_SERVER_ERROR,
        );
      }
    }
  }

  @Get('/:id')
  @HttpCode(HttpStatus.OK)
  async getUserById(@Param('id') userId: string) {
    try {
      const user = await this.usersService.getUserById(userId);
      return user;
    } catch (error) {
```

Змн.	Арк.	№ докум.	Підпис	Дата	Код контролера користувачів	Літ.	Маса	Масштаб
Розроб.		Панчук Д.В.						
Перевір.		Мельничук А.Ю.				Арк.	Аркушів	1
Т. Контр.						Циклова комісія Комп'ютерної інженерії		
Реценз.								
Н. Контр.								
Затверд.		Тащук О.Ю.						

```
            throw new HttpException('Щось пішло не так',
HttpStatus.BAD_REQUEST);
          }
        }

        @Put('/update-user-name')
        @HttpCode(HttpStatus.OK)
        async updateUserName(
          @Body('id') userId: string,
          @Body('name') userName: string,
        ) {
          try {
            const updateUser = await this.usersService.updateUserName(
              userId,
              userName,
            );

            return updateUser;
          } catch (error) {
            throw new HttpException('Щось пішло не так',
HttpStatus.BAD_REQUEST);
          }
        }

        @Post('/transaction')
        @HttpCode(HttpStatus.OK)
        async addTransaction(
          @Body('userId') userId: string,
          @Body('amount') amount: number,
          @Body('date') date: Date,
          @Body('type') type: TransactionType,
          @Body('description') description: string,
        ) {
          try {
```

Змн.	Арк.	№ докум.	Підпис	Дата	**Код контролера користувачів**	Літ.	Маса	Масштаб
Розроб.		Панчук Д.В.						
Перевір.		Мельничук А.Ю.				Арк.	Аркушів	1
Т. Контр.						Циклова комісія Комп'ютерної інженерії		
Реценз.								
Н. Контр.								
Затверд.		Тащук О.Ю.						

```
            const message = await this.usersService.addTransaction(userId, {
              amount: +amount,
              date: date,
              type: type,
              description: description,
            });
            return message;
          } catch (error) {
            throw new HttpException('Щось пішло не так',
HttpStatus.BAD_REQUEST);
          }
        }

        @Delete('/:userId/transaction/:transactionId')
        @HttpCode(HttpStatus.OK)
        async deleteTransaction(
          @Param('userId') userId: string,
          @Param('transactionId') transactionId: string,
        ) {
          try {
            const message = await this.usersService.deleteTransaction(
              userId,
              transactionId,
            );
            return message;
          } catch (error) {
            throw new HttpException('Щось пішло не так',
HttpStatus.BAD_REQUEST);
          }
        }

        // Logout
        @Get('/logout')
        logout(@Request() req): any {
```

Змн.	Арк.	№ докум.	Підпис	Дата	Код контролера користувачів	Літ.	Маса	Масштаб
Розроб.		Панчук Д.В.						
Перевір.		Мельничук А.Ю.				Арк.	Аркушів	1
Т. Контр.								
Реценз.						Циклова комісія Комп'ютерної інженерії		
Н. Контр.								
Затверд.		Тащук О.Ю.						

```
      req.session.destroy();
      return { msg: 'The user session has ended' };
    }
  }
```

Змн.	Арк.	№ докум.	Підпис	Дата	**Код контролера користувачів**	Літ.	Маса	Масштаб
Розроб.		Панчук Д.В.						
Перевір.		Мельничук А.Ю.				Арк.	Аркушів	1
Т. Контр.						Циклова комісія Комп'ютерної інженерії		
Реценз.								
Н. Контр.								
Затверд.		Тащук О.Ю.						

Схема користувачів та транзакцій (UserSchema) у бекенді (NestJS)

```
import * as mongoose from 'mongoose';

export enum TransactionType {
  EXPENSE = 'Витрата',
  INCOME = 'Надходження',
}

export interface Transaction {
  _id?: string;
  date: Date;
  amount: number;
  type: TransactionType;
  description: string;
}

export const UserSchema = new mongoose.Schema(
  {
    email: {
      type: String,
      required: true,
      unique: true,
    },
    password: {
      type: String,
      required: true,
    },
    name: {
      type: String,
      required: true,
    },
    transactions: [
```

					Схема користувачів та транзакцій (	Літ.	Маса	Масштаб
Змн.	Арк.	№ докум.	Підпис	Дата				
Розроб.		Панчук Д.В.						
Перевір.		Мельничук А.Ю.				Арк.	Аркушів 1	
Т. Контр.						Циклова комісія Комп'ютерної інженерії		
Реценз.								
Н. Контр.								
Затверд.		Тащук О.Ю.						

```
        {
          _id: {
            type: mongoose.Schema.Types.ObjectId,
            default: mongoose.Types.ObjectId,
          },
          date: { type: Date, required: true },
          amount: { type: Number, required: true },
          description: { type: String, required: true },
          type: {
            type: String,
            enum: Object.values(TransactionType),
            required: true,
          },
        },
      ],
    },
    { timestamps: true },
);

export interface User extends mongoose.Document {
  _id: string;
  email: string;
  password: string;
  name: string;
  transactions: Transaction[];
}
```

Змн.	Арк.	№ докум.	Підпис	Дата	**Схема користувачів та транзакцій (**	Літ.	Маса	Масштаб
Розроб.		Панчук Д.В.						
Перевір.		Мельничук А.Ю.						
Т. Контр.						Арк.	Аркушів	1
Реценз.						Циклова комісія Комп'ютерної інженерії		
Н. Контр.								
Затверд.		Тащук О.Ю.						

Код модуля користувачів (UsersModule) у бекенді (NestJS)

```
import { Module } from '@nestjs.common';
import { MongooseModule } from '@nestjs/mongoose';
import { AuthModule } from 'src/auth/auth.module';
import { AuthService } from 'src/auth/auth.service';
import { UsersController } from './users.controller';
import { UserSchema } from './users.model';
import { UsersService } from './users.service';

@Module({
  imports: [MongooseModule.forFeature([{ name: 'user', schema: UserSchema
}])],
  controllers: [UsersController],
  providers: [UsersService],
  exports: [UsersService],
})
export class UsersModule {}
```

Змн.	*Арк.*	*№ докум.*	*Підпис*	*Дата*	**Код модуля користувачів**	*Літ.*	*Маса*	*Масштаб*
Розроб.		*Панчук Д.В.*						
Перевір.		*Мельничук А.Ю.*						
Т. Контр.						*Арк.*	*Аркушів*	1
Реценз.								
Н. Контр.						*Циклова комісія Комп'ютерної інженерії*		
Затверд.		*Тащук О.Ю.*						

Тестування UsersService у бекенді

```
import { Test, TestingModule } from '@nestjs/testing';
import { UsersService } from './users.service';

describe('UsersService', () => {
  let service: UsersService;

  beforeEach(async () => {
    const module: TestingModule = await Test.createTestingModule({
      providers: [UsersService],
    }).compile();

    service = module.get<UsersService>(UsersService);
  });

  it('should be defined', () => {
    expect(service).toBeDefined();
  });
});
```

					Тестування UsersService	Літ.	Маса	Масштаб
Змн.	Арк.	№ докум.	Підпис	Дата				
Розроб.		Панчук Д.В.						
Перевір.		Мельничук А.Ю.						
Т. Контр.						Арк.	Аркушів	1
Реценз.								
Н. Контр.						Циклова комісія Комп'ютерної інженерії		
Затверд.		Тащук О.Ю.						

Код сервісу користувачів (UsersService) у бекенді (NestJS)

```
import { Injectable } from '@nestjs/common';
import { InjectModel } from '@nestjs/mongoose';
import { Model } from 'mongoose';
import { User } from './users.model';
import { Transaction } from './users.model';
import mongoose from 'mongoose';

class ConflictError extends Error {
  status: number;

  constructor(message: string) {
    super(message);
    this.status = 409;
  }
}

@Injectable()
export class UsersService {
  constructor(@InjectModel('user')    private    readonly    userModel:
Model<User>) {}

  async insertUser(userEmail: string, password: string, name: string) {
    const email = userEmail.toLowerCase();

    const existingUser = await this.userModel.findOne({ email });
    if (existingUser) {
      throw new ConflictError('Користувач з таким email вже існує');
    }

    try {
      const newUser = new this.userModel({
        email,
        password,
```

Змн.	Арк.	№ докум.	Підпис	Дата	Код сервісу користувачів	Літ.	Маса	Масштаб
Розроб.		Панчук Д.В.						
Перевір.		Мельничук А.Ю.				Арк.	Аркушів	1
Т. Контр.						Циклова комісія Комп'ютерної інженерії		
Реценз.								
Н. Контр.								
Затверд.		Тащук О.Ю.						

```
        name,
      });

      await newUser.save();

      return newUser;
    } catch (error) {
      throw error;
    }
  }

  async getUser(userEmail: string) {
    const email = userEmail.toLowerCase();
    const user = await this.userModel.findOne({ email });
    return user;
  }

  async getUserById(userId: string) {
    try {
      const user = await this.userModel.findById(userId);
      if (!user) {
        throw new Error('User not found');
      }
      return user;
    } catch (error) {
      throw new Error('Error getting user data');
    }
  }

  async updateUserName(userId: string, newName: string) {
    try {
      const updatedUser = await this.userModel.findOneAndUpdate(
        { _id: userId },
        { name: newName },
        { new: true },
```

Змн.	Арк.	№ докум.	Підпис	Дата	Код сервісу користувачів	Літ.	Маса	Масштаб
Розроб.		Панчук Д.В.						
Перевір.		Мельничук А.Ю.				Арк.	Аркушів	1
Т. Контр.						Циклова комісія Комп'ютерної інженерії		
Реценз.								
Н. Контр.								
Затверд.		Тащук О.Ю.						

```
        );

        if (!updatedUser) {
          throw new Error('User not found');
        }
        return { message: 'User name updated successfully' };
      } catch (error) {
        throw new Error('Error updating user name: ' + error.message);
      }
    }

    async addTransaction(userId: string, newTransaction: Transaction) {
      try {
        const transactionId = new mongoose.Types.ObjectId();
        const result = await this.userModel.findOneAndUpdate(
          { _id: userId },
          { $push: { transactions: { _id: transactionId, ...newTransaction
} } },
          { new: true },
        );

        if (!result) {
          throw new Error('User not found');
        }

        return { ...newTransaction, id: transactionId };
      } catch (error) {
        throw new Error('Error adding transaction: ' + error.message);
      }
    }

    async deleteTransaction(userId: string, transactionId: string) {
      try {
```

Змн.	Арк.	№ докум.	Підпис	Дата	**Код сервісу користувачів**	Літ.	Маса	Масштаб
Розроб.		Панчук Д.В.						
Перевір.		Мельничук А.Ю.				Арк.	Аркушів	1
Т. Контр.								
Реценз.						Циклова комісія Комп'ютерної інженерії		
Н. Контр.								
Затверд.		Тащук О.Ю.						

```
      const result = await this.userModel.findByIdAndUpdate(
        userId,
        { $pull: { transactions: { _id: transactionId } } },
        { new: true },
      );

      if (!result) {
        throw new Error('User not found');
      }

      return { message: 'Transaction deleted successfully' };
    } catch (error) {
      throw new Error('Error deleting transaction: ' + error.message);
    }
  }
}
```

Змн.	Арк.	№ докум.	Підпис	Дата	**Код сервісу користувачів**	Літ.	Маса	Масштаб
Розроб.		Панчук Д.В.						
Перевір.		Мельничук А.Ю.				Арк.	Аркушів	1
Т. Контр.								
Реценз.						Циклова комісія Комп'ютерної інженерії		
Н. Контр.								
Затверд.		Тащук О.Ю.						

Налаштування клієнта API (apiClient) у фронтенді

```
import axios from "axios";

export const apiClient = axios.create({
  baseURL: import.meta.env.VITE_API_URL,
  responseType: "json",
});
```

					Налаштування клієнта API	Літ.	Маса	Масштаб
Змн.	Арк.	№ докум.	Підпис	Дата			
Розроб.		Панчук Д.В.					
Перевір.		Мельничук А.Ю.				Арк.	Аркушів 1
Т. Контр.							
Реценз.						Циклова комісія Комп'ютерної інженерії	
Н. Контр.							
Затверд.		Тащук О.Ю.					

Компонент Button у фронтенді

```
function Button({
  children,
  type = "button",
  className = "",
  onClick,
  isLoading = false,
  isDisabled = false,
}) {
  return (
    <button
      type={type}
      disabled={isDisabled}
      onClick={onClick}
      className={`bg-formItemsBackground rounded-xl xs:text-sm border-0
disabled:cursor-not-allowed outline-none text-textColor px-3 py-1 h-10 text-xl
cursor-pointer hover:bg-[#150C16] ${className}`}>
      {isLoading ? "Обробка..." : children}
    </button>
  );
}

export default Button;
```

					Компонент Button	*Літ.*	*Маса*	*Масштаб*
Змн.	*Арк.*	*№ докум.*	*Підпис*	*Дата*				
Розроб.		*Панчук Д.В.*						
Перевір.		*Мельничук А.Ю.*				*Арк.*	*Аркушів* 1	
Т. Контр.						*Циклова комісія Комп'ютерної інженерії*		
Реценз.								
Н. Контр.								
Затверд.		*Тащук О.Ю.*						

Компонент CustomSelector у фронтенді

```
import React, { useState } from "react";

const CustomSelector = ({ options, onSelect }) => {
  const [selectedOption, setSelectedOption] = useState(null);

  const handleOptionSelect = (option) => {
    setSelectedOption(option);
    onSelect(option);
  };

  return (
    <div className='h-10 rounded-xl text-textColor mt-6 text text-xl bg-
formItemsBackground px-2'>
      <ul>
        {options.map((option) => (
          <li
            key={option.value}
            onClick={() => handleOptionSelect(option)}
            className={option === selectedOption ? "selected" : ""}>
            {option.label}
          </li>
        ))}
      </ul>
    </div>
  );
};

export default CustomSelector;
```

					Компонент CustomSelector	Літ.	Маса	Масштаб
Змн.	Арк.	№ докум.	Підпис	Дата			
Розроб.		Панчук Д.В.					
Перевір.		Мельничук А.Ю.				Арк.	Аркушів 1
Т. Контр.							
Реценз.						Циклова комісія Комп'ютерної інженерії	
Н. Контр.							
Затверд.		Тащук О.Ю.					

Компонент Dialog у фронтенді

```
import { useEffect, useRef } from "react";
import { createPortal } from "react-dom";

const portalRoot = document.getElementById("portal-root") ||
document.body;

export default function Dialog({ children, className = "", handleClose })
{
  const elRef = useRef(document.createElement("div"));

  useEffect(() => {
    portalRoot.appendChild(elRef.current);

    return () => {
      portalRoot.removeChild(elRef.current);
    };
  }, []);

  return createPortal(
    <div
      onClick={handleClose}
      className='w-screen bg-black/70 z-50 h-full md:h-full fixed top-0
left-0 flex'>
      <div
        onClick={(e) => e.stopPropagation()}
        className={`rounded-xl relative m-auto z-50 border-[1px] border-
textColor bg-[#0d0d0d] p-5 w-fit h-fit flex ${className}`}>
        {children}
      </div>
    </div>,
    elRef.current
  );
}
```

					Компонент Dialog	Літ.	Маса	Масштаб
Змн.	Арк.	№ докум.	Підпис	Дата				
Розроб.		Панчук Д.В.					
Перевір.		Мельничук А.Ю.				Арк.	Аркушів 1
Т. Контр.							
Реценз.						Циклова комісія Комп'ютерної інженерії	
Н. Контр.							
Затверд.		Тащук О.Ю.					

Компонент Info у фронтенді

```
import Label from "../Label";
import Amount from "../Amount";
import { formatNumber } from "../../helpers/number";
import { TransactionType } from "../../utils/enumTransactions";

function Info({ transactions }) {
  const calculateBalance = () => {
    return transactions.reduce((total, transaction) => {
      const amount = parseFloat(transaction.amount) || 0;
      return transaction.type === TransactionType.EXPENSE
        ? total - amount
        : total + amount;
    }, 0);
  };

  const calculateIncome = () => {
    return transactions.reduce((total, transaction) => {
      const amount = parseFloat(transaction.amount) || 0;
      return transaction.type === TransactionType.INCOME
        ? total + amount
        : total;
    }, 0);
  };

  const calculateExpenses = () => {
    return transactions.reduce((total, transaction) => {
      const amount = parseFloat(transaction.amount) || 0;
      return transaction.type === TransactionType.EXPENSE
        ? total + amount
        : total;
    }, 0);
  };
```

```
            const balance = calculateBalance();
            const income = calculateIncome();
            const expenses = calculateExpenses();

            return (
              <div className='w-full h-60 xs:h-36 md:h-44 bg-[#704E71] rounded-xl
    flex'>
                <div className='h-4/6 w-10/12 m-auto flex flex-col'>
                  <div className='h-3/6 w-full flex justify-between xs:mb-3'>
                    <div className='flex flex-col '>
                      <Label className='!text-xl xs:!text-base md:!text-base'>
                        Баланс
                      </Label>
                      <Amount className='!text-4xl xs:!text-xl md:!text-2xl'>
                        {formatNumber(balance)}
                      </Amount>
                    </div>
                  </div>
                  <div className='h-3/6 w-full flex'>
                    <div className='flex flex-col justify-end'>
                      <Label>Доходи</Label>
                      <Amount sign='+'>{formatNumber(income)}</Amount>
                    </div>
                    <div className='ml-8 flex flex-col justify-end'>
                      <Label>Витрати</Label>
                      <Amount sign='-'>{formatNumber(expenses)}</Amount>
                    </div>
                  </div>
                </div>
              </div>
            );
          }

          export default Info;
```

					Компонент Info	Літ.	Маса	Масштаб
Змн.	Арк.	№ докум.	Підпис	Дата				
Розроб.		Панчук Д.В.				Арк.	Аркушів	1
Перевір.		Мельничук А.Ю.				Циклова комісія Комп'ютерної інженерії		
Т. Контр.								
Реценз.								
Н. Контр.								
Затверд.		Тащук О.Ю.						

Компонент RegisterTransactions у фронтенді

```
import React, { useState, useEffect } from "react";
import ButtonAdd from "../ButtonAdd";
import Transaction from "./components/Transaction";
import { formatUkrainianDateTime } from "../../helpers/date";

function RegisterTransactions({
transactions,
actionButtonClick,
deleteTransactions,
}) {
const [transactionsByDay, setTransactionsByDay] = useState({});

useEffect(() => {
const sortedTransactions = [...transactions].sort(
(a, b) => new Date(b.date) - new Date(a.date)
);

const groupedTransactions = {};

sortedTransactions.forEach((transaction) => {
const date = new Date(transaction.date);
const day = date.getDate();
const month = date.toLocaleString("default", { month: "long" });
const year = date.getFullYear();
const formattedDate = `${day} ${month} ${year}`;

if (!groupedTransactions[formattedDate]) {
groupedTransactions[formattedDate] = [];
}

groupedTransactions[formattedDate].push(transaction);
});
```

					Компонент RegisterTransactions	Літ.	Маса / Масштаб
Змн.	Арк.	№ докум.	Підпис	Дата			
Розроб.		Панчук Д.В.					
Перевір.		Мельничук А.Ю.				Арк.	Аркушів 1
Т. Контр.							
Реценз.						Циклова комісія Комп'ютерної інженерії	
Н. Контр.							
Затверд.		Тащук О.Ю.					

```
setTransactionsByDay(groupedTransactions);
}, [transactions]);

return (
<div className='flex flex-col'>
<div className='h-80 w-full overflow-y-auto text-textColor flex'>
<div className='h-fit flex-col mx-auto w-11/12 mt-3 flex pt-5'>
{Object.keys(transactionsByDay).map((date) => (
<div key={date} className=''>
<div className='w-full'>
<p className='text-lg text-center italic'>{date}</p>
</div>
{transactionsByDay[date].map((transaction) => (
<Transaction
transaction={transaction}
key={transaction.id}
deleteTransactions={deleteTransactions}
/>
))}
</div>
))}
</div>
</div>
<ButtonAdd onClick={actionButtonClick} />
</div>
);
}

export default RegisterTransactions;
```

Змн.	Арк.	№ докум.	Підпис	Дата	**Компонент RegisterTransactions**	Літ.	Маса	Масштаб
Розроб.		Панчук Д.В.						
Перевір.		Мельничук А.Ю.				Арк.	Аркушів	1
Т. Контр.						Циклова комісія Комп'ютерної інженерії		
Реценз.								
Н. Контр.								
Затверд.		Тащук О.Ю.						

Компонент TrackerActions у фронтенді

```
import { useState } from "react";
import InputWrapper from "../InputWrapper";
import { ErrorMessage, Field, Form, Formik } from "formik";
import Button from "../Button";
import { useEffect } from "react";
import { transactionValidationSchema } from
"../../validations/transaction.schema";
import { apiClient } from "../../app/apiClient";

function TrackerActions({ setTransactions, userId }) {
  const initialValues = {
    description: "",
    amount: "",
    type: "Надходження",
    date: "",
  };

  const [maxDate, setMaxDate] = useState("");

  const cancelClick = () => {
    clearStateOperation();
  };

  useEffect(() => {
    updateMaxDate();
  }, []);

  function updateMaxDate() {
    const today = new Date().toISOString().split("T")[0];
    setMaxDate(today);
  }

  const createTransactions = async (values, { resetForm }) => {
```

Змн.	Арк.	№ докум.	Підпис	Дата	Компонент RegisterTransactions	Літ.	Маса	Масштаб
Розроб.		Панчук Д.В.						
Перевір.		Мельничук А.Ю.						
Т. Контр.						Арк.	Аркушів	1
Реценз.						Циклова комісія Комп'ютерної інженерії		
Н. Контр.								
Затверд.		Тащук О.Ю.						

```
            try {
              const responce = await apiClient.post("users/transaction", {
                ...values,
                amount: +values.amount,
                userId: userId,
              });
              const { data } = responce;
              setTransactions((prev) => [...prev, data]);
              resetForm();
            } catch (error) {}
          };

          return (
            <div className='w-full h-5/6 flex flex-col mt-5 cursor-pointer'>
              <Formik
                initialValues={initialValues}
                validationSchema={transactionValidationSchema}
                onSubmit={createTransactions}>
                {({ values }) => (
                  <Form className='w-11/12 mx-auto bg-backgroundForm rounded-t-
xl h-full flex mt-2'>
                    <div className='w-10/12 mx-auto flex flex-col'>
                      <div className='mt-4'>
                        <Field
                          type='text'
                          inputName='description'
                          name='description'
                          labelValue='Опис'
                          as={InputWrapper}
                        />
                        <ErrorMessage
                          className='text-red-600 text-xs'
                          name='description'
                          component='span'
```

Змн.	Арк.	№ докум.	Підпис	Дата	**Компонент RegisterTransactions**	Літ.	Маса	Масштаб
Розроб.		Панчук Д.В.						
Перевір.		Мельничук А.Ю.						
Т. Контр.						Арк.	Аркушів	1
Реценз.						Циклова комісія Комп'ютерної інженерії		
Н. Контр.								
Затверд.		Тащук О.Ю.						

```
        />
      </div>
      <div className='mt-4'>
        <Field
          type='text'
          labelValue='Сума ₴'
          inputName='amount'
          name='amount'
          as={InputWrapper}
        />
        <ErrorMessage
          className='text-red-600 text-xs'
          name='amount'
          component='span'
        />
      </div>
      <div className='mt-4'>
        <Field
          labelValue='Дата'
          inputName='date'
          type='date'
          name='date'
          required
          max={maxDate}
          as={InputWrapper}
        />
        <ErrorMessage
          className='text-red-600 text-xs'
          name='date'
          component='span'
        />
      </div>
      <div className='flex items-center justify-evenly my-4'>
        <div className=''>
```

Змн.	Арк.	№ докум.	Підпис	Дата	**Компонент RegisterTransactions**	Літ.	Маса	Масштаб
Розроб.		Панчук Д.В.						
Перевір.		Мельничук А.Ю.						
Т. Контр.						Арк.	Аркушів	1
Реценз.						Циклова комісія Комп'ютерної інженерії		
Н. Контр.								
Затверд.		Тащук О.Ю.						

```
            <Field
              id='expence-radio'
              type='radio'
              name='type'
              value='Надходження'
              checked={values.type === "Надходження"}
            />
            <label
              htmlFor='expence-radio'
              className='ml-3 text-textColor text-xl'>
              Надходження
            </label>
          </div>
          <div className=''>
            <Field
              id='income-radio'
              type='radio'
              value='Витрата'
              checked={values.type === "Витрата"}
              name='type'
            />
            <label
              htmlFor='income-radio'
              className='ml-3 text-textColor text-xl'>
              Витрата
            </label>
          </div>
        </div>
        <div className='flex items-center'></div>

        <div className='mb-2 mt-5 flex mx-auto gap-5'>
          <Button onClick={cancelClick}>Відмінити</Button>
          <Button type='submit'>Підтвердити</Button>
        </div>
```

Змн.	Арк.	№ докум.	Підпис	Дата	Компонент RegisterTransactions	Літ.	Маса	Масштаб
Розроб.		Панчук Д.В.						
Перевір.		Мельничук А.Ю.				Арк.	Аркушів 1	
Т. Контр.						Циклова комісія Комп'ютерної інженерії		
Реценз.								
Н. Контр.								
Затверд.		Тащук О.Ю.						

```
                </div>
              </Form>
            )}
          </Formik>
        </div>
      );
    }

    export default TrackerActions;
```

					Компонент	*Літ.*	*Маса*	*Масштаб*
Змн.	*Арк.*	*№ докум.*	*Підпис*	*Дата*				
Розроб.		*Панчук Д.В.*			**RegisterTransactions**			
Перевір.		*Мельничук А.Ю.*						
Т. Контр.						*Арк.*	*Аркушів* 1	
Реценз.						*Циклова комісія Комп'ютерної інженерії*		
Н. Контр.								
Затверд.		*Тащук О.Ю.*						