МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Сумський державний університет
Факультет електроніки та інформаційних технологій
Кафедра інформаційних технологій

«До захисту допущено»
В.о. завідувача кафедри

_____ Світлана ВАЩЕНКО

____ _____ 2023 р.

КВАЛІФІКАЦІЙНА РОБОТА

на здобуття освітнього ступеня магістр

зі спеціальності 122 «Комп'ютерні науки»,

освітньо-професійної програми «Інформаційні технології проектування»

на тему: «Web-орієнтована система обліку персонального бюджету»

Здобувача (ки) групи ІТ.м-24 Денисенко Павло Дмитрович
(шифр групи) (прізвище, ім'я, по батькові)

Кваліфікаційна робота містить результати власних досліджень. Використання ідей, результатів і текстів інших авторів мають посилання на відповідне джерело.

_____ Павло Денисенко
(підпис) (Ім'я та ПРІЗВИЩЕ здобувача)

Керівник ст. викладач, к.т.н., доц., Наталія Федотова _____
(посада, науковий ступінь, вчене звання, Ім'я та ПРІЗВИЩЕ) (підпис)

Суми – 2023

Сумський державний університет
Факультет електроніки та інформаційних технологій
Кафедра інформаційних технологій
Спеціальність 122 «Комп'ютерні науки»
Освітньо-професійна програма «Інформаційні технології проектування»

ЗАТВЕРДЖУЮ
В.о. завідувача кафедри ІТ

Світлана ВАЩЕНКО
«_____» _____ 2023 р.

ЗАВДАННЯ

на кваліфікаційну роботу магістра студентові

Денисенко Павло Дмитрович
(прізвище, ім'я, по батькові)

1 Тема кваліфікаційної роботи «Web-орієнтована система обліку персонального бюджету»

затверджена наказом по університету від «08» листопада 2023 р. № 1249-VI

2 Термін здачі студентом кваліфікаційної роботи «___» ___грудня___ 2023 р.

3 Вхідні дані до кваліфікаційної роботи перелік вимог до розробки web-орієнтованої системи

4 Зміст розрахунково-пояснювальної записки (перелік питань, що їх належить розробити) аналіз предметної області, постановка задачі та методи дослідження, проектування web-орієнтованої системи, реалізація web-орієнтованої системи

5 Перелік графічного матеріалу (з точним зазначенням обов'язкових слайдів презентації) контекстна діаграма IDEF0, діаграма декомпозиції IDEF0, діаграма Use Case, ER-діаграма бази даних, архітектура web-додатку

6. Консультанти випускної роботи із зазначенням розділів, що їх стосуються:

Розділ	Консультант	Підпис, дата	
		Завдання видав	Завдання прийняв
Визначення мети	*Федотова Н.А.*		
Планування	*Федотова Н.А.*		
Визначення вимог	*Федотова Н.А.*		
Календарний план	*Федотова Н.А.*		

Дата видачі завдання _____.

Керівник _____
(підпис)

Завдання прийняв до виконання _____
(підпис)

КАЛЕНДАРНИЙ ПЛАН

№ п/п	Назва етапів кваліфікаційної роботи	Термін виконання етапів роботи	Примітка
1	Аналіз предметної області	24.10 – 31.10	
2	Планування робіт	01.11 – 17.11	
3	Практична реалізація	20.11 – 07.12	
4	Тестування та презентація	08.12 – 15.12	

Магістрант _____ Павло ДЕНИСЕНКО

Керівник роботи _____ к.т.н., доц. Наталія ФЕДОТОВА

АНОТАЦІЯ

Тема кваліфікаційної роботи магістра «Web-орієнтована система обліку персонального бюджету».

Пояснювальна записка складається зі вступу, 4 розділів, висновків, списку використаних джерел із 45 найменувань, додатків. Загальний обсяг роботи − 83 сторінок, у тому числі 44 сторінок основного тексту, 5 сторінок списку використаних джерел, 28 сторінок додатків.

Актуальність роботи полягає в зростанні інтересу до фінансової грамотності, необхідності ефективного управління фінансами та зручному доступі до інформації. Тому є потреба в інструменті для аналізу витрат.

Мета роботи: розробка та впровадження веб-орієнтованої системи обліку персонального бюджету.

Робота включає в себе аналіз існуючих підходів до управління фінансами, проектування інтерфейсу користувача, вибір технологічного стеку для веб-розробки, а також реалізацію функціональності системи, яка дозволяє зручно вести облік доходів та витрат, аналізувати та оптимізувати фінансові витрати. Робота спрямована на полегшення процесу управління особистими фінансами та забезпечення ефективного контролю над бюджетним плануванням.

Практичне значення полягає в забезпеченні зручного та ефективного контролю над фінансами, сприянні точному відстеженню витрат, а також допомозі у плануванні власного бюджету користувача через доступ до системи з будь-якого пристрою з Інтернет-підключенням завдяки використанню Web-орієнтованої системи обліку персонального бюджету .

Ключові слова: web-орієнтована система, витрати, персональний бюджет, React JS, REST API, ендпойнт.

ЗМІСТ

ВСТУП

Актуальність. У сучасному світі, де швидкі технологічні зміни і зростання обсягу електронних фінансових транзакцій визначають нові вимоги до управління фінансами, здатність ефективно вести облік фінансів, розуміти та контролювати витрати стає критичною для багатьох людей, які прагнуть досягти фінансової стабільності та управління своїми ресурсами. Розвиток технологій, зокрема веб-розробки, надає унікальні можливості для створення зручних та ефективних інструментів управління фінансами, які легко доступні з будь-якого пристрою з підключенням до Інтернету. Система обліку персонального бюджету відповідає цим потребам та надає користувачам інструмент для досягнення фінансової відповідальності. Нижче наведено ключові аспекти актуальності розробки веб-орієнтованої системи обліку персонального бюджету:

1. Зростання інтересу до фінансової грамотності:

У сучасному світі існує зростаюча необхідність в розвитку фінансової грамотності серед населення. Веб-орієнтована система обліку персонального бюджету надасть користувачам зручний інструмент для вивчення та аналізу їхніх фінансів.

2. Необхідність ефективного управління фінансами:

З ростом складності фінансового життя і нестабільністю економічного середовища, особистий контроль над фінансами стає ключовим елементом стабільності та благополуччя. Веб-орієнтована система допоможе користувачам здійснювати ефективний контроль над своїми витратами та доходами.

3. Зручний доступ до інформації:

Оскільки сучасне суспільство характеризується великою мобільністю та високим рівнем використання Інтернету, веб-орієнтована система обліку фінансів забезпечить користувачам зручний доступ до інформації з будь-якого пристрою та місця.

4. Потреба в інструменті для аналізу витрат:

Користувачі все частіше виявляють бажання не лише вести облік своїх фінансів, але й отримувати аналітичні дані та рекомендації стосовно оптимізації витрат. Розробка веб-системи, яка надає інструменти для аналізу витрат, відповідає цій потребі.

5. Розвиток технологій та електронних сервісів:

Завдяки стрімкому розвитку технологій та високій доступності Інтернету, веб-орієнтовані фінансові сервіси стають все більш популярними серед користувачів, які шукають зручні та інноваційні рішення.

Отже, розробка веб-орієнтованої системи обліку персонального бюджету є актуальним та важливим напрямком, який відповідає потребам сучасного суспільства в ефективних інструментах фінансового управління.

Об’єкт дослідження – процес обліку персонального бюджету.

Предмет дослідження –інформаційне забезпечення обліку персонального бюджету. Розглядаються аспекти технічної реалізації та функціональності системи для забезпечення її ефективності та зручності використання.

Метою даної роботи є розробка та впровадження веб-орієнтованої системи обліку персонального бюджету, яка дозволить користувачам ефективно вести контроль над своїми фінансами та забезпечити зручний інструмент для аналізу витрат.

Для досягнення поставленої мети проекту необхідно реалізувати ряд ключових завдань:

- Проаналізувати предметну область для визначення актуальності та цільової аудиторії використання системи, провести порівняльний аналіз існуючих інформаційних систем.
- Визначити технології для розробки системи.
- Розробити модель та структуру інформаційної системи.
- Здійснити програмну реалізацію структури та функціоналу системи.
- Провести тестування.

Практичне цінність полягає в забезпеченні зручного та ефективного контролю над фінансами, сприянні точному відстеженню витрат, а також допомозі у плануванні власного бюджету користувача через доступ до системи з будь-якого пристрою з Інтернет-підключенням..

Гіпотеза дослідження:

Припускається, що розробка та впровадження веб-орієнтованої системи обліку персонального бюджету, заснованої на сучасних технологіях та забезпеченої високою ступенем функціональності, сприятиме підвищенню фінансової грамотності та ефективному управлінню фінансами користувачів. При цьому, оптимальний вибір технологій, структури та функціоналу системи дозволить створити зручний та доступний інструмент для ведення обліку та аналізу витрат, що сприятиме формуванню позитивної фінансової поведінки серед користувачів. Таким чином, гіпотеза передбачає позитивний вплив веб-орієнтованої системи на фінансову дисципліну та здатність користувачів до ефективного управління своїм персональним бюджетом.

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Огляд останніх досліджень і публікацій

Фінансова поведінка громадян напряму впливає на фінансовий ринок, тіньову економіку та ставлення до реформ. Опитування ОЕСР показує, що українці більше орієнтовані на короткострокову перспективу та схильні до витрат, а не заощаджень, що пояснюється історичним контекстом та впливом російської агресії.

Згідно [1], індекс фінансової поведінки українців, складає 5,2 з 9. Це вище, ніж у сусідніх країнах (4,8), але нижче за середній бал 5,4 у тридцяти країнах, що брали участь в опитуванні.

Бажання людей використовувати фінансові послуги залежить від їхньої здатності розуміти фінансові продукти, впевненості в збереженні грошей і впевненості, що спори будуть вирішені швидко й справедливо [2]. Це вказує, що недостатній рівень грамотності та довіри призводить до меншого використання фінансових механізмів.

Індекс фінансової грамотності українців за останні три роки зріс до 12,3 балів, в порівнянні з 11,6 балами у 2018 році. Про це свідчать результати дослідження "Фінансова грамотність, фінансова інклюзія та фінансовий добробут в Україні у 2021 році", проведеного Проєктом USAID "Трансформація фінансового сектору" у співпраці з Національним банком України. [3]

НБУ готує Національну стратегію розвитку фінансової грамотності до 2025 року разом із міністерствами та відомствами[3]. Де вказано, що «розрив у фінансовій грамотності залишається між містами та сілами, але загалом індекс українців поки що становить 58% від максимального значення. Найвищий рівень спостерігається у вікових групах 25–34 роки та 30–59 років». Освіченість прямо корелює з рівнем фінансової грамотності: вища освіта - вищий бал.

Дослідження також вказує на важливість інформаційних технологій та застосування фінансових знань на практиці для підвищення грамотності. Варто відзначити, що українці частіше концентруються на короткострокових планах та витратах, ніж на заощадженні. Є потреба в подальшій освіті з фінансової грамотності, особливо серед молоді та людей старше 60 років.

Також, багато українських родин відчувають фінансові труднощі, що виникають через несправедливий розподіл доходів. Гостра нестача коштів до зарплати змушує звертатися до кредитних організацій. Вирішення цієї проблеми полягає в управлінні сімейним бюджетом. [4]

Сімейний бюджет – це стратегічний план доходів та витрат, що дозволяє отримати повну картину фінансів для ефективного їх використання. Безконтрольна витрата коштів призводить до непорозумінь, боргів, та нестабільності.

Контроль над витратами є головним етапом. Пильність щодо грошових потоків і відповідність бюджету - ключ до розумної економії. Спільне рішення щодо розподілу бюджету допомагає створити сприятливий процес, а не лише обов'язковий контроль. Сімейний бюджет є інструментом розумної економії та забезпечення майбутнього. [5]

Отже, розробка та впровадження таких сервісів, як системи обліку персонального бюджету, є важливим етапом для покращення фінансової грамотності та забезпечення стійкого фінансового майбутнього громадян України. Це буде сприяти покращенню фінансової поведінки громадян в цілому.

1.2 Аналіз інформаційних систем

Багато фінансових експертів та спеціалістів з особистих фінансів погоджуються, що створення бюджету – ключ до контролю над своїми фінансами. В роботі [6] розглянуті ключові параметри, які рекомендовані до врахування при створення додатку.

Багато компаній створили додатки, які виконують всю важку роботу за нас. Зазвичай бюджетні додатки пов'язані з банківськими рахунками і кредитними картками та автоматично відстежують покупки. Вони категоризують витрати за різні сфери, щоб точно знати, куди йдуть кошти. Використовуючи хороший бюджетний додаток, люди економлять час, який інакше витратили б на ручне введення цифр в електронну таблицю, при цьому спонукаючи себе витрачати менше і заощаджувати більше. [7]

Проаналізуємо найбільш популярні на даний момент додатки та web-системи.

1. **YNAB (You Need A Budget)**: – це додаток і сервіс для обліку особистого бюджету, який базується на принципах фінансової відповідальності. YNAB допомагає користувачам створити бюджет, визначити фінансові цілі та вести контроль над своїми витратами. Зосереджений на плануванні і ефективному використанні грошей. [8]

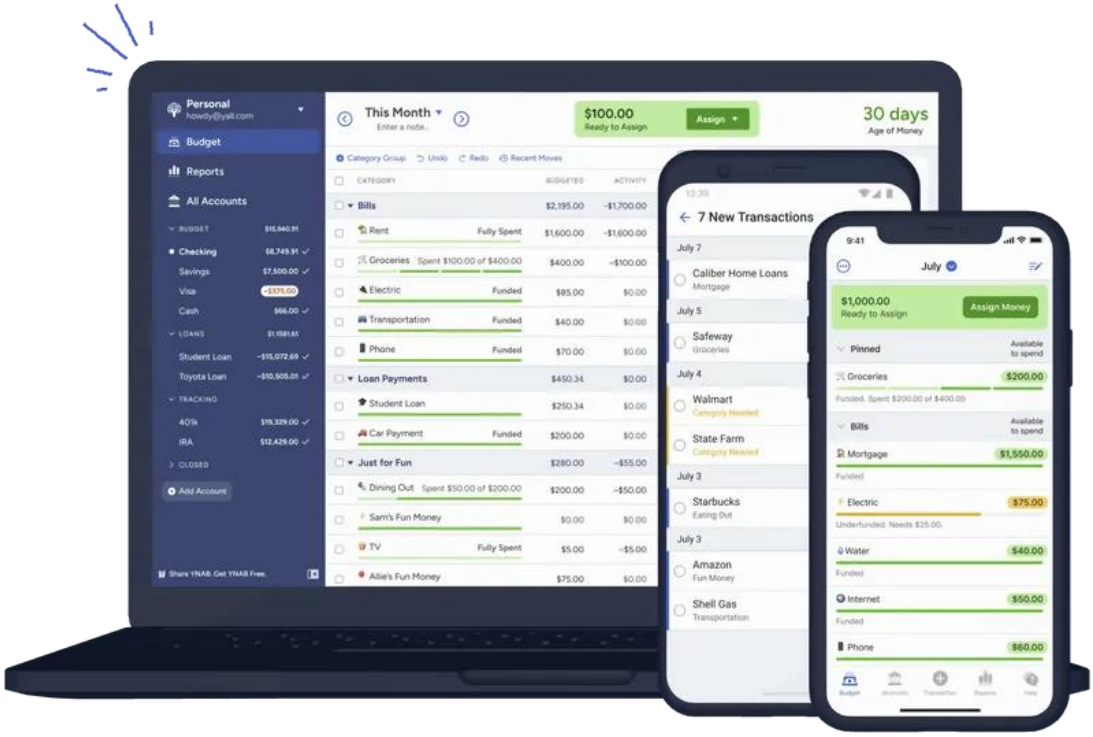

Рисунок 1.1 – YNAB – вигляд інтерфейсу рішення

Джерело:[8]

Переваги: Інноваційний підхід до бюджетування, акцент на цілеспрямованому витрачанні.

Недоліки: Платна підписка, можливий більший поріг входження для новачків.

2. **Mint** – є безкоштовним сервісом для управління фінансами, який автоматично інтегрується з банківськими рахунками та кредитними картками користувачів. Надає можливості відстеження витрат, створення бюджету та нагадує про рахунки. [9]

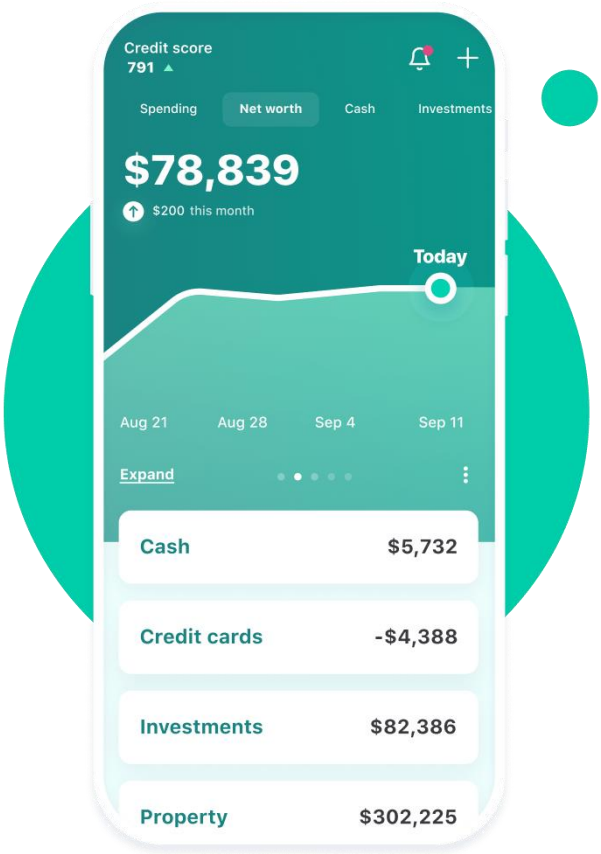

Рисунок 1.2 – Mint – вигляд застосунку

Джерело:[9]

Переваги: Безкоштовний, автоматично відстежує та категоризує витрати.

Недоліки: Можливість проблем із безпекою, не завжди точний розподіл за категоріями.

3. **PocketGuard** – відслідковує витрати, автоматично категоризує витрати та допомагає користувачам планувати економії. Аналізує фінансовий стан та пропонує рекомендації щодо покращення фінансового благополуччя. [10]

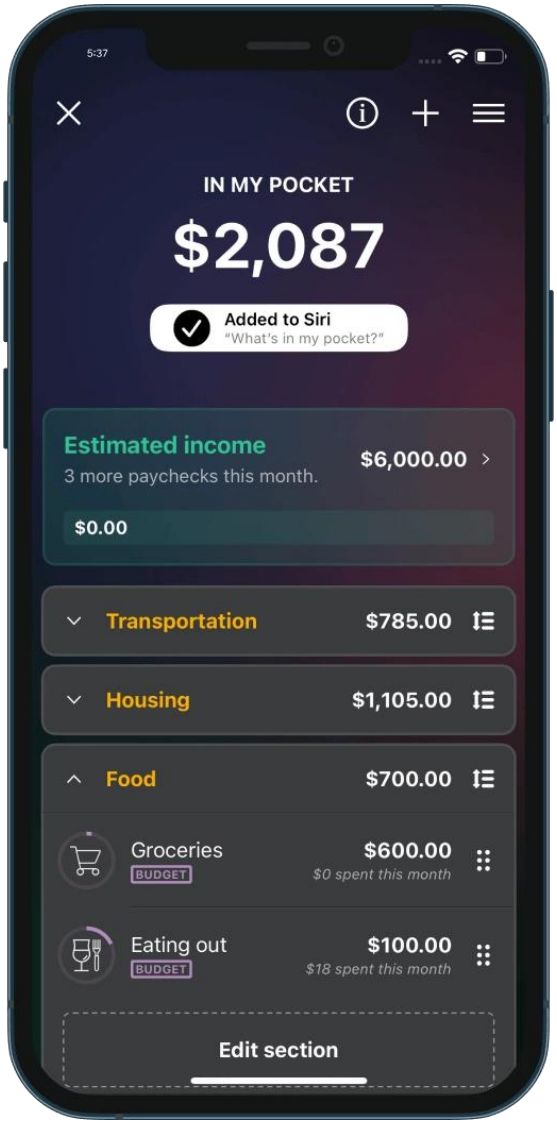

Рисунок 1.3 – PocketGuard – вигляд застосунку

Джерело:[10]

Переваги: Простий та ефективний, автоматично оновлює фінансові дані.

Недоліки: Залежність від стабільного інтернет-з'єднання, обмежені можливості для бізнес-користувачів.

4. **GoodBudget** – пропонує концепцію "енвелопового" бюджетування, де користувачі визначають конкретні суми грошей для кожної категорії витрат. Забезпечує планування та ведення обліку бюджету [11].

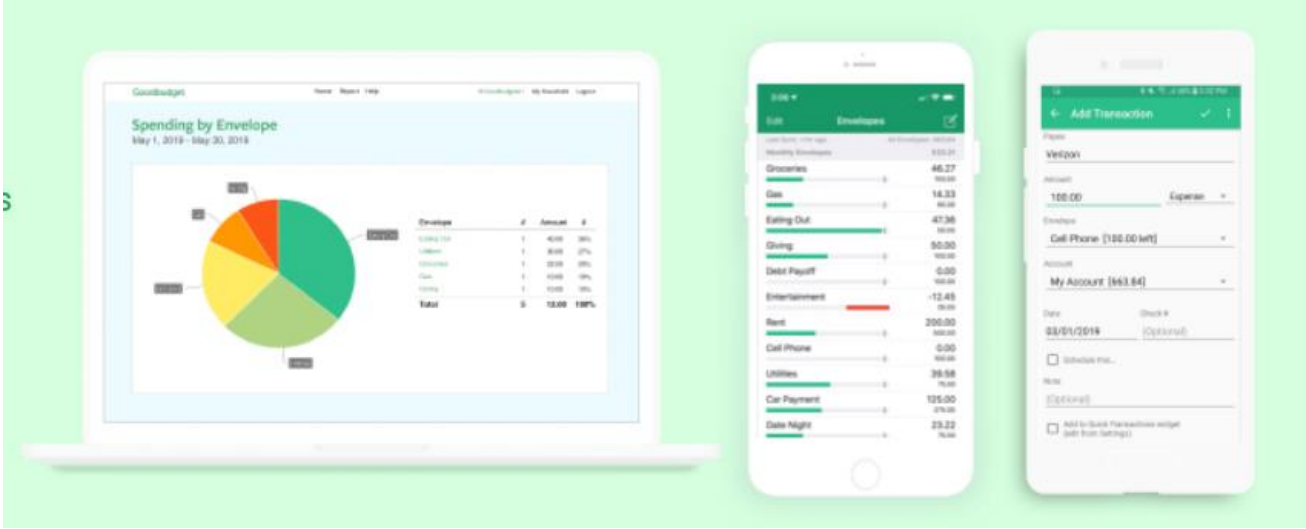

Рисунок 1.4 – GoodBudget – вигляд інтерфейсу рішення

Джерело:[11]

Переваги: Простий у використанні, фокус на діловій моделі розподілу грошей.

Недоліки: Обмежена автоматизація та інтеграція, більші можливості за плату.

Проведений аналіз різноманітних додатків для обліку особистого бюджету вказує на те, що кожен із них має свої переваги та недоліки. Найбільші з них можна виділити такі, як: платна підписка, мовна адаптація, тощо.

З урахуванням цих факторів, написання нового додатка для обліку особистого бюджету, спеціально адаптованого до потреб українських користувачів, є виправданим завданням, що підвищить ефективність його використання та користь для користувачів в першу чергу в нашій країні.

1.3 Результат дослідження аналогів

Під час ретельного аналізу існуючих аналогів були виявлені головні недоліки, які потребують виправлення у нашій власній розробці. З метою систематизації цих виявлених проблем була створена таблиця (див. таблицю 1.1) з порівняльними критеріями.

Таблиця 1.1 – Порівняння досліджених інформаційних систем.

Назва	**YNAB**	**Mint**	**PocketGuard**	**GoodBudget**	**Розроблювана система**
Вид підписки	платна	безкоштовна	умовно безкоштовна	умовно безкоштовна	безкоштовна
Складність інтерфейсу	дуже складна	складна	складна	складна	легко
Швидкість освоєння користувачем	дуже повільно	повільно	повільно	повільно	швидко
Наявність вибору валют	присутня	відсутня	відсутня	відсутня	присутня
Наявність української мови	відсутня	відсутня	відсутня	відсутня	присутня

Джерело: побудовано автором

2 ПОСТАНОВКА ЗАДАЧІ ТА МЕТОДИ ДОСЛІДЖЕННЯ

2.1 Мета та задачі дослідження

Метою даної роботи є розробка та впровадження web-орієнтованої системи обліку персонального бюджету. Головною метою є створення ефективного та зручного інструменту для користувачів, який дозволить їм ефективно вести облік власних фінансів, планувати та аналізувати витрати, сприяючи тим самим раціональному та виваженому управлінню особистим бюджетом.

Крім того, є необхідність у використанні передових технологій розробки веб-додатків для побудови надійної, безпечної та функціональної системи. Проект передбачає використання сучасних інструментів та практик для розробки якісного інтерфейсу користувача, забезпечення конфіденційності фінансової інформації, та створення інтуїтивно зрозумілого середовища для взаємодії з додатком.

Також, впровадження даного веб-додатку передбачено сприяти підвищенню фінансової грамотності користувачів, допомагаючи їм усвідомлювати та ефективно використовувати свої фінансові ресурси. Розробка інструменту для ведення обліку персонального бюджету також має на меті забезпечити зручний та доступний засіб для фінансового планування, що сприяє підвищенню рівня фінансової свідомості серед користувачів.

Завдання для досягнення поставленої мети проекту з декілька етапів, а саме:

1. Детальний аналіз проблемної області та вивчення актуальності:

- Провести аналіз проблем, пов'язаних із управлінням персональним бюджетом.
- Розглянути сучасні публікації та дослідження для визначення актуальності теми та ідентифікації потреб цільової аудиторії.

2. Порівняльний аналіз існуючих інформаційних систем:

- Вивчити функціонал та особливості існуючих веб-орієнтованих систем обліку бюджету.
- Визначити переваги та недоліки конкурентних рішень.

3. Вибір оптимальних технологій:

- Провести дослідження різних технологій для розробки веб-систем.
- Визначити оптимальний стек технологій, який враховує вимоги проекту, швидкість розробки та масштабованість.

4.Розробка моделі та структури інформаційної системи:

- Створити детальну модель системи, включаючи базу даних, логічну та фізичну архітектуру.
- Розробити структуру інтерфейсу користувача для забезпечення зручності взаємодії з системою.

5.Програмна реалізація структури та функціоналів системи:

- Реалізувати серверну та клієнтську частини веб-системи відповідно до розробленої моделі.
- Забезпечити основні функціональні можливості, такі як ведення обліку, аналіз витрат, генерація статистики та інші.

6.Тестування системи:

- Провести тестування системи для перевірки відповідності функціоналу вимогам та виявлення можливих помилок.

2.2 Методи дослідження

Для розробки було обрано такі основні методи дослідження:

1. **Теоретичний метод**.

Використання теоретичного методу передбачає аналіз особливостей та можливостей предметної області. У контексті розробки web-орієнтованої системи

обліку персонального бюджету, теоретичний метод дозволяє детально вивчити сучасні технології розробки веб-додатків, методи забезпечення безпеки фінансової інформації, а також ефективні та інтуїтивно зрозумілі практики інтерфейсного дизайну.

Використання теоретичного методу виявляється при аналізі бізнес-процесів управління фінансами, виборі оптимальних технологій та інструментів для розробки, а також при визначенні стратегій забезпечення конфіденційності та безпеки даних користувачів.

Для застосування теоретичного методу до даної роботи, були корисні такі ресурси, які були використані під час аналізу існуючих систем та технології:

- **Технології розробки веб-додатків:** Mozilla Developer Network (MDN) [12], W3Schools [13], Stack Overflow [14]
- **Бізнес-процеси управління фінансами:** Harvard Business Review [15], McKinsey & Company Insights [16], Investopedia [17]
- **Забезпечення конфіденційності та безпека даних:** OWASP (Open Web Application Security Project) [18], JSON Web Tokens [19]

2. **Емпіричний метод**.

Емпіричний метод включає в себе застосування спостережень, вимірювань та експериментів для отримання практичних даних. У випадку розробки системи обліку бюджету, цей метод використовується для аналізу принципів та процедур фінансового управління користувачів, враховуючи їхні потреби та переваги.

Для застосування емпіричного методу в розробці системи обліку бюджету та аналізу принципів фінансового управління, були корисні такі ресурси, які були використані для побудови таблиці порівнянь:

- **Статистичні дані та дослідження:** World Bank Open Data [20], International Monetary Fund (IMF) Data [21], Statista [22]
- **Фінансові звіти та аналіз:** U.S. Securities and Exchange Commission (SEC) EDGAR Database [23], Financial Times Markets Data [24].
- **Економічні та фінансові журнали:** Journal of Finance [25], Journal of Financial Economics [26]

- **Спеціалізовані фінансові дослідження та звіти:** Deloitte Insights [27]

  McKinsey & Company Financial Services [28]

3. **Метод моделювання.**

Включає розробку схем та діаграм, що показують процес роботи системи. Для цього проекту використовується діаграма IDEF0 (Integrated DEFinition for Function Modelling) [29].

Діаграма IDEF0 використовується для моделювання функцій системи обліку бюджету, ідентифікації основних етапів обробки даних та визначення взаємозв'язків між різними компонентами системи (наведено у розділ 3).

4. **Системно-функціональний метод.**

Системно-функціональний метод дозволяє досліджувати компоненти системи та їх взаємодії. У розробці системи обліку бюджету цей метод використовується для аналізу функціональних вимог та взаємодії користувачів із системою.

Застосування цих методів дозволяє забезпечити комплексний підхід до розробки, враховуючи як теоретичні аспекти, так і практичні вимоги користувачів та ефективність функціонування системи.

2.3 Технології та інструменти реалізації задач

Для виконання мети, яка була сформульована в розділі 2.1 даної роботи виділимо такі основні технології та інструменти для їх виконання:

1. **Розробка front-end частини:**

Створення інтерфейсу з використанням React JS та Material UI для забезпечення зручного та естетичного користування. Реалізація можливості автентифікації та авторизації користувачів для захисту особистої фінансової інформації.

2. **Розробка back-end частини:**

Використання Express JS (веб-фреймворк для Node.js) для створення серверної частини додатку. Забезпечення взаємодії між front-end і базою даних, обробка запитів користувача.

3. **Створення бази даних:**

Проектування та створення бази даних з використанням PostgreSQL для збереження фінансових даних користувачів. Розробка структури бази даних, що включає таблиці для витрат, категорій, користувачів та інші необхідні об'єкти.

4. **Можливості ведення фінансових записів**:

Реалізація функціоналу додавання, редагування та видалення фінансових записів користувачем. Категоризація витрат для надання деталізованої статистики.

5. **Створення статистики та графіків**:

Використання бібліотеки Chart.js для побудови графіків та діаграм для візуалізації фінансової статистики. Розробка функціоналу для аналізу витрат за певний період часу.

6. **Оптимізація та забезпечення безпеки**:

Забезпечення оптимізації веб-додатка для швидкого завантаження та ефективного використання ресурсів. Впровадження заходів безпеки для захисту фінансових даних користувачів.

7. **Тестування та валідація**:

Проведення різноманітних тестів, включаючи одиничні для перевірки правильності та надійності додатку. Забезпечення валідації введених даних для уникнення помилок та збереження консистентності інформації.

Всі ці визначені задачі охоплюють весь життєвий цикл розробки додатку та є ключовими для досягнення успішної реалізації системи обліку персонального бюджету.

2.4 Обґрунтування вибору технологій

В даному проекті ми будем використовувати такі технології:

1. **React JS** [30]**:**

- *Компонентний підхід:* React пропонує компонентний підхід до розробки інтерфейсу, що дозволяє створювати зручний та масштабований front-end. Компоненти можна повторно використовувати, що спрощує розробку та обслуговування.
- *Велика спільнота:* Значна кількість розробників використовують React, тому можна знайти безліч матеріалів, бібліотек та рішень для швидкої розробки.
- *Єдність мови:* Використання JavaScript [31] як основної мови для розробки дозволяє створювати консистентний та ефективний код як на клієнтській, так і на серверній стороні.

2. **Material UI** [32]**:**

- *Готовий дизайн:* Material UI забезпечує готові та стилізовані компоненти відповідно до принципів Material Design від Google. Це полегшує розробку естетичного та сучасного інтерфейсу.
- *Кастомізація:* Material UI дозволяє легко кастомізувати компоненти, що дає можливість адаптувати їх до унікального дизайну проекту.

3. **Chart.js** [33]**:**

- *Візуалізація даних:* Chart.js надає широкий спектр можливостей для створення інтерактивних та привабливих графіків. З його допомогою користувачі зможуть легко аналізувати свої витрати через різноманітні діаграми та графіки.
- *Легкість використання:* Простий API та зручна документація Chart.js роблять його ідеальним інструментом для швидкої та ефективної візуалізації фінансових даних.

4. **Express JS** [34]**:**

- *Швидкість та простота:* Express - легкий та ефективний фреймворк, що дозволяє швидко розробляти серверну частину додатку. Він ідеально підходить для веб-застосунків з невеликою до середньої складності.

- *Middleware:* Express дозволяє легко використовувати middleware для обробки запитів, аутентифікації користувачів та інших важливих завдань.

5. PostgreSQL [35]**:**

- *Надійність та безпека:* PostgreSQL - це потужна система управління базами даних з високим рівнем надійності та захисту даних. Вона гарантує стабільне зберігання фінансової інформації користувачів.

- *Сумісність з Node.js:* Зручна інтеграція PostgreSQL з Node.js додатком (Express) забезпечує ефективну взаємодію з базою даних, дозволяючи легко виконувати операції збереження та отримання інформації.

Вибір React JS, Material UI та Chart.js для реалізації front-end та візуалізації даних у веб-додатку з обліку персонального бюджету обумовлений численними їх перевагами. Інтеграція цих технологій відбувається легко та безперебійно, надаючи зручність у розробці front-end частини та відображенні статистичних даних. Співробітництво React JS із Material UI дозволяє зручно створювати компоненти із сучасним та стильним дизайном, а Chart.js додає можливості інтерактивних графіків та діаграм.

Активна спільнота визначається як важливий аспект для стабільності та постійного розвитку проекту. React JS, Material UI та Chart.js користуються широкою підтримкою та регулярними оновленнями від розробницької спільноти. Це гарантує, що обрані технології залишатимуться актуальними та отримуватимуть підтримку у майбутньому.

Взаємодія між обраними технологіями гарантує їхню сумісність і робить їх ідеальним вибором для створення стабільного та ефективного front-end. Вони взаємодіють між собою без конфліктів, спрощуючи тим самим процес розробки та утримання системи.

Масштабованість є ще однією перевагою. Вибрані технології не лише відповідають поточним вимогам проекту, а й дозволяють легко розширювати додаток у майбутньому, адаптуючись до зростання обсягу даних та підвищення користувацького трафіку.

Вибір цих технологій є стратегічною, оскільки вони не тільки відповідають функціональним вимогам проекту, а й сприяють швидкій розробці, надійності, та простоті майбутнього супроводу системи обліку персонального бюджету.

3 ПРОЕКТУВАННЯ WEB-ОРІЄНТОВАНОЇ СИСТЕМИ

Після аналізу сфери дослідження, визначення актуальності, а також після формулювання мети та задач для реалізації проекту, наступним етапом є проектування web орієнтованої системи.

На цьому етапі ми створимо діаграми за допомогою нотації IDEF0 та діаграму Use Case, а також детально опишемо послідовність процесів, які взяли участь у розробці системи.

3.1 Структурно-функціональне моделювання роботи додатку

Методологія моделювання за допомогою діаграми IDEF0 (Integration Definition for Function Modeling) використовується для створення структурно-функціональних моделей системи. Ця методологія дозволяє візуалізувати ключові функції, які виконуються в системі, а також їх взаємодію. Діаграма IDEF0 є ефективним інструментом для визначення об'єктів, процесів та взаємозв'язків між ними, що є важливим для розуміння функціональності системи та ідентифікації ключових етапів роботи [29].

У випадку нашого проекту, діаграма IDEF0 це ідеальний інструмент для відображення ключових функцій та їх взаємозв'язків у веб-орієнтованій системі обліку персонального бюджету. Основні функції системи, такі як аутентифікація та авторизація користувачів, управління витратами, категоризація витрат, статистика та графіки, можна відобразити за допомогою блоків та стрілок на діаграмі IDEF0.

Важливою перевагою діаграми IDEF0 є те, що вона дозволяє нам чітко визначити та вказати на послідовність функцій та їх взаємозв'язки, що сприяє кращому розумінню структури системи для всіх учасників проекту. Така модель

може служити основою для подальшого проектування, реалізації та аналізу системи обліку бюджету.

Процес "Облік персонального бюджету" містить дані, які визначають його функціонування:

1. **Вхідні дані:** Це початкові відомості, які ініціюють процес. У цьому випадку - "Потреба вести персональний бюджет". Це може бути ініціатива користувача та/або його намір активно вести облік своїх фінансів.
2. **Вихідні дані:** Це результати, які отримуємо в результаті процесу. У даному випадку - "Статистика відображення витрат". Система видає користувачеві статистичні дані, які ілюструють його витрати та фінансову активність.
3. **Управління:** Це дані, які визначають, як система повинна функціонувати та які є вимогами до її роботи. Тут вони представлені як "Вимоги до функціонування системи". Це включає в себе всі функціональні вимоги до системи та інші аспекти.
4. **Механізми:** Це виконавчі чинники або елементи, які забезпечують реалізацію процесу. У даному випадку - "Користувач, Технічне забезпечення, Web-орієнтована система". Користувач взаємодіє з системою, технічне забезпечення та сама система забезпечують можливості для реалізації задачі.

Ця концептуалізація відображає всі необхідні компоненти та їх взаємодію в процесі обліку персонального бюджету та зображена на рисунку 3.1.

Рисунок 3.1 – Контекстна діаграма IDEF0

Джерело: побудовано автором

Наступним кроком зробимо декомпозицію першого рівня діаграми IDEF0 веб-додатка "Web-орієнтована система обліку персонального бюджету". Вона зображена на рисунку – 3.2 і включає п’ять основних процесів, які деталізують реалізацію системи:

1. **Автентифікація користувача:**

Вхідні дані: Потреба вести персональний бюджет.

Вихідні дані: Система підтверджує або відхиляє автентифікацію користувача.

2. **Введення даних про витрати:**

Вхідні дані: Дані про користувача після автентифікації.

Вихідні дані: Введені користувачем дані про витрати.

3. **Збереження даних про витрати:**

Вхідні дані: Введені користувачем дані про витрати.

Вихідні дані: Система зберігає введені дані для подальшого використання та аналізу.

4. **Обчислення статистики:**

Вхідні дані: Збережені дані про витрати.

Вихідні дані: Обчислені дані, сума витрат по кожній категорії.

5. **Відображення витрат на діаграмі:**

Вхідні дані: Обчислені дані по категоріям.

Вихідні дані: Статистика відображення витрат у вигляді кільцевої діаграми.

Під час декомпозиції реалізації системи враховані вхідні дані, вихідні дані, управління та механізми, що включають Користувача, Технічне забезпечення та Web-орієнтовану систему. Кожен з цих процесів виконує конкретні завдання, спрямовані на досягнення загальної мети – ефективного обліку персонального бюджету.

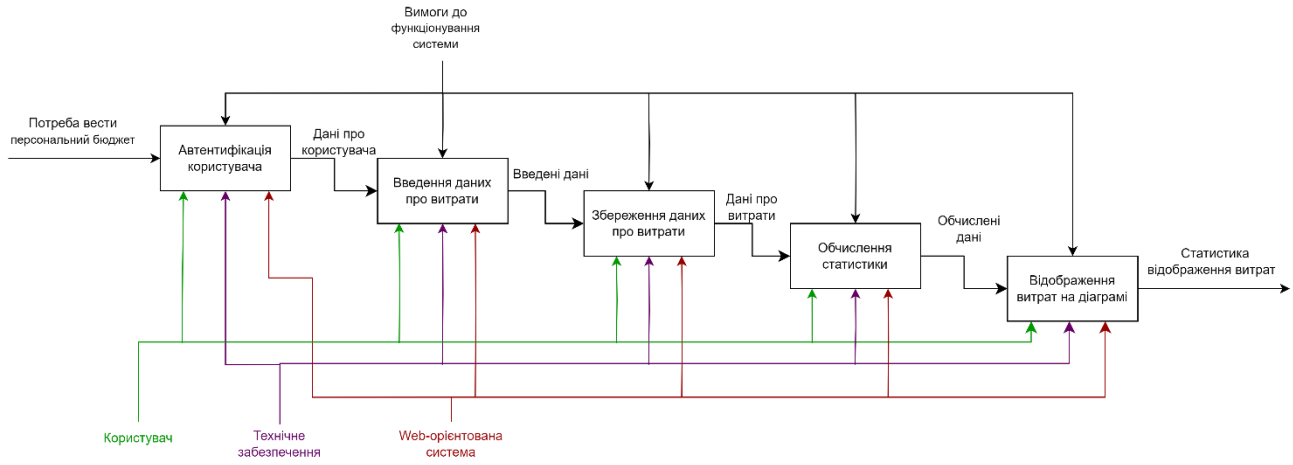

Рисунок 3.2 – Діаграма декомпозиції

Джерело: побудовано автором

Ця деталізована декомпозиція взаємозв'язків показує, як кожна частина системи взаємодіє з іншими для досягнення загальної мети.

3.2 Моделювання варіантів використання

Діаграма Use Case є графічним зображенням взаємодії між акторами (користувачами) та системою у контексті конкретного функціоналу чи сценарію. Основна мета - визначити, які операції можуть бути виконані акторами в системі та як система реагує на їхні запити. [36]

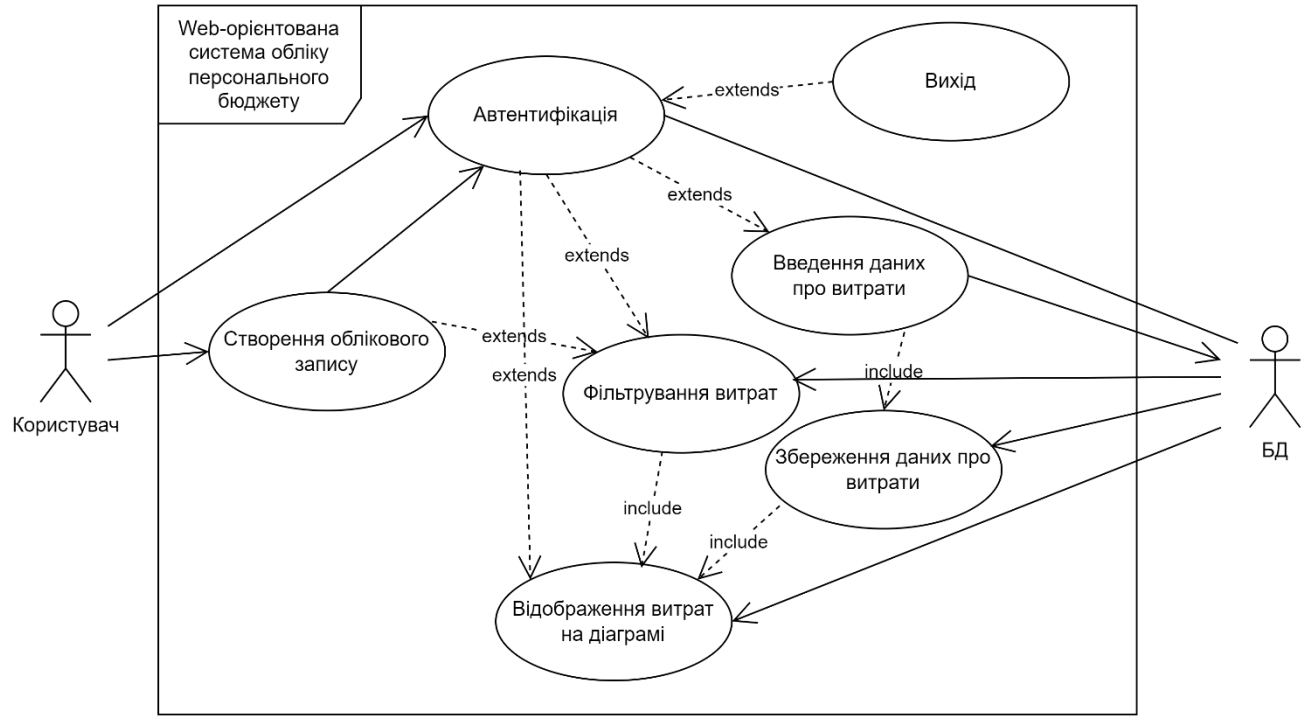

Рисунок 3.3 – Діаграма Use Case

Джерело: побудовано автором

Розглянемо більш детально взаємодію користувача з веб-додатком "Web-орієнтована система обліку персонального бюджету" зображені на рисунку 3.3:

1. **Створення облікового запису:**

Опис: Якщо у користувача немає облікового запису, він може перейти на сторінку реєстрації для створення нового облікового запису.

Варіанти:

- *Перехід на сторінку реєстрації*: Користувач обирає опцію "Зареєструватися" та переходить на сторінку реєстрації.

- *Введення даних для реєстрації*: Користувач вводить необхідні дані для створення нового облікового запису.

- *Завершення реєстрації:* Система підтверджує успішне створення облікового запису для користувача.

2. **Автентифікація:**

Опис: Користувач, який вже має обліковий запис, вводить свої дані для входу в систему.

Варіанти:

- *Введення існуючих даних:* Користувач вводить свій логін та пароль для автентифікації.

- *Невдала автентифікація:* Користувач вводить неправильні дані, та отримує повідомлення про помилку.

3. **Фільтрування витрат:**

Опис: Після вдалої автентифікації або створення облікового запису, користувач може відфільтрувати існуючи дані, що були раніше додані до веб-додатка.

Варіанти:

- *Огляд існуючих категорій, валют, дат:* Користувач може відфільтрувати дані для побудови статистики витрат.

4. **Введення даних про витрати:**

Опис: Користувач має можливість додавати дані про свої витрати.

Варіанти:

- *Вибір категорії:* Користувач обирає категорію витрат (харчування, транспорт, розваги і т.д.).

- *Вибір валюти:* Користувач обирає валюту (UAH, USD, EUR).

- *Вибір дати:* Користувач обирає дату за яку відбулись витрати.

- *Введення суми:* Користувач вказує суму витрат.

5. **Збереження даних про витрати:**

Опис: Користувач може зберегти введені дані про витрати для подальшого аналізу.

Варіанти:

- *Підтвердження збереження:* Після введення даних користувач підтверджує їх збереження натисненням кнопки «Зберегти».
- *Перевірка валідації полів:* Система перевіряє введені данні на помилки та в разі їх відсутності зберігає данні до бази даних. Якщо помилки знайдені, користувачу відображається повідомлення про необхідність їх усунення для збереження даних.

6. **Обчислення статистики:**

Опис: Система автоматично обчислює та відображає статистичну інформацію про витрати користувача.

Варіанти:

- *Обчислення:* Додаються видатки по кожній категорії за обраний період.

7. **Відображення витрат на діаграмі:**

Опис: Система надає користувачеві можливість відображення своїх витрат у вигляді кільцевої діаграми для зручного аналізу.

Варіанти:

- *Вибір даних:* Користувач може вибрати дату за який бажає переглянути статистику витрат, а також валюту.
- *Відображення різних категорій*: Графічне представлення витрат за різними категоріями за допомогою діаграми.

8. **Вихід:**

Опис: Система надає користувачеві можливість вийти і завершити сеанс.

Варіанти:

- *Натискання кнопки:* Користувач при натисканні на кнопку «Вийти» завершує сеанс та автоматично перенаправляється на сторінку входу в систему для введення даних для авторизації.

3.3 Проектування бази даних

Діаграма взаємозв'язків сутностей (ERD) [37] є графічним інструментом, який відображає зв'язки між об'єктами, поняттями чи подіями в інформаційній системі. ERD використовується для моделювання даних і визначення зв'язків між ними, що може служити основою для створення реляційної бази даних.

ER-діаграми служать візуальною відправною точкою для проектування бази даних, візуалізують взаємозв'язки між таблицями та визначають вимоги до інформаційних систем в організації. Нижче представлена ER-діаграма для бази даних системи обліку персонального бюджету:

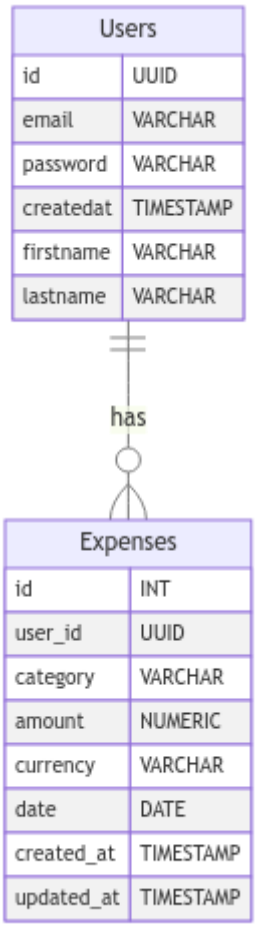

Рисунок 3.4 – ER-діаграма бази даних

Джерело: побудовано автором

Ця ER-діаграма відображає зв'язок між користувачем та його витратами, дозволяючи системі відстежувати та організовувати витрати кожного користувача.

Розглянемо більш детальний розбір ER-діаграми:

1. **Таблиця "Users":**

Сутність "Users" відображає інформацію про користувачів системи.

Атрибути:

- *id* (Ключ): Унікальний ідентифікатор користувача.
- *email*: Адреса електронної пошти користувача для ідентифікації.
- *password*: Пароль користувача для забезпечення безпеки.
- *createdat*: Дата та час створення запису користувача.
- *firstname*: Ім'я користувача.
- *lastname*: Прізвище користувача.

2. **Таблиця "Expenses":**

Сутність "Expenses" відображає витрати, пов'язані з кожним користувачем.

Атрибути:

- *id* (Ключ): Унікальний ідентифікатор витрати.
- *user_id* (Зовнішній ключ): Зовнішній ключ, який посилається на поле id таблиці users, вказуючи, якому користувачеві належать витрати.
- *category*: Категорія витрати (наприклад, їжа, транспорт, розваги тощо).
- *amount*: Сума витрати.
- *currency*: Валюта витрати (за замовчуванням 'UAH').
- *date*: Дата витрати.
- *created_at*: Дата та час створення запису витрати.
- *updated_at*: Дата та час останнього оновлення запису витрати.

4 ПРАКТИЧНА РЕАЛІЗАЦІЯ WEB-ОРІЄНТОВАНОЇ СИСТЕМИ

4.1 Архітектура web-додатку

Архітектура веб-додатку визначає структуру та організацію компонентів та модулів програмного забезпечення, встановлюючи їхню взаємодію для досягнення конкретної функціональності [38].

Архітектура web-додатку обліку персонального бюджету представлена на рисунку 4.1 та виглядає наступним чином:

Frontend: містить окремі компоненти побудовані за допомогою React JS фреймворка, що відповідають за форми додавання та редагування витрат, вивід статистики та виведення списку витрат. Бібліотека Material UI надає змогу в використанні вже готових функціональних компонентів (модальні вікна, елементи форми, тощо), а також можливість стилізувати web-додаток в цілому. Бібліотека Chart.js незамінна в виводі статистики в вигляді кільцевої діаграми.

Backend: написаний з архітектурним стилем REST API [39] з використанням Express JS фреймворку, для обробки запитів від клієнта. Містить ендпойнти для автентифікації користувача та CRUD [39] операцій з витратами та middleware для перевірки JWT-токена при кожному запиті, щоб забезпечити автентифікацію користувача.

Забезпечено підключення до **бази даних** PostgreSQL, використовуючи пакет node-postgres [40] для роботи з базою даних. Структура бази даних наведена на рисунку 3.4

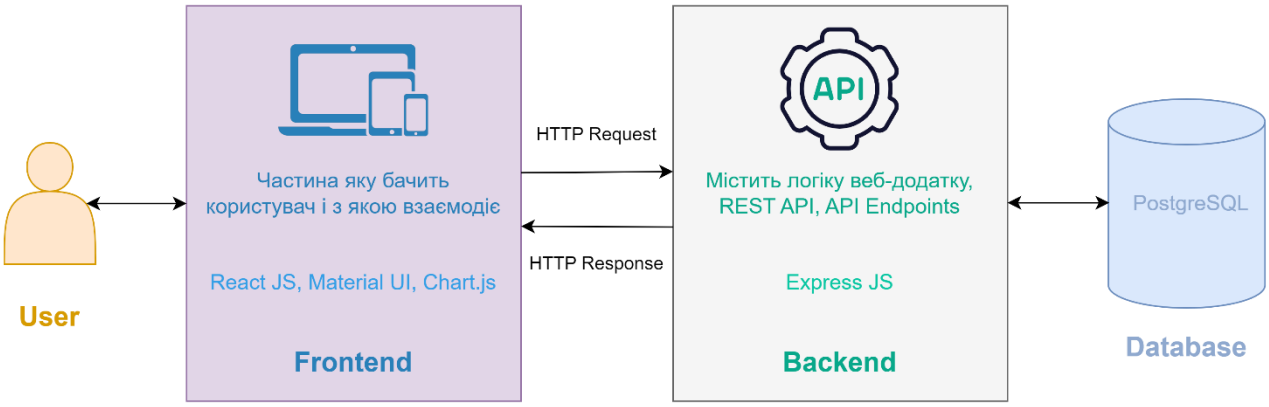

Рисунок 4.1 – Архітектура web-додатку

Джерело: побудовано автором

4.2 Програмна реалізація

4.2.1 Налаштування початку роботи

Для початку роботи над веб-додатком, необхідно спочатку встановити на локальну машину Node.js, що є міжплатформовим середовищем виконання JavaScript із відкритим кодом [41].

Після встановлення Node.js, наступним кроком буде створення структури проекту, яка оптимально відображає логічну організацію різних компонентів. Рисунок 4.2 відображає приклад такої структури проекту.

Після створення структури проекту, слід налаштувати доступ та підключити базу даних, в нашоми випадку це PostgreSQL. Це включає в себе встановлення та конфігурацію бази даних для забезпечення ефективного зберігання і отримання даних для веб-додатку.

Далі важливим кроком є налаштування підключення до системи контролю версій, такої як GitHub [42]. Це забезпечить ефективний контроль версій і спільну роботу над проектом для розробників. Інтеграція з GitHub дозволяє зберігати, відслідковувати та об'єднувати зміни в коді між різними розробниками та гілками розробки.

Загальний процес встановлення, налаштування та початку роботи над веб-додатком охоплює ці кроки, забезпечуючи системний та структурний фундамент для подальшого розвитку та управління проектом.

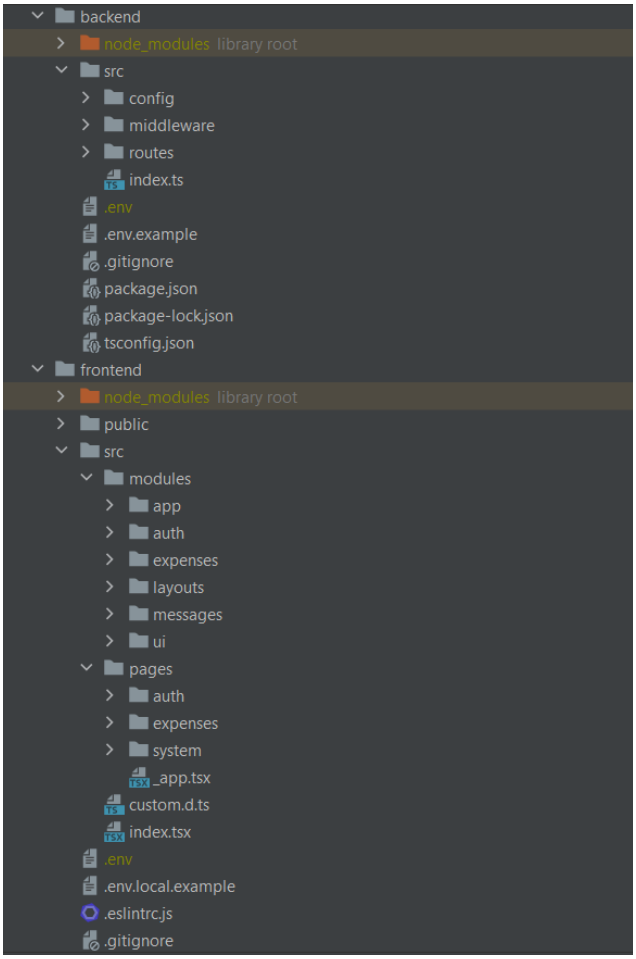

Рисунок 4.2 – Структура web-додатку

Джерело: побудовано автором

4.2.2 Реалізація backend частини

Для початку, розглянемо важливі аспекти реалізації автентифікації та авторизації веб-додатку з точки зору користувача та архітектури.

Спроектовано та реалізовано ендпойнт (ключові елементи) для реєстрації, який приймає від клієнта особисті дані (email, пароль, ім’я та прізвище). Реалізація **auth** ендпойнту (реєстрація та автентифікація користувача) наведено в Додатку Б.

Передбачено перевірку унікальності електронної пошти та інших обов'язкових полів. У разі успішної реєстрації автоматично створюється новий запис про користувача в базі даних(код наведено у Додатку Б) .

Розроблено та імплементовано ендпойнт для логіну, який приймає від клієнта дані для автентифікації (email та пароль). Забезпечено перевірку правильності введених даних.

При успішному вході створюється та відправляється JWT токен, який містить ідентифікатор користувача та іншу інформацію.

Створено middleware для верифікації JWT токена (захист від несанкціонованого доступу та перевірка ідентифікації користувача при кожному запиті). Цей middleware перевіряє правильність підпису та час життя токена. Реалізація **middleware** для верифікації JWT-токена наведено в ДодаткуБ.

Якщо токен валідний, інформація з нього додається до об'єкта запиту, що дозволяє ідентифікувати користувача.

Для роботи з даними витрат та проведення всіх CRUD операцій було створено «/expense» ендпойнт. Розглянемо основні його можливості:

- *Створення витрати (Create Expense):* забезпечує можливість користувачам додавати нові витрати.

Створено REST ендпойнт для обробки POST-запитів із даними нової витрати. Витрата додається до бази даних.

- *Отримання витрат (Read Expenses):* надає користувачам можливість переглядати список своїх витрат.

Створено REST ендпойнт для обробки GET-запитів, який повертає список витрат конкретного користувача.

- *Оновлення витрати (Update Expense):* дозволяє користувачам змінювати дані про конкретну витрату.

Створено REST ендпойнт для обробки PUT-запитів із новими даними витрати. Витрата оновлюється в базі даних.

- *Видалення Витрати (Delete Expense):* надає користувачам можливість видаляти зайві або помилкові витрати.

Створено REST ендпойнт для обробки DELETE-запитів із ідентифікатором витрати. Витрата видаляється з бази даних.

Код реалізації backend частини з вказанням взаємозв'язків і запитів до бази для всіх перелічених вище ендпойнтів надано в Додатку Б.

4.2.3 Реалізація frontend частини

Реалізацію почнемо зі створення механізму автентифікації користувача. Для цього створимо контекст AuthContext та AuthProvider для зручного передавання токену через додаток. Використаємо також локальне сховище (localStorage) для зберігання токену між сесіями.

```
export const AuthContext : React.Context<AuthContextValue… = createContext<AuthContextValue | undefined>( defaultValue: undefined);

3 usages  pdenisenko
export const AuthProvider = ({ children }: AuthContextProps) => {
  const [token : Token , setToken : React.Dispatch<React.SetStateA… ] = useState<Token>(localStorage.getItem( key: 'token'));

  const decodedToken : "" | JwtPayload | null = token && jwtDecode(token);

  useEffect( effect: () : void => {
    if (decodedToken && decodedToken.exp) {
      if (decodedToken.exp < currentTimestamp) {
        setToken( value: null);
      }
    }
  }, deps: [decodedToken]);

  const saveToken = useCallback( callback: (token: Token) : void => {
    setToken(token);

    if (token) {
      localStorage.setItem('token', token);
    } else {
      localStorage.removeItem( key: 'token');
    }
  }, deps: []);

  return (
    <AuthContext.Provider value={{ token, setToken: saveToken }}>{children}</AuthContext.Provider>
  );
};
```

Рисунок 4.3 – Створення **AuthProvider**

Джерело: побудовано автором

Після цього було створено компоненти для форми входу та форми реєстрації з використанням React Hook Form [43] для зручного управління станом форм та валідації даних (код наведено у Додатку Б).

Дали написали інтеграцію з backend частиною, для цього використано Axios [44] для здійснення HTTP-запитів до Auth ендпойнтів на backend (реєстрація, вхід). Реалізували логіку взаємодії з сервером для входу та реєстрації, передаючи необхідні дані.

Додали код для валідації обов'язкових полів та правильного формату даних. Реалізували обробку помилок та відображення відповідних повідомлень користувачеві.

Аналогічна реалізація проводилась і для форм створення та редагування видатків. Здійснили взаємодію з backend за допомогою Axios для відправлення HTTP-запитів до backend ендпойнтів, зокрема ендпойнтів для створення та редагування витрат. Налаштували відповідні запити для передачі необхідних даних для отримання попередніх даних про витрату. Забезпечили можливість редагування витрат, вивівши попередні дані у форму для їх зручного оновлення.

Наступним кроком додали валідацію для обов'язкових полів та правильного формату даних. Реалізували обробку помилок та відображення відповідних повідомлень користувачеві.

Створили компонент форми для фільтрації витрат за різними параметрами фільтрації, такими як: валюта, категорія, початкова та кінцева дата. Забезпечили можливість відміни фільтрації та повернення до повного списку витрат. Розробили кнопку і функціонал для скидання введених параметрів фільтрації.

Реалізовані функції фільтрації, що наведені в Додатку Б, дозволяють динамічно фільтрувати витрати на основі різних критеріїв та за змовчуванням відображати дані для валюти UAH та витрат поточного місяця.

Після цього створили компонент для відображення списку витрат. Для цього використано дані про витрати, що враховують фільтрацію. Кожен елемент списку представляє окрему витрату з можливістю редагування та видалення. Застосували

Material UI для стилізації та використали компоненти інтерфейсу, такі як: список, іконки, тощо.

Компонент для відображення списку витрат має можливість виводити інформацію про кожну витрату, таку як категорія, сума, валюта та інше. Для цього реалізовано кнопки для редагування та видалення кожної витрати.

Створено компонент для кільцевої діаграми. Для цього було використано Chart.js, бібліотеку для візуалізації даних у вигляді графіків та діаграм, в яку передаються дані про витрати для обчислення сум витрат за кожною категорією. Кожна сума витрат за категорію обчислюється на основі вхідних даних, отриманих з backend через запити на ендпойнти.

Після цього перешли до створення кільцевої діаграми, де кожен сектор відповідає певній категорії, а розмір сектора відображає загальну суму витрат у цій категорії. Застосовано кольорову кодировку для кращого розуміння категорій. Додано можливість натискання на сектор діаграми для отримання більш детальної інформації про витрати у вибраній категорії у вигляді тултіпу з назвою категорії і загальною сумою витрат за обраний період.

Всі ці перелічені компоненти дозволяють користувачу зручно переглядати та аналізувати свої витрати як у вигляді списку, так і у вигляді графічної діаграми.

Також, слід зазначити, про проведення загальної стилізації з використанням бібліотеки Material UI. Використано готові компоненти задля поліпшення зовнішнього вигляду. Налаштовані тема для загальної стилізації компонентів, сторінок та отримання привабливого та зручного дизайну. Реалізовано респонсів поведінка додатку для кращого доступу користувачами з різних пристроїв (телефон, планшет, екран монітору).

Код реалізації frontend частини основних процесів, що беруть участь у функціонуванні web-системи наведено в Додатку Б.

4.3 Демонстрація роботи

Розглянемо роботу web-орієнтована система обліку персонального бюджету з точки зору користувача.

Для початку користувачу буде запропоновано ввійти в систему, заповнивши необхідні поля як показано на рисунку 4.3

Якщо користувач незареєстрований, то він переходить на сторінку реєстрації (рисунок 4.4) і після успішної реєстрації (система повідомить про це в вигляді сповіщення) , користувач буде перенаправлений знову на сторінку входу для вводу даних для авторизації в системі.

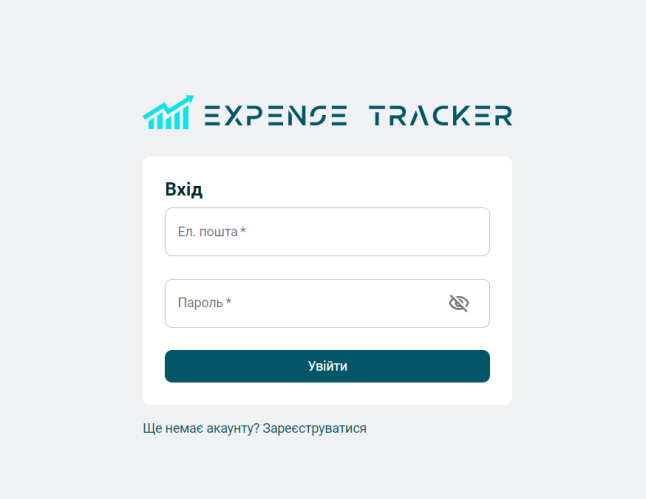

Рисунок 4.4 – Форма входу в систему

Джерело: побудовано автором

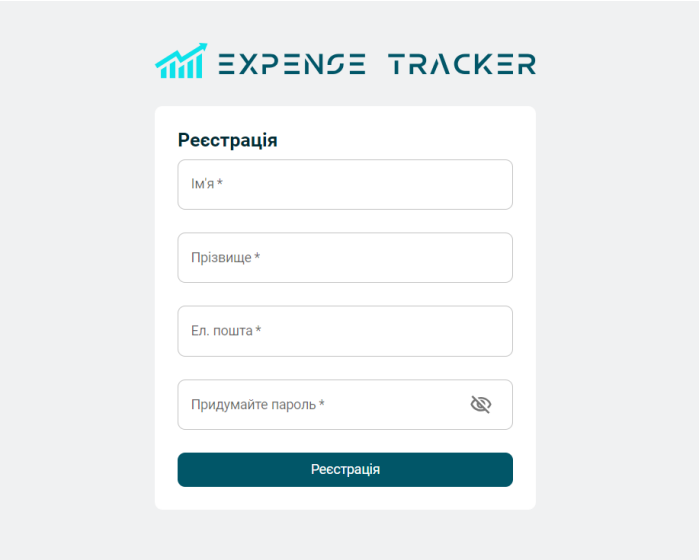

Рисунок 4.5 – Форма реєстрації користувача

Джерело: побудовано автором

Після успішної авторизації, користувач потрапляє на головну сторінку web-орієнтованої системи (рисунок 4.5) для керування персональними витратами.

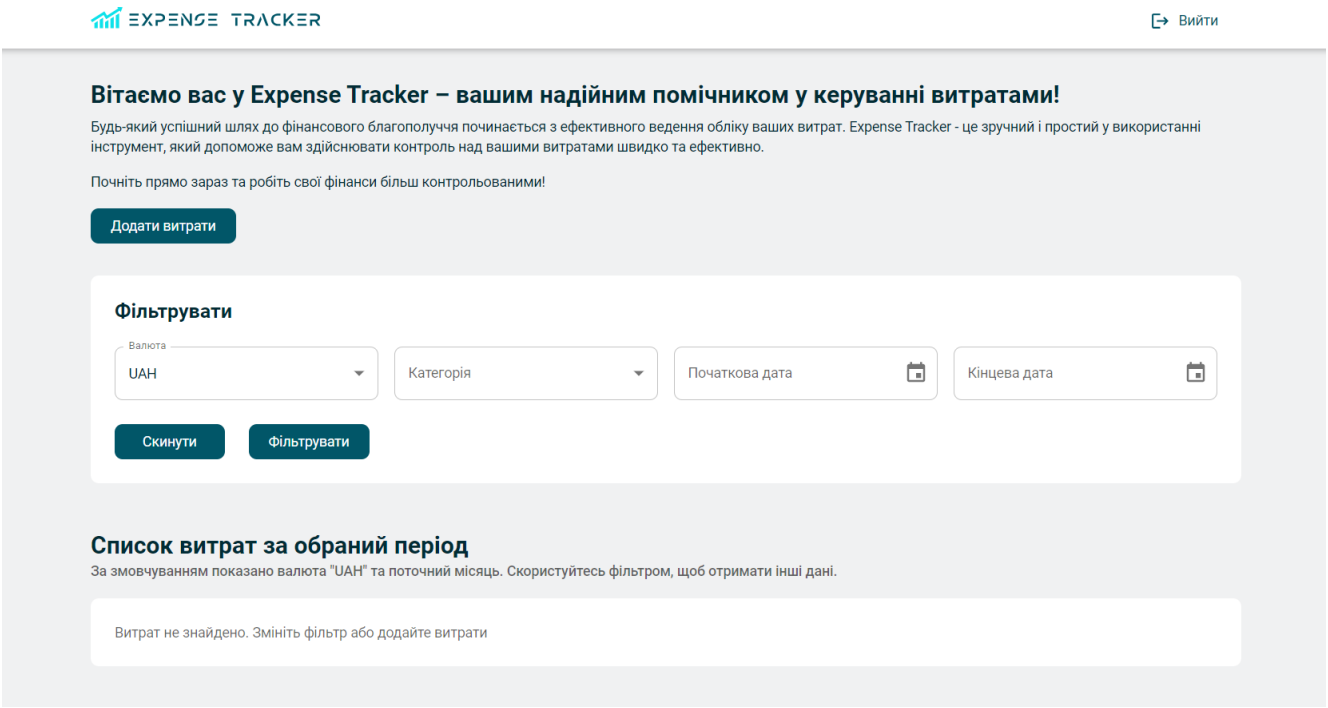

Рисунок 4.6 – Головна сторінка

Джерело: побудовано автором

З головної сторінки, користувачу, надається можливість вводу витрат, їх фільтрації та переглядання статистики за обраний період, тощо.

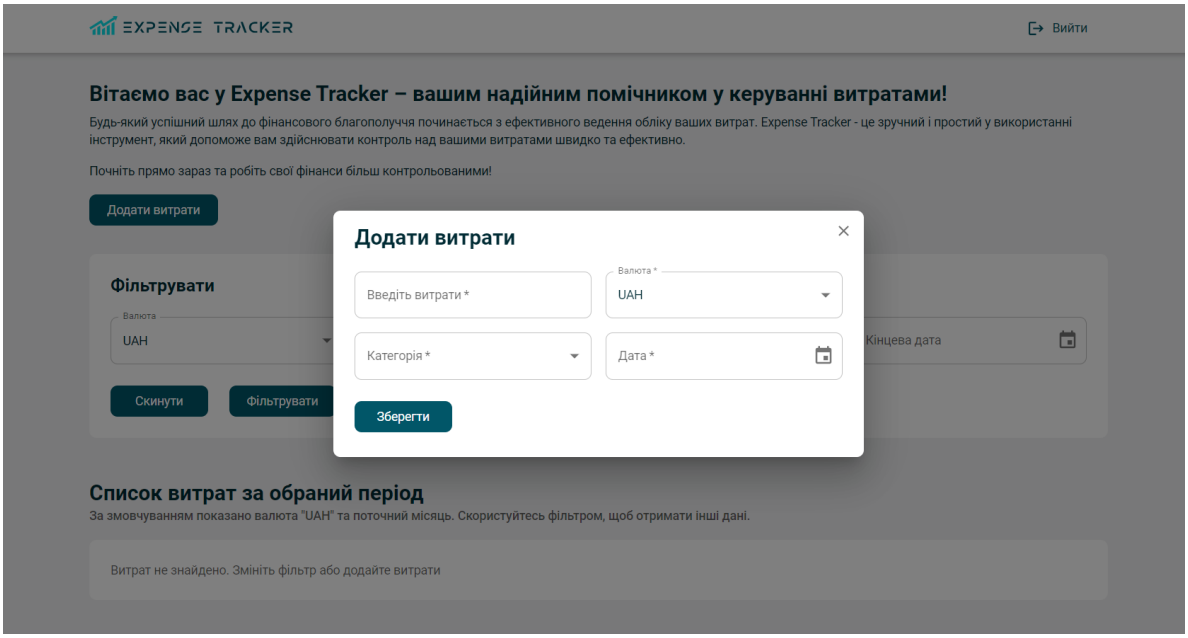

Рисунок 4.7 – Форма додавання витрат

Джерело: побудовано автором

Додавання витрат відбувається в модальному вікні (рисунок 4.6), попередньо натиснувши на кнопку «Додати витрати»

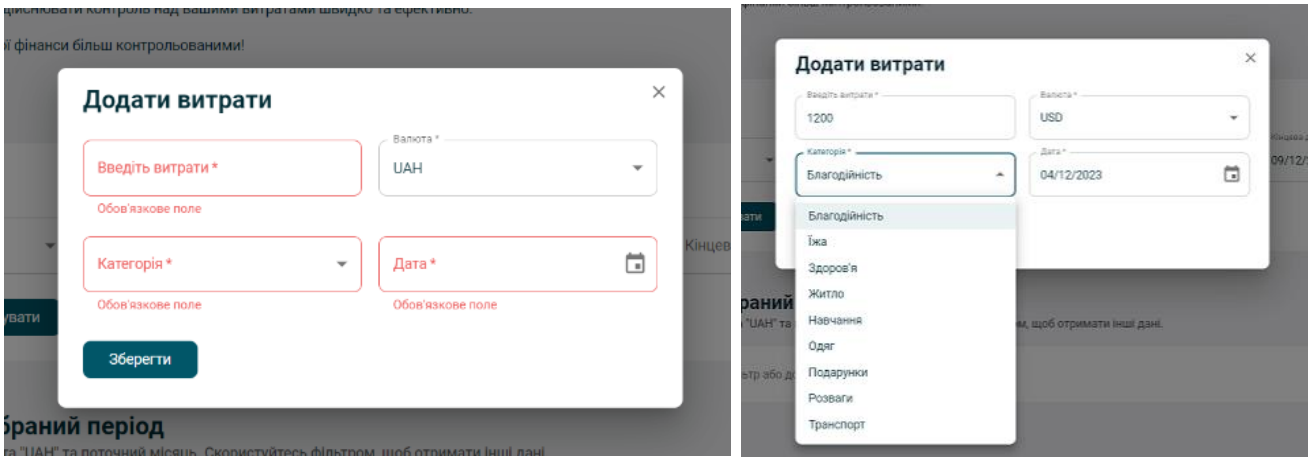

Рисунок 4.8 – Валідація даних додавання витрат

Джерело: побудовано автором

Під час додавання витрат, система перевірить дані на валідність та обов’язковість (рисунок 4.7), та при успішному процесу збереже введені дані, показавши повідомлення в правому верхньому куті екрану (рисунок 4.8), та очистивши поля форми «Додати витрати» для можливості наступного додавання даних.

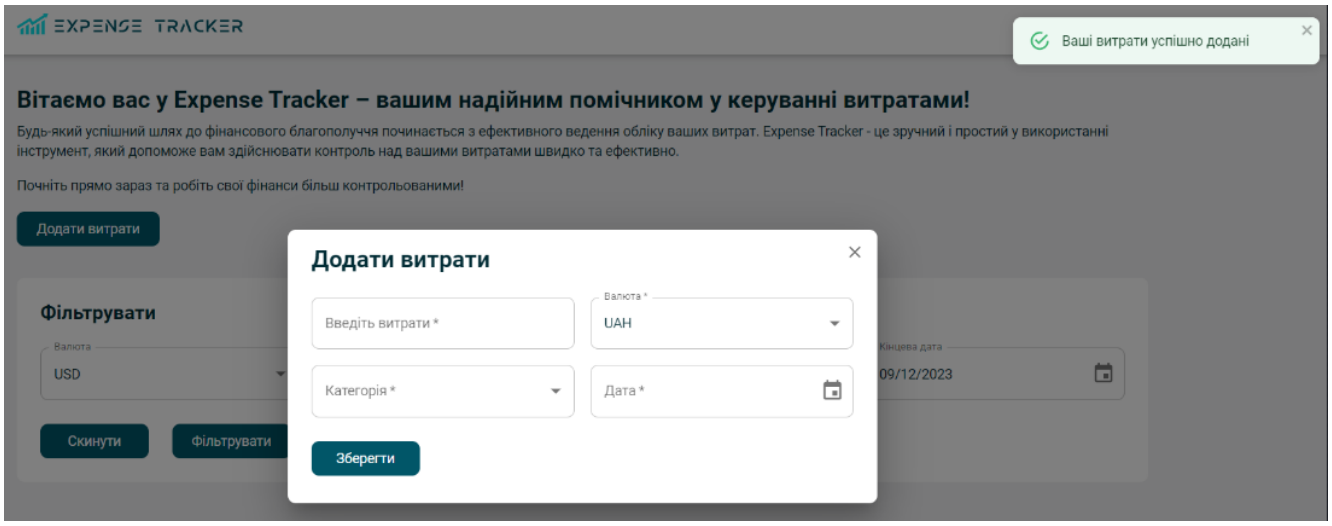

Рисунок 4.9 – Успішне додавання витрат

Джерело: побудовано автором

Статистика по витратам відображається як кільцевою діаграмою, з сумою по кожній категорії, так і за допомогою списку (рисунок 4.9)

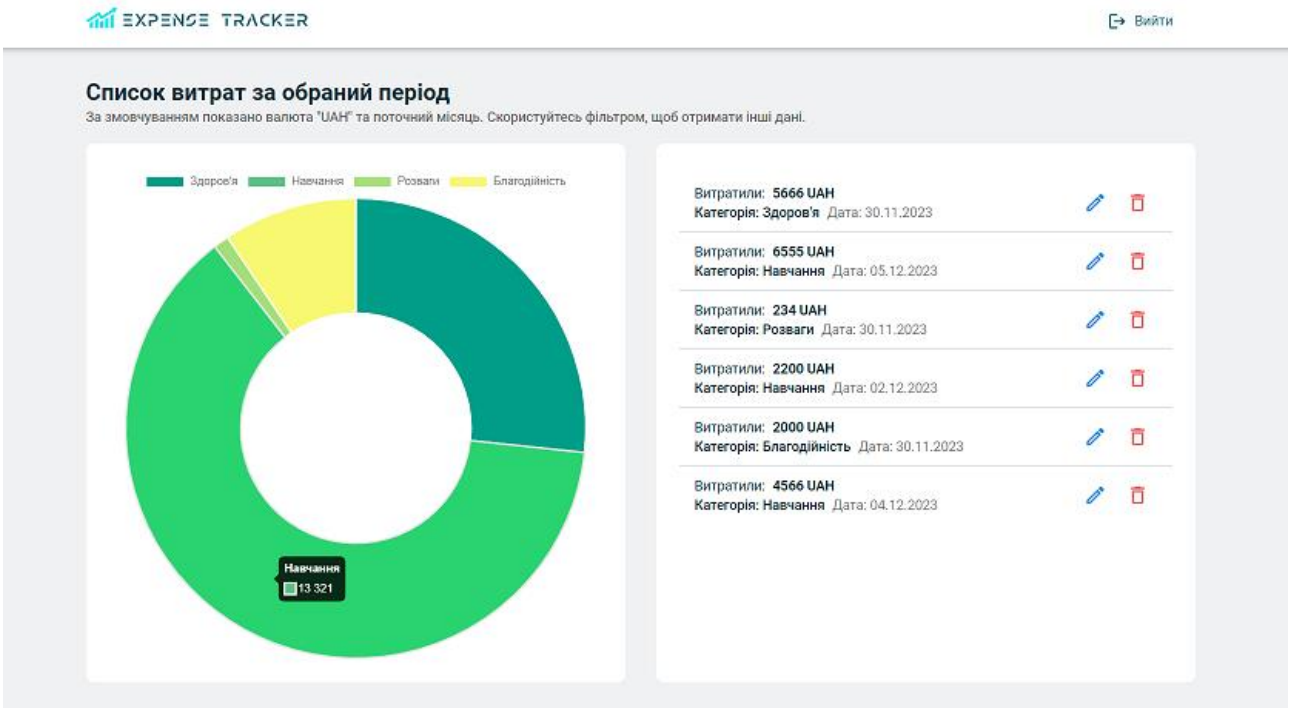

Рисунок 4.10 – Відображення статистики по витратам

Джерело: побудовано автором

Дані, що відображаються в статистиці, можна відфільтрувати за допомогою форми фільтрації (рисунок 4.10) обравши потрібний параметр (валюта, категорія та дата), так і за допомогою легенди кільцевої діаграми прибравши непотрібні категорії, що показано на рисунку 4.11.

Зі списку витрат, який знаходиться з права від діаграми, можна відредагувати потрібні значення по кожному запису. Для цього необхідно натиснути на іконку редагування та змінити дані в вікні редагування (рисунок 4.12). Також з загального списку витрат можна видалити потрібний запис, натиснувши на іконку видалення.

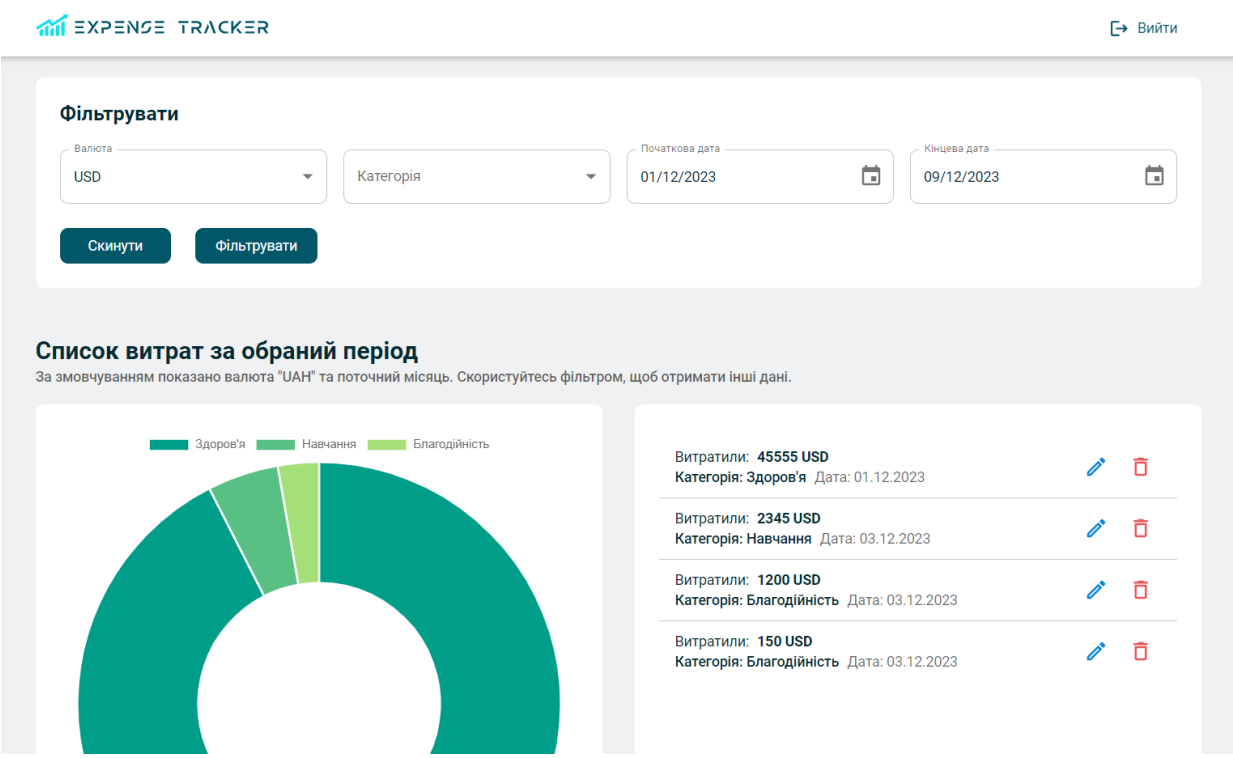

Рисунок 4.11 – Форма фільтрації даних для відображення статистики

Джерело: побудовано автором

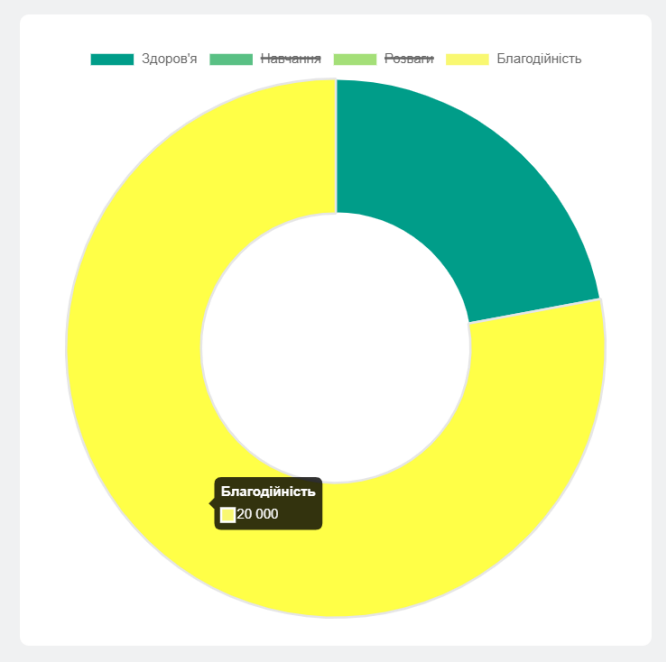

Рисунок 4.12 – Фільтрація видатків за допомогою легенди діаграми

Джерело: побудовано автором

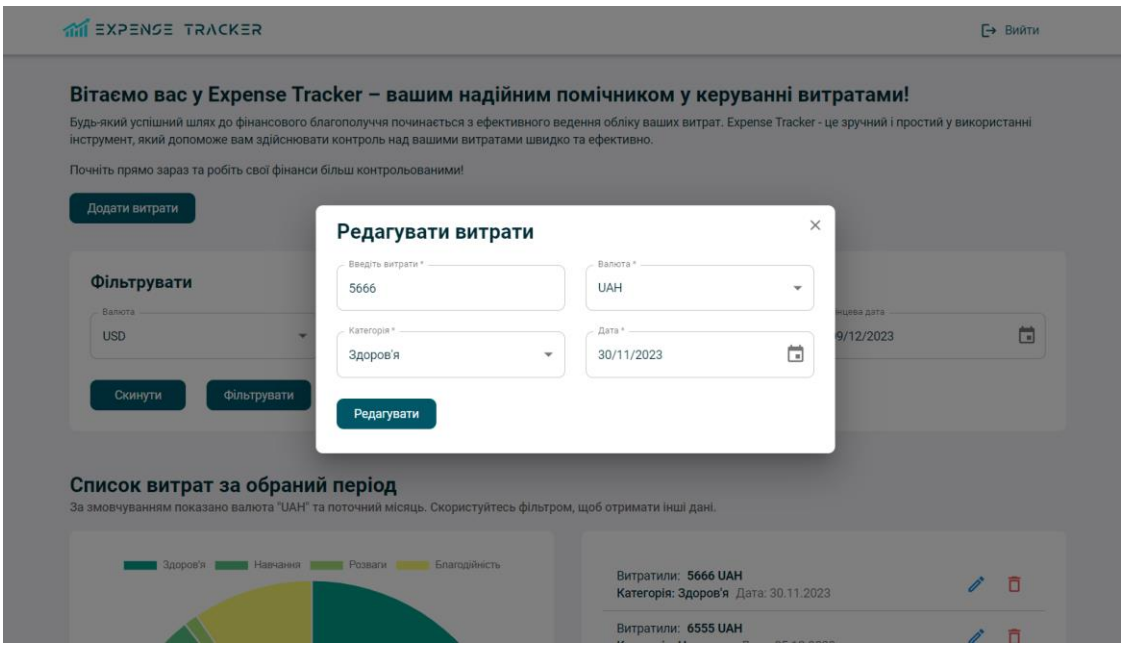

Рисунок 4.13 – Форма редагування витрат

Джерело: побудовано автором

На рисунку 4.13 представлено як виглядає наша системи з різних де девайсів телефон та планшет, підкреслюючи її доступність для користувача з різних платформ.

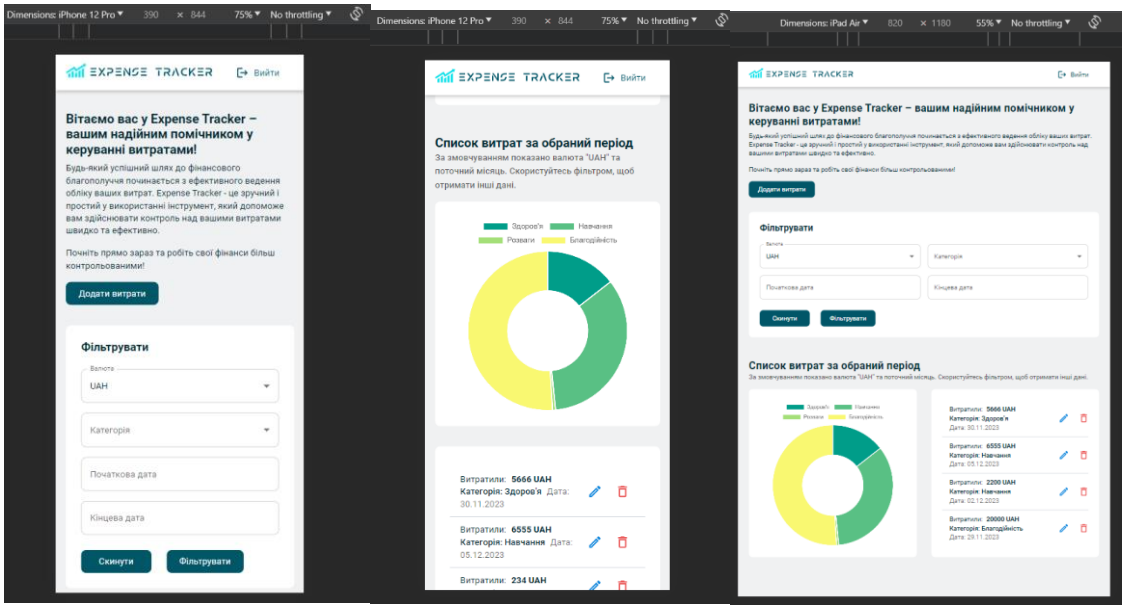

Рисунок 4.14 – Вид web-додатку на різних пристроях (mobile, tablet)

Джерело: побудовано автором

4.4 Тестування web-додатку

Щоб перевірити коректність роботи системи необхідно було провести тестування.

Існує кілька видів тестувань, які використовуються для різних цілей та на різних етапах розробки програмного забезпечення. Основні види тестувань включають: модульне тестування (Unit Testing), інтеграційне тестування (Integration Testing) і т.н.

Тестування, що проводиться користувачем або представником клієнта. Оцінка відповідності продукту визначеним вимогам та його придатності для використання в реальних умовах (функціональне, нефункціональне, тестування відмовостійкості, тощо). [45]

Ми будем використовувати функціональне тестування – це вид тестування програмного забезпечення, спрямований на перевірку того, чи виконує система свої функції згідно з визначеними вимогами та очікуваннями користувачів.

Сценарії тестування:

- Вхід/реєстрація користувача.
- Створення, редагування витрат.
- Фільтрація витрат за допомогою форми.
- Відображення списку витрат та кільцевої діаграми.

Тестові дані:

- Створені тестові користувачі для реєстрації.
- Заповнені витрати для тестування списку та діаграми.

За результатами тестувань, було сформовано таблицю 4.1, де відображено результати.

Таблиця 4.1 Тестування функціоналу web-системи

Протестовано	**Очікуваний результат**	**Фактичний результат**	**0/1**
Реєстрація користувача	Успішна реєстрація, перенаправлення на сторінку входу	Користувач був перенаправлений після реєстрації на сторінку входу	1
Вхід користувача	Успішний вхід, перенаправлення на головну сторінку	Після коректно введених даних, перенаправлення на головну сторінку	1
Створення витрат	Витрати створюються, користувач бачить повідомлення про успіх	Після створення витрат, повідомлення про успіх	1
Редагування витрат	Зміни витрат відображаються в списку	Зміни витрат відображаються в списку	1
Фільтрація витрат за допомогою форми	Відображення витрат, що відповідають обраним параметрам	Витрати відобразились згідно обраним параметрам	1
Відображення списку витрат	Список витрат відображається коректно	Список витрат відображається коректно	1
Кільцева діаграма	Діаграма відображає суми витрат за категоріями	Діаграма відображає суми витрат за категоріями	1

Тестовий репорт демонструє, що всі складові Web-орієнтованої системи пройшли успішне функціональне тестування. Критичних багів не виявлено, всі функції працюють правильно і відповідають очікуванням користувача.

ВИСНОВКИ

В ході виконання кваліфікаційної роботи магістра з розробки web-орієнтованої системи обліку персонального бюджету були визначені ключові аспекти та етапи, які необхідні для успішної реалізації проекту. У рамках розробки проекту було виконано всі поставлені завдання на кожному етапі:

- Проведений аналіз проблем, пов'язаних із управлінням персональним бюджетом, що надало підставу для розробки ефективного рішення.
- Проведено дослідження сучасних публікацій та досліджень для підтвердження актуальності теми та визначення потреб цільової аудиторії.
- Вивчено функціонал та особливості існуючих веб-орієнтованих систем обліку бюджету та визначено переваги та недоліки конкурентних рішень, що сприятиме формуванню конкурентоспроможного продукту.
- Проведено дослідження технологій для розробки веб-систем та обрано оптимальний стек технологій, враховуючи вимоги проекту, швидкість розробки та масштабованість.
- Створено детальну модель системи, включаючи базу даних, логічну та фізичну архітектуру.
- Розроблено структуру інтерфейсу користувача для забезпечення зручності взаємодії з системою.
- Реалізовано серверну та клієнтську частини веб-системи відповідно до розробленої моделі та забезпечено основні функціональні можливості, які включають ведення обліку, аналіз витрат, генерацію статистики та інші.
- Проведено тестування системи для перевірки відповідності функціоналу вимогам та виявлення можливих помилок.

В результаті виконання всіх цих етапів отримано функціональну та ефективну веб-систему обліку персонального бюджету, яка готова до подальшого розвитку та впровадження в реальне використання.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Фінансова грамотність населення: зарубіжний досвід і вітчизняні реалії [Електронний ресурс] – Режим доступу до ресурсу: http://dspace.wunu.edu.ua/bitstream/316497/14651/1/%D0%A4%D0%86%D0%9D%20%D0%93%D0%A0%D0%90%D0%9C%D0%9E%D0%A2%20%D0%92%D1%96%D1%81%D0%BD%D0%B8%D0%BA%20%D0%A2%D0%9D%D0%95%D0%A3%20%D0%9A%D1%96%D0%B7%D0%B8%D0%BC%D0%B0%20%D0%A2.pdf

2. Фінансова грамотність, фінансова інклюзія та фінансовий добробут в Україні у 2021. Звіт за результатами дослідження [Електронний ресурс] – Режим доступу до ресурсу: https://bank.gov.ua/admin_uploads/article/Research_Financial_Literacy_Inclusion_Welfare_2021.pdf

3. Національний банк України. За останні три роки рівень фінансової грамотності українців поліпшився – результати дослідження. [Електронний ресурс] – Режим доступу до ресурсу: https://bank.gov.ua/ua/news/all/za-ostanni-tri-roki-riven-finansovoyi-gramotnist-ukrayintsiv--polipshivsya--rezultati-doslidjennya

4. Сімейний бюджет. Як фінансова грамотність населення впливає на фінансовий добробут держави? [Електронний ресурс] – Режим доступу до ресурсу: https://simeinyi-budzhet.ua/familybudget/%D1%8F%D0%BA-%D1%84%D1%96%D0%BD%D0%B0%D0%BD%D1%81%D0%BE%D0%B2%D0%B0-%D0%B3%D1%80%D0%B0%D0%BC%D0%BE%D1%82%D0%BD%D1%96%D1%81%D1%82%D1%8C-%D0%BD%D0%B0%D1%81%D0%B5%D0%BB%D0%B5%D0%BD%D0%BD%D1%8F-%D0%B2/

5. ФГВФО. Сімейний бюджет: як і навіщо його планувати? [Електронний ресурс] – Режим доступу до ресурсу: https://www.fg.gov.ua/articles/48008-simeyniy-byudzhet-yak-i-navishcho-yogo-planuvati.html

6. Фінансова грамотність. Що? Чому? Як? [Електронний ресурс] – Режим доступу до ресурсу: http://www.fst-ua.info/wp-content/uploads/2019/08/Financial_Literacy_Textbook_Aug2019.pdf

7. CNBC Select. These top budgeting apps sync with your bank accounts, are widely available and come highly rated [Електронний ресурс] – Режим доступу до ресурсу: https://www.cnbc.com/select/best-budgeting-apps/

8. YNAB. Change Your Relationship With Money [Електронний ресурс] – Режим доступу до ресурсу: https://www.ynab.com/

9. Mint. Managing money, made simple [Електронний ресурс] – Режим доступу до ресурсу: https://mint.intuit.com/

10. PocketGuard. Always know where your money goes [Електронний ресурс] – Режим доступу до ресурсу: https://pocketguard.com/

11. GoodBudget. Budget with a why Spend, save, and give toward what's important in life [Електронний ресурс] – Режим доступу до ресурсу: https://goodbudget.com/

12. Mozilla Developer Network (MDN) [Електронний ресурс] – Режим доступу до ресурсу: https://developer.mozilla.org/en-US/

13. W3Schools Online Web Tutorials) [Електронний ресурс] – Режим доступу до ресурсу: https://www.w3schools.com/

14. Stack Overflow - Where Developers Learn, Share, & Build Careers [Електронний ресурс] – Режим доступу до ресурсу: https://stackoverflow.com/

15. Harvard Business Review - Ideas and Advice for Leaders [Електронний ресурс] – Режим доступу до ресурсу: https://hbr.org/

16. McKinsey & Company [Електронний ресурс] – Режим доступу до ресурсу: https://www.mckinsey.com/featured-insights

17. Investopedia [Електронний ресурс] – Режим доступу до ресурсу: https://www.investopedia.com/

18. OWASP Top Ten | OWASP Foundation [Електронний ресурс] – Режим доступу до ресурсу: https://owasp.org/www-project-top-ten/

19. JSON Web Tokens [Електронний ресурс] – Режим доступу до ресурсу: https://jwt.io/

20. World Bank Open Data [Електронний ресурс] – Режим доступу до ресурсу: https://data.worldbank.org/

21. International Monetary Fund (IMF) Data [Електронний ресурс] – Режим доступу до ресурсу: https://data.imf.org/

22. Statista - The Statistics Portal for Market Data, Market Research and Market Studies [Електронний ресурс] – Режим доступу до ресурсу: https://www.statista.com/

23. U.S. Securities and Exchange Commission (SEC) EDGAR Database [Електронний ресурс] – Режим доступу до ресурсу: https://www.sec.gov/edgar

24. Markets data - stock market, bond, equity, commodity prices - FT.com [Електронний ресурс] – Режим доступу до ресурсу: https://markets.ft.com/data

25. The Journal of Finance - Wiley Online Library [Електронний ресурс] – Режим доступу до ресурсу: https://onlinelibrary.wiley.com/journal/15406261

26. Journal of Financial Economics [Електронний ресурс] – Режим доступу до ресурсу: https://www.sciencedirect.com/journal/journal-of-financial-economics

27. Business insights, analysis & perspectives [Електронний ресурс] – Режим доступу до ресурсу: https://www2.deloitte.com/us/en/insights.html

28. Financial Services consulting | McKinsey & Company [Електронний ресурс] – Режим доступу до ресурсу: https://www.mckinsey.com/industries/financial-services/how-we-help-clients

29. А. В. Яковенко, О. О. Коновал Основи програмування: методичні вказівки до виконання комп'ютерних практикумів з дисципліни «Управління IT-проектами». Управління IT-проектами. НТУУ «КПІ ім. І. Сікорського», 2017. – 47с

30. React. The library for web and native user interfaces [Електронний ресурс] – Режим доступу до ресурсу: https://react.dev/

31. Ecma International [Електронний ресурс] – Режим доступу до ресурсу: https://ecma-international.org/

32. MUI: The React component library you always wanted [Електронний ресурс] – Режим доступу до ресурсу: https://mui.com/

33. Chart.js. Open source HTML5 Charts for your website [Електронний ресурс] – Режим доступу до ресурсу: https://www.chartjs.org/

34. Express. Node.js web application framework [Електронний ресурс] – Режим доступу до ресурсу: https://expressjs.com/

35. PostgreSQL. The world's most advanced open source database. [Електронний ресурс] – Режим доступу до ресурсу: https://www.postgresql.org/

36. Хацько Н.Є., Гавриленко С.Ю. Методичні вказівки до виконання лабораторної роботи “Розробка діаграми варіантів використання у середовищі Umbrello UML Modeller” – Харків : НТУ «ХПІ», 2019. – 40 с

37. Data Management Information, News and Tips from TechTarget [Електронний ресурс] – Режим доступу до ресурсу: https://www.techtarget.com/searchdatamanagement/definition/entity-relationship-diagram-ERD

38. ClickiT. Web Application Architecture: The Latest Guide 2024 [Електронний ресурс] – Режим доступу до ресурсу: https://www.clickittech.com/devops/web-application-architecture/

39. IBM. What is a REST API? [Електронний ресурс] – Режим доступу до ресурсу: https://www.ibm.com/topics/rest-apis

40. node-postgres [Електронний ресурс] – Режим доступу до ресурсу: https://node-postgres.com/

41. Node.js is an open-source, cross-platform JavaScript runtime environment [Електронний ресурс] – Режим доступу до ресурсу: https://nodejs.org/en

42. GitHub [Електронний ресурс] – Режим доступу до ресурсу: https://github.com/

43. React Hook Form. Performant, flexible and extensible forms with easy-to-use validation. [Електронний ресурс] – Режим доступу до ресурсу: https://react-hook-form.com/

44. Axios is a simple promise based HTTP client for the browser and node.js. [Електронний ресурс] – Режим доступу до ресурсу: https://axios-http.com/

45. QATestLab. Огляд видів тестування [Електронний ресурс] – Режим доступу до ресурсу: https://training.qatestlab.com/blog/technical-articles/review-the-types-of-testing/

ДОДАТОК А

А.1 Ідентифікація мети ІТ-проекту

Метод **SMART** — це система критеріїв для постановки і оцінки мети та завдань. Кожна літера у слові "SMART" представляє собою конкретний критерій, який допомагає зробити мету більш точною та досяжною.

Таблиця А.1 – Деталізація мети проекту методом SMART

Specific (Конкретна)	Розробити веб-додаток для відстеження фінансів та управління бюджетом.
Measurable (Вимірювана)	Забезпечити можливість додавання, редагування та видалення витрат.
Achievable (Досяжна)	Використовуючи технології React JS, Material UI, Chart.js, PostgreSQL і Express JS, реалізувати функції автентифікації, авторизації, категоризації витрат та візуалізації статистики
Relevant (Реалістична)	Сприяти зручному веденню особистого бюджету та аналізу фінансів. Проект відповідає сучасним потребам в управлінні фінансами та полегшує їх аналіз.
Time-framed (Обмежена у часі)	Реалізація пов'язана із визначеним часовим рамками. Робота має бути завершена відповідно до узгоджених строків замовником проекту, дотримуючись календарного графіку.

Джерело: побудовано автором

А.2 Планування змісту структури робіт

Планування змісту структури робіт здійснюється за допомогою **WBS** (Work Breakdown Structure або Структура розбиття робіт).

WBS – це інструмент управління проектом, який розкладає проект на менші, більш керовані частини для полегшення його виконання та контролю. WBS діаграма відображає ієрархічну структуру завдань та робіт, розкладених за рівнями деталізації, щоб легше визначити обсяг та послідовність робіт у проекті.

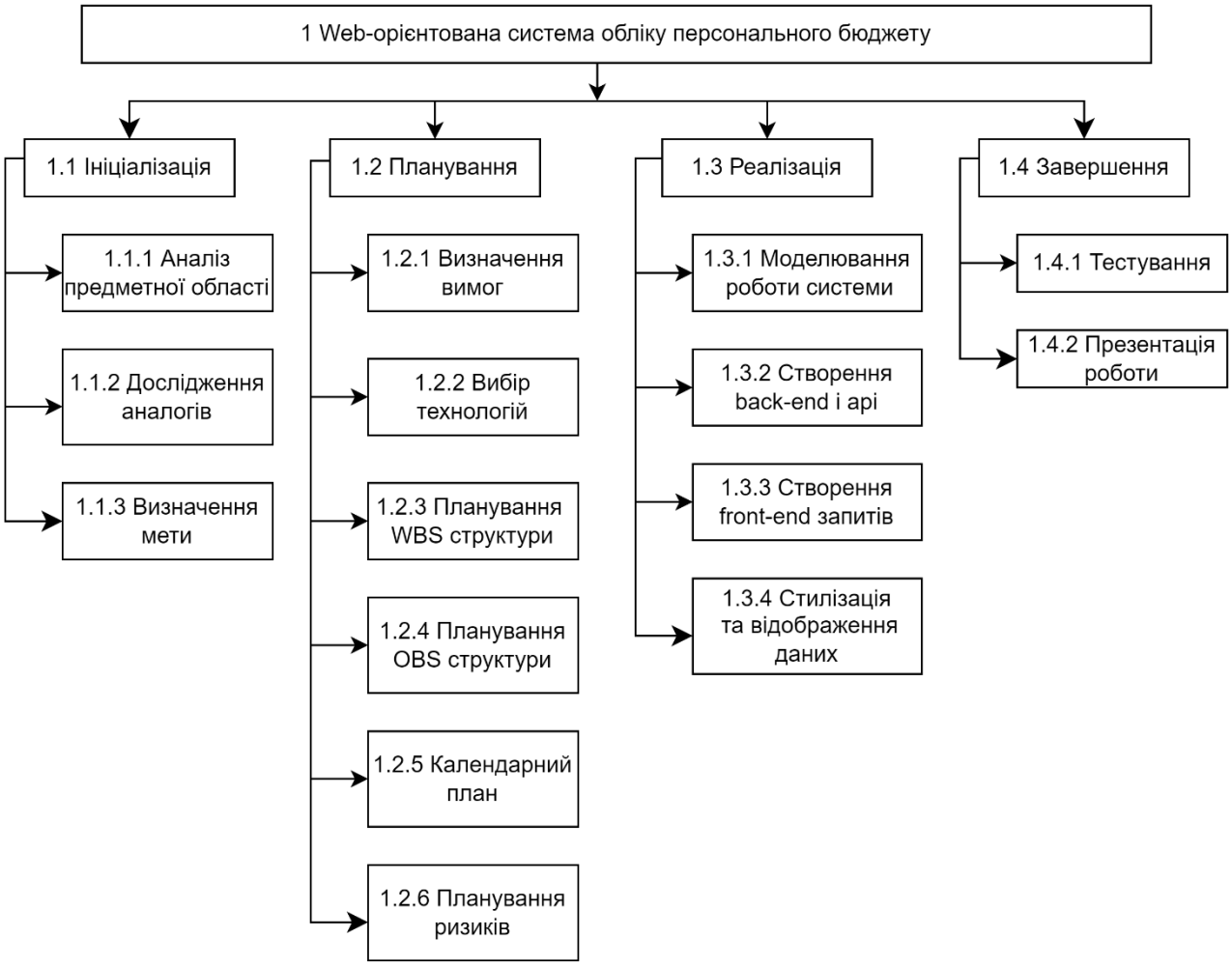

Рисунок А.1 – WBS-структура проекту

Джерело: побудовано автором

OBS (Organizational Breakdown Structure) або структура розподілу обов'язків в організації - це ієрархічна діаграма, яка відображає робочі групи та їхні відповідальності в межах проекту чи організації. Матриця відповідальності визначає ролі та відповідальності конкретних осіб в межах проекту.

OBS в контексті впровадження готового проекту для системи обліку персонального бюджету може виглядати наступним чином:

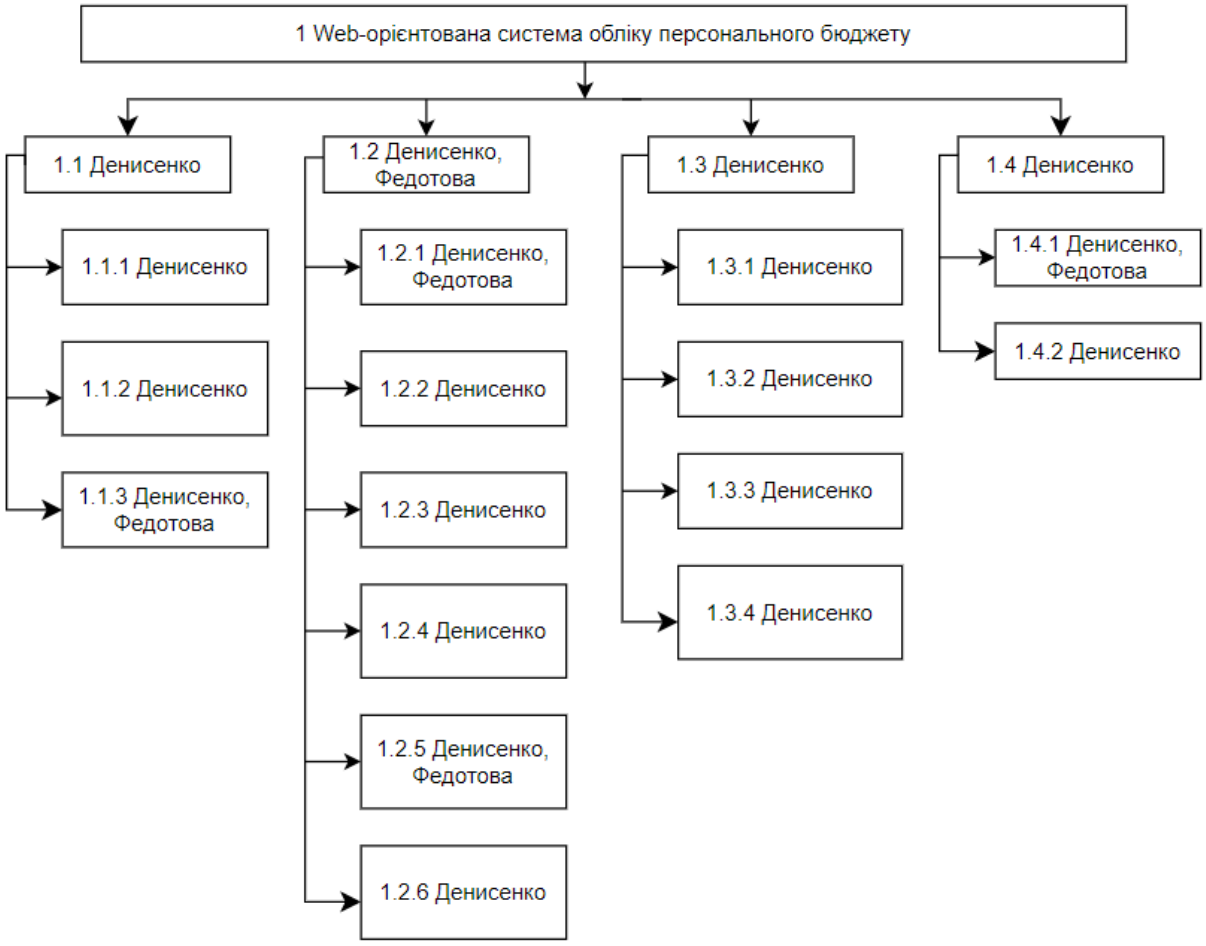

Рисунок А.2 – OBS-структура проекту

Джерело: побудовано автором

Ця діаграма відображає ієрархію в організації для успішного впровадження готового проекту з обліку персонального бюджету. Кожен рівень включає різні ролі та відповідальності, від директора з інформаційних технологій до конкретних команд для впровадження.

Після проектування WBS та OBS структур необхідно побудувати матрицю відповідальності, що показує відповідальність виконавців залежно від ролі. Матрицю відповідальності проекту наведено у таблиці А.2.

Таблиця А.2 – Матриця відповідальності

Назва	Виконавці	
	Денисенко П.Д.	Федотова Н.А.
1.1 Ініціалізація	+	
1.1.1Аналіз предметної області	+	+
1.1.2 Дослідження аналогів	+	
1.1.3 Визначення мети	+	
1.2 Планування	+	+
1.2.1 Визначення вимог	+	+
1.2.2 Вибір технологій	+	
1.2.3 Планування WBS структури	+	
1.2.4 Планування OBS структури	+	
1.2.5 Календарний план	+	+
1.2.6 Планування ризиків	+	
1.3 Реалізація	+	
1.3.1 Моделювання роботи системи	+	
1.3.2 Створення back-end і api	+	
1.3.3 Створення front-end запитів	+	
1.3.4 Стилізація та відображення даних	+	
1.4 Завершення	+	
1.4.1 Тестування	+	+
1.4.2 Презентація роботи	+	

Джерело: побудовано автором

А.3 Побудова календарного графіку виконання інформаційної системи

Для оцінки тривалості виконання роботи над проектом було створено діаграму Ганта. Діаграма Ганта необхідна для відображення завдань та часу запланованого на ці завдання. Завдяки зручній структурі діаграми, вона не потребує багато місця і є досить інформативною. Також вона допомагає відстежувати дати початку та тривалості виконання кожного етапу розробки. Побудована діаграма Ганта зображена на рисунку А.3.

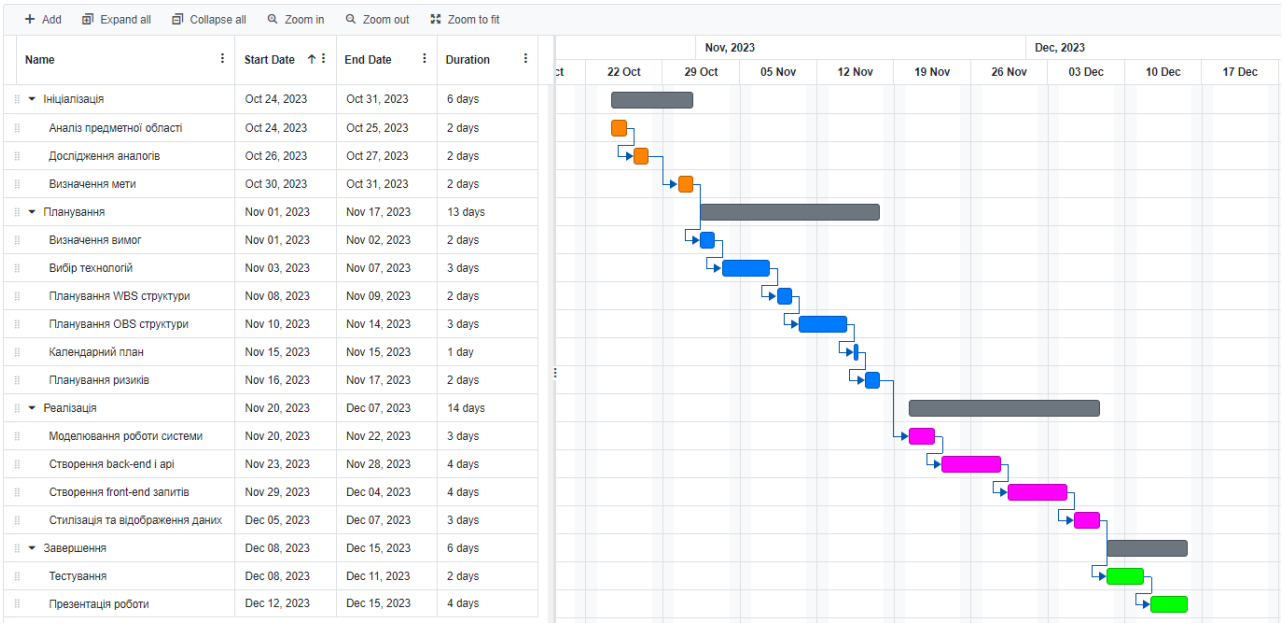

Рисунок А.3 – Діаграма Ганта проекту

Джерело: побудовано автором

А.4 Планування ризиків проекту

Розробка кожного проекту потребує планування ризиків. Якісне планування допоможе заздалегідь виявити майбутні проблеми та вжити заходів для їх усунення.

До головних ризиків під час розробки додатку можна віднести:

- відсутність електроенергії;
- поломка техніки;
- закінчення строку дії ліцензії на програмні продукти;
- відходження від строків виконання роботи зазначеного в календарному плані;
- людський фактор;
- відсутність інтернет зв’язку.

Згідно правил є такі критерії оцінки ризиків:

1. Ймовірність виникнення
2. Рівень ризику
3. Ступінь впливу
4. Ступінь втрат

В свою чергу кожен критерій включає в себе градацію по значущості Ймовірність виникнення:

1. Мінімальна ймовірність
2. Низка ймовірність
3. Середня ймовірність
4. Висока ймовірність
5. Максимальна ймовірність

Рівень ризику:

1. Прийнятні
2. Виправдні

3. Недопустимі

Ступінь впливу:

1. Мінімальний
2. Незначний
3. Допустимий
4. Значний
5. Максимальний

Ступінь втрат:

1. Мінімальний
2. Незначний
3. Допустимий
4. Значний
5. Максимальний

Таблиця А.3 – Матриця відповідальності ймовірності виникнення

Джерело: побудовано автором

Ймовірність виникнення		R1	R2	R3	R4	R5	R6	R7	R8	R9	R10
1	Мінімальна										
2	Низька										
3	Середня										
4	Висока										
5	Максимальна										

Таблиця А.4 – Матриця відповідальності рівня ризику

Джерело: побудовано автором

Рівень ризику		R1	R2	R3	R4	R5	R6	R7	R8	R9	R10
1	Прийнятний										
2	Виправданий										
3	Недопустимий										

Таблиця А.5 – Матриця відповідальності ступеня впливу

Джерело: побудовано автором

Ступінь впливу		R1	R2	R3	R4	R5	R6	R7	R8	R9	R10
1	Мінімальний										
2	Незначний										
3	Допустимий										
4	Значний										
5	Максимальний										

Таблиця А.6 – Матриця відповідальності ступеня втрат

Джерело: побудовано автором

Ймовірність виникнення		R1	R2	R3	R4	R5	R6	R7	R8	R9	R10
1	Мінімальні										
2	Незначні										
3	Допустимі										
4	Значні										
5	Максимальні										

ДОДАТОК Б

Backend

Реалізація **middleware** для верифікації JWT-токена

```
import { Request as ExpressRequest, Response } from 'express';
import jwt, { Secret } from 'jsonwebtoken';

interface Request extends ExpressRequest {
  userId?: number;
}

// Middleware to verify JWT token
const verifyToken = (req: Request, res: Response, next: Function) => {
  const token = req.header('Authorization');

  if (!token) {
    return res.status(401).json({ message: 'Unauthorized' });
  }

  jwt.verify(token, process.env.JWT_SECRET as Secret, (err: any, decoded: any) =>
{
    if (err) {
      return res.status(401).json({ message: 'Invalid token' });
    }

    req.userId = decoded.userId;
    next();
  });
};
```

Реалізація **auth** ендпойнту (реєстрація та автентифікація користувача)

```
import express, { Request, Response } from 'express';
import bcrypt from 'bcrypt';
import jwt, { Secret } from 'jsonwebtoken';
import pool from '../config/db';

const router = express.Router();

router.post('/register', async (req: Request, res: Response) => {
  const { email, password, firstname, lastname } = req.body;

  // Hash the password before saving it to the database
  const hashedPassword = await bcrypt.hash(password, 10);

  try {
    await pool.query('INSERT INTO users(email, password, firstname, lastname)
VALUES($1, $2, $3, $4)', [email, hashedPassword, firstname, lastname]);
```

```
    res.status(201).json({ message: 'User registered successfully' });
  } catch (error) {
    console.error('Error registering user', error);
    res.status(500).send('Internal Server Error');
  }
});

router.post('/login', async (req: Request, res: Response) => {
  const { email, password } = req.body;

  try {
    const result = await pool.query('SELECT * FROM users WHERE email = $1',
[email]);
    const user = result.rows[0];

    if (user) {
      // Check if the provided password matches the stored hashed password
      const passwordMatch = await bcrypt.compare(password, user.password);

      if (passwordMatch) {
        // Create and send a JWT token
        const token = jwt.sign({ userId: user.id, email: user.email },
process.env.JWT_SECRET as Secret, { expiresIn: '1d' });
        res.status(200).json({ data: { token }});
      } else {
        res.status(401).json({ message: 'Invalid credentials' });
      }
    } else {
      res.status(404).json({ message: 'User not found' });
    }
  } catch (error) {
    console.error('Error logging in', error);
    res.status(500).send('Internal Server Error');
  }
});

export default router;
```

Реалізація **expense** ендпойнту (всі CRUD операції з витратами)

```
import express, { Request as ExpressRequest, Response } from 'express';

import pool from '../config/db';
import verifyToken from '../middleware';

const router = express.Router();

interface Request extends ExpressRequest {
  userId?: number;
}

// Endpoint to create an expense
router.post('/expenses', verifyToken, async (req: Request, res: Response) => {
  const { category, amount, currency, date } = req.body;

  try {
    await pool.query(
      'INSERT INTO expenses(user_id, category, amount, currency, date) VALUES($1,
```

```
$2, $3, $4, $5)',
      [req.userId, category, amount, currency, date]
    );
    res.status(201).json({ message: 'Expense created successfully' });
  } catch (error) {
    console.error('Error creating expense', error);
    res.status(500).send('Internal Server Error');
  }
});

// Endpoint to get all expenses for the authenticated user
router.get('/expenses', verifyToken, async (req: Request, res: Response) => {
  try {
    const result = await pool.query('SELECT * FROM expenses WHERE user_id = $1',
[req.userId]);
    const expenses = result.rows;
    res.status(200).json({ data: expenses });
  } catch (error) {
    console.error('Error fetching expenses', error);
    res.status(500).send('Internal Server Error');
  }
});

// Endpoint to get a single expense by id
router.get('/expenses/:id', verifyToken, async (req: Request, res: Response) => {
  const expenseId = req.params.id;

  try {
    const result = await pool.query('SELECT * FROM expenses WHERE id = $1 AND
user_id = $2', [expenseId, req.userId]);
    const expense = result.rows[0];

    if (expense) {
      res.status(200).json({ data: expense });
    } else {
      res.status(404).json({ error: 'Expense not found' });
    }
  } catch (error) {
    console.error('Error fetching expense', error);
    res.status(500).json({ error: 'Internal Server Error' });
  }
});


// Endpoint to update an expense
router.put('/expenses/:id', verifyToken, async (req: Request, res: Response) => {
  const expenseId = req.params.id;
  const { category, amount, currency, date } = req.body;

  try {
    const result = await pool.query(
      'UPDATE expenses SET category = $1, amount = $2, currency = $3, date = $4,
updated_at = CURRENT_TIMESTAMP WHERE id = $5 AND user_id = $6 RETURNING *',
      [category, amount, currency, date, expenseId, req.userId]
    );

    if (result.rows.length === 0) {
      return res.status(404).json({ message: 'Expense not found or unauthorized'
});
    }

    res.status(200).json(result.rows[0]);
  } catch (error) {
```

```
    console.error('Error updating expense', error);
    res.status(500).send('Internal Server Error');
  }
});

// Endpoint to delete an expense
router.delete('/expenses/:id', verifyToken, async (req: Request, res: Response) =>
{
  const expenseId = req.params.id;

  try {
    const result = await pool.query('DELETE FROM expenses WHERE id = $1 AND
user_id = $2 RETURNING *', [expenseId, req.userId]);

    if (result.rows.length === 0) {
      return res.status(404).json({ message: 'Expense not found or unauthorized'
});
    }

    res.status(200).json({ message: 'Expense deleted successfully' });
  } catch (error) {
    console.error('Error deleting expense', error);
    res.status(500).send('Internal Server Error');
  }
});


export default router;
```

Frontend

Створення **AuthProvider**

```
import { createContext, useState, useEffect, ReactNode, useCallback } from
'react';
import { jwtDecode } from 'jwt-decode';

import { Token } from 'src/modules/app/types';

const currentTimestamp = Math.floor(Date.now() / 1000);

interface AuthContextProps {
  children: ReactNode;
}

interface AuthContextValue {
  token: Token;
  setToken: (token: Token) => void;
}

export const AuthContext = createContext<AuthContextValue | undefined>(undefined);
```

```
export const AuthProvider = ({ children }: AuthContextProps) => {
  const [token, setToken] = useState<Token>(localStorage.getItem('token'));

  const decodedToken = token && jwtDecode(token);

  useEffect(() => {
    if (decodedToken && decodedToken.exp) {
      if (decodedToken.exp < currentTimestamp) {
        setToken(null);
      }
    }
  }, [decodedToken]);

  const saveToken = useCallback((token: Token) => {
    setToken(token);

    if (token) {
      localStorage.setItem('token', token);
    } else {
      localStorage.removeItem('token');
    }
  }, []);

  return (
    <AuthContext.Provider value={{ token, setToken: saveToken
}}>{children}</AuthContext.Provider>
  );
};
```

Форма **Входу**

```
import React, { useCallback, useMemo } from 'react';
import { useForm } from 'react-hook-form';
import { useMutation } from '@tanstack/react-query';
import * as yup from 'yup';
import { yupResolver } from '@hookform/resolvers/yup';
import { Button, Typography, Box, Stack, Paper } from '@mui/material';

import { Text, Password } from 'src/modules/ui/forms';
import { useMessages } from 'src/modules/messages/hooks/useMessages';
import { useAuth } from 'src/modules/auth/hooks/useAuth';

import { AuthData } from '../interfaces';
import { authLogin } from '../hooks/crud';

interface LoginFormProps {
  onSubmit: () => void;
}

export const LoginForm = ({ onSubmit }: LoginFormProps) => {
  const { setToken } = useAuth();
  const { show } = useMessages();

  const { mutate } = useMutation((data: AuthData) => authLogin(data, setToken), {
    onSuccess: () => {
      onSubmit();
      show({ message: 'Вітаємо!', severity: 'success' });
```

```
    },
    onError: ({ message }) => show({ message: `Помилка при логіні: ${message}` }),
  });

  const schema = useMemo(
    () =>
      yup.object({
        email: yup
          .string()
          .email('Електронна пошта повинна бути дійсною')
          .required("Обов'язкове поле"),
        password: yup
          .string()
          .min(8, 'Повинно бути більше 8 символів')
          .required("Обов'язкове поле"),
      }),
    [],
  );

  const { handleSubmit, control } = useForm<Omit<AuthData, 'firstname' |
'lastname'>>({
    defaultValues: {
      email: '',
      password: '',
    },
    resolver: yupResolver(schema),
  });

  const handleFormSubmit = useCallback(
    (data: AuthData) => {
      mutate(data);
    },
    [mutate],
  );

  return (
    <Paper elevation={0} sx={{ p: 3, mb: 2 }}>
      <form onSubmit={handleSubmit(handleFormSubmit)} noValidate>
        <Typography variant="h3" mb={1}>
          Вхід
        </Typography>
        <Stack spacing={3}>
          <Text label="Ел. пошта" name="email" type="email" control={control}
required />
          <Password label="Пароль" name="password" type="password"
control={control} required />
        </Stack>

        <Box mt={3}>
          <Button type="submit" fullWidth>
            Увійти
          </Button>
        </Box>
      </form>
    </Paper>
  );
};
```

Форма **Реєстрації**

```
import { useMemo } from 'react';
import { useForm } from 'react-hook-form';
import { useMutation } from '@tanstack/react-query';
import * as yup from 'yup';
import { yupResolver } from '@hookform/resolvers/yup';
import { Button, Typography, Box, Stack, Paper } from '@mui/material';

import { Text, Password } from 'src/modules/ui/forms';
import { useMessages } from 'src/modules/messages/hooks/useMessages';

import { AuthRegisterData } from '../interfaces';
import { authRegisterUser } from '../hooks/crud';

interface RegistrationFormProps {
  onSubmit: () => void;
}

export const RegistrationForm = ({ onSubmit }: RegistrationFormProps) => {
  const { show } = useMessages();

  const { mutate } = useMutation(authRegisterUser, {
    onSuccess: () => {
      onSubmit();
      show({ message: 'Ви успішно створили аккаунт', severity: 'success' });
    },
  });

  const schema = useMemo(
    () =>
      yup.object({
        firstname: yup.string().required("Обов'язкове поле"),
        lastname: yup.string().required("Обов'язкове поле"),
        email: yup
          .string()
          .email('Електронна пошта повинна бути дійсною')
          .required("Обов'язкове поле"),
        password: yup
          .string()
          .min(8, 'Повинно бути більше 8 символів')
          .required("Обов'язкове поле"),
      }),
    [],
  );

  const { handleSubmit, control } = useForm<AuthRegisterData>({
    defaultValues: {
      firstname: '',
      lastname: '',
      email: '',
      password: '',
    },
    resolver: yupResolver(schema),
  });

  const handleFormSubmit = (data: AuthRegisterData) => {
    mutate(data);
  };

  return (
    <Paper elevation={0} sx={{ p: 3, mb: 2 }}>
```

```
      <form onSubmit={handleSubmit(handleFormSubmit)} noValidate>
        <Typography variant="h3" mb={1}>
          Реєстрація
        </Typography>

        <Stack spacing={3}>
          <Text label="Ім'я" name="firstname" control={control} required />
          <Text label="Прізвище" name="lastname" control={control} required />
          <Text label="Ел. пошта" name="email" type="email" control={control}
required />
          <Password
            label="Придумайте пароль"
            name="password"
            type="password"
            control={control}
            required
          />
        </Stack>
        <Box mt={3}>
          <Button type="submit" fullWidth>
            Реєстрація
          </Button>
        </Box>
      </form>
    </Paper>
  );
};
```

Запити до API ендпойнтів для **автентифікації**

```
import { apiClient } from 'src/modules/app/configs/axios';
import { Token } from 'src/modules/app/types';

import { AuthData, AuthRegisterData } from '../interfaces';

const endpoint = '/auth';

export const authLogin = async (data: AuthData, setToken: (token: Token) => void)
=> {
  try {
    const response = await apiClient.post(`${endpoint}/login`, data);
    const token = response.data.token;

    setToken(token);

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

export const authRegisterUser = async (data: AuthRegisterData) => {
  const response = await apiClient.post(`${endpoint}/register`, data);
  return response.data;
};
```

Головна сторінка додатку

```
import { useState, useCallback, useEffect, useMemo } from 'react';

import { useQuery, useMutation, useQueryClient } from '@tanstack/react-query';
import { Box, Button, Typography } from '@mui/material';

import { useAuth } from 'src/modules/auth/hooks/useAuth';
import { Loader } from 'src/modules/ui/loader';
import { ExpensesFormModal } from
'src/modules/expenses/components/ExpensesFormModal';
import { ExpensesList } from 'src/modules/expenses/components/ExpensesList';
import { ExpensesFilterForm } from
'src/modules/expenses/components/ExpensesFilterForm';
import { Expense } from 'src/modules/expenses/interfaces';
import { filterByCurrency, filterByDate } from 'src/modules/expenses/utils';
import { fetchExpenses, deleteExpense } from 'src/modules/expenses/hooks/crud';
import { useMessages } from 'src/modules/messages/hooks/useMessages';
import { useModal } from 'src/modules/ui/modal';

export const ExpensesPage = () => {
  const { token } = useAuth();
  const queryClient = useQueryClient();
  const { show } = useMessages();
  const { openModal, open, onClose } = useModal();

  const [isLoading, seIsLoading] = useState<boolean>(true);

  // Use React Query to fetch all expenses
  const { data, isError } = useQuery(['allExpenses'], () => fetchExpenses(token));

  const initialData = useMemo(() => {
    if (data && !isError) {
      seIsLoading(false);

      return data.filter(
        (expense: Expense) => filterByDate(expense) && filterByCurrency(expense,
'UAH'),
      );
    }
  }, [data, isError]);

  const [expenses, setExpenses] = useState<Expense[]>([]);

  useEffect(() => {
    if (!isLoading) {
      setExpenses(initialData);
    }
  }, [initialData, isLoading]);

  useEffect(() => {
    if (isError) {
      show({ message: 'Помилка отримання даних. Спробуйте пізніше' });
    }
  }, [isError, show]);

  const deleteExpenseMutation = useMutation((id: number) => deleteExpense(id,
token), {
    onSuccess: () => {
      // refetch the expenses data
      queryClient.invalidateQueries(['allExpenses']);
```

```
      show({ message: 'Запис успішно видалено', severity: 'success' });
    },
    onError: ({ message }) => show({ message: `Помилка видалення: ${message}` }),
  });

  const handleDelete = useCallback(
    (id: number) => {
      deleteExpenseMutation.mutate(id);
    },
    [deleteExpenseMutation],
  );

  const handleFiltered = useCallback((filteredExpenses: Expense[]) => {
    setExpenses(filteredExpenses);
  }, []);

  const handleReset = useCallback(
    (reset: boolean) => {
      if (reset && !isLoading && !isError) {
        setExpenses(initialData);
      }
    },
    [isError, isLoading, initialData],
  );

  return (
    <>
      <Box mb={4}>
        <Typography variant="h2" mb={1}>
          Вітаємо вас у Expense Tracker - вашим надійним помічником у керуванні
витратами!
        </Typography>
        <Typography paragraph>
          Будь-який успішний шлях до фінансового благополуччя починається з
ефективного ведення
          обліку ваших витрат. Expense Tracker - це зручний і простий у
використанні інструмент,
          який допоможе вам здійснювати контроль над вашими витратами швидко та
ефективно.
        </Typography>
        <Typography paragraph>
          Почніть прямо зараз та робіть свої фінанси більш контрольованими!
        </Typography>
        <Button onClick={openModal}>Додати витрати</Button>
      </Box>

      <Loader isLoading={isLoading}>
        <ExpensesFilterForm expenses={data} onSubmit={handleFiltered}
onReset={handleReset} />

        <ExpensesList expenses={expenses} onDelete={handleDelete} token={token} />
      </Loader>

      <ExpensesFormModal open={open} onClose={onClose} token={token} />
    </>
  );
};
```

Форма **створення** витрат

```
import { useCallback } from 'react';
import { useMutation, useQueryClient } from '@tanstack/react-query';
import { useForm } from 'react-hook-form';
import { yupResolver } from '@hookform/resolvers/yup';
import * as yup from 'yup';
import { Box, Button, Grid } from '@mui/material';

import { Text, DatePicker } from 'src/modules/ui/forms';
import { useMessages } from 'src/modules/messages/hooks/useMessages';
import { Token } from 'src/modules/app/types';

import { categoryOptions, currencyOptions } from '../constants';
import { Expense } from '../interfaces';
import { createExpense } from '../hooks/crud';

interface ExpenseFormProps {
  token: Token;
}

export const ExpensesForm = ({ token }: ExpenseFormProps) => {
  const queryClient = useQueryClient();
  const { show } = useMessages();

  const schema = yup.object().shape({
    amount: yup.number().required("Обов'язкове поле").positive('Сума має бути
позитивною'),
    currency: yup.string().required("Обов'язкове поле"),
    category: yup.string().required("Обов'язкове поле"),
    date: yup.date().required("Обов'язкове поле"),
  });

  // Create mutation for creating an expense
  const createExpenseMutation = useMutation((data: Expense) => createExpense(data,
token), {
    onSuccess: () => {
      // refetch the expenses data
      queryClient.invalidateQueries(['allExpenses']);

      show({ message: 'Ваші витрати успішно додані', severity: 'success' });
      reset();
    },
    onError: ({ message }) => show({ message: `Помилка: ${message}` }),
  });

  const { control, handleSubmit, formState, reset } = useForm<Expense>({
    defaultValues: {
      amount: undefined,
      currency: 'UAH',
      category: '',
      date: undefined,
    },
    resolver: yupResolver(schema),
  });

  const handleFormSubmit = useCallback(
    (data: Expense) => {
      createExpenseMutation.mutate(data);
    },
    [createExpenseMutation],
  );
```

```
  return (
    <Box py={1}>
      <form onSubmit={handleSubmit(handleFormSubmit)} noValidate>
        <Grid container spacing={2}>
          <Grid item xs={12} sm={6}>
            <Text label="Введіть витрати" name="amount" type="number"
control={control} required />
          </Grid>

          <Grid item xs={12} sm={6}>
            <Text
              label="Валюта"
              name="currency"
              control={control}
              required
              select
              options={currencyOptions}
            />
          </Grid>

          <Grid item xs={12} sm={6}>
            <Text
              label="Категорія"
              name="category"
              control={control}
              required
              select
              options={categoryOptions}
            />
          </Grid>

          <Grid item xs={12} sm={6}>
            <DatePicker label="Дата" name="date" control={control} required />
          </Grid>
        </Grid>

        <Box pt={3}>
          <Button type="submit" disabled={formState.isSubmitting}>
            Зберегти
          </Button>
        </Box>
      </form>
    </Box>
  );
};
```

Форма **редагування** витрат

```
import { useCallback, useEffect } from 'react';
import { useMutation, useQueryClient, useQuery } from '@tanstack/react-query';
import { useForm } from 'react-hook-form';
import { yupResolver } from '@hookform/resolvers/yup';
import * as yup from 'yup';
import { Box, Button, Grid } from '@mui/material';

import { Text, DatePicker } from 'src/modules/ui/forms';
import { useMessages } from 'src/modules/messages/hooks/useMessages';
```

```
import { Token } from 'src/modules/app/types';
import { Loader } from 'src/modules/ui/loader';

import { categoryOptions, currencyOptions } from '../constants';
import { Expense } from '../interfaces';
import { updateExpense, fetchOneExpense } from '../hooks/crud';

interface ExpenseFormProps {
  token: Token;
  editId: number;
}

export const ExpensesFormEdit = ({ editId, token }: ExpenseFormProps) => {
  const queryClient = useQueryClient();
  const { show } = useMessages();

  const schema = yup.object().shape({
    amount: yup.number().required("Обов'язкове поле").positive('Сума має бути
позитивною'),
    currency: yup.string().required("Обов'язкове поле"),
    category: yup.string().required("Обов'язкове поле"),
    date: yup.date().required("Обов'язкове поле"),
  });

  // Use the useQuery hook to fetch a single expense
  const {
    data: oneExpense,
    isLoading,
    isError,
  } = useQuery(['expense', editId], () => fetchOneExpense(editId, token));

  // Define the mutation for updating an expense
  const updateExpenseMutation = useMutation((data: Expense) => updateExpense(data,
token), {
    onSuccess: () => {
      // refetch the expenses data
      queryClient.invalidateQueries(['allExpenses']);

      show({ message: 'Ваші витрати успішно додані', severity: 'success' });
    },
    onError: ({ message }) => show({ message: `Помилка: ${message}` }),
  });

  const { control, handleSubmit, formState, reset } = useForm<Expense>({
    resolver: yupResolver(schema),
  });

  useEffect(() => {
    if (!isLoading && !isError && oneExpense) {
      reset({
        amount: oneExpense.amount,
        currency: oneExpense.currency,
        category: oneExpense.category,
        date: oneExpense.date,
      });
    }
  }, [isError, isLoading, oneExpense, reset]);

  const handleFormSubmit = useCallback(
    (data: Expense) => {
      updateExpenseMutation.mutate({ id: editId, ...data });
    },
    [editId, updateExpenseMutation],
```

```
  );

  return (
    <Box py={1}>
      <Loader isLoading={isLoading} />

      <form onSubmit={handleSubmit(handleFormSubmit)} noValidate>
        <Grid container spacing={3}>
          <Grid item xs={12} sm={6}>
            <Text label="Введіть витрати" name="amount" type="number"
control={control} required />
          </Grid>

          <Grid item xs={12} sm={6}>
            <Text
              label="Валюта"
              name="currency"
              control={control}
              required
              select
              options={currencyOptions}
            />
          </Grid>

          <Grid item xs={12} sm={6}>
            <Text
              label="Категорія"
              name="category"
              control={control}
              required
              select
              options={categoryOptions}
            />
          </Grid>

          <Grid item xs={12} sm={6}>
            <DatePicker label="Дата" name="date" control={control} required />
          </Grid>
        </Grid>

        <Box pt={3}>
          <Button type="submit" disabled={formState.isSubmitting}>
            Редагувати
          </Button>
        </Box>
      </form>
    </Box>
  );
};
```

Форма **фільтрації** витрат

```
import React, { useCallback, useMemo } from 'react';
import { useForm, useWatch } from 'react-hook-form';
import { yupResolver } from '@hookform/resolvers/yup';
import * as yup from 'yup';

import { Button, Grid, Stack, Paper, Typography } from '@mui/material';
```

```
import { DatePicker, Text } from 'src/modules/ui/forms';

import { categoryOptions, currencyOptions } from '../constants';
import { filterByDate, filterByCurrency, filterByCategory } from '../utils';
import { Expense } from '../interfaces';

interface ExpenseFormProps {
  expenses: Expense[];
  onSubmit: (filteredExpenses: Expense[]) => void;
  onReset: (reset: boolean) => void;
}

export const ExpensesFilterForm = ({ expenses, onSubmit, onReset }:
ExpenseFormProps) => {
  const schema = useMemo(
    () =>
      yup.object().shape({
        selectedCurrency: yup.string(),
        selectedCategory: yup.string(),
        startDate: yup.date(),
        endDate: yup.date(),
      }),
    [],
  );

  const { reset, handleSubmit, control } = useForm({
    defaultValues: {
      selectedCurrency: 'UAH',
      selectedCategory: '',
      startDate: undefined,
      endDate: undefined,
    },
    resolver: yupResolver(schema),
  });

  const filter = useWatch({ control });

  const filteredExpenses = useMemo(
    () =>
      expenses.filter(
        expense =>
          filterByDate(expense, filter.startDate, filter.endDate) &&
          filterByCategory(expense, filter.selectedCategory) &&
          filterByCurrency(expense, filter.selectedCurrency),
      ),
    [expenses, filter],
  );

  const handleFormSubmit = useCallback(
    () => onSubmit(filteredExpenses),
    [filteredExpenses, onSubmit],
  );

  const handleFormReset = useCallback(() => {
    reset();
    onReset(true);
  }, [onReset, reset]);

  return (
    <Paper elevation={0} sx={{ p: 3, mb: 6 }}>
      <Typography variant="h3" mb={3}>
        Фільтрувати
      </Typography>
```

```
      <form onSubmit={handleSubmit(handleFormSubmit)} noValidate>
        <Grid container spacing={2}>
          <Grid item xs={12} sm={6} md={3}>
            <Text
              label="Валюта"
              name="selectedCurrency"
              control={control}
              select
              options={currencyOptions}
            />
          </Grid>

          <Grid item xs={12} sm={6} md={3}>
            <Text
              label="Категорія"
              name="selectedCategory"
              control={control}
              select
              options={categoryOptions}
            />
          </Grid>

          <Grid item xs={12} sm={6} md={3}>
            <DatePicker label="Початкова дата" name="startDate" control={control}
/>
          </Grid>

          <Grid item xs={12} sm={6} md={3}>
            <DatePicker label="Кінцева дата" name="endDate" control={control} />
          </Grid>
        </Grid>

        <Stack direction="row" spacing={3} mt={3}>
          <Button onClick={handleFormReset}>Скинути</Button>
          <Button type="submit">Фільтрувати</Button>
        </Stack>
      </form>
    </Paper>
  );
};
```

Функції для фільтрації витрат

```
import { Expense } from '../interfaces';
import { categoryOptions } from '../constants';

export const filterByDate = (
  expenses: Expense[] | Expense,
  startDate?: Date,
  endDate?: Date,
): Expense[] | boolean => {
  if (Array.isArray(expenses)) {
    const filterDate = (date: Date): boolean => {
      return (!startDate || (date && date >= startDate)) && (!endDate || (date &&
date <= endDate));
    };

    return expenses.filter(expense => {
```

```
      const expenseDate = new Date(expense.date);
      return filterDate(expenseDate);
    });
  }

  const expenseDate = new Date(expenses.date);
  return (!startDate || expenseDate >= startDate) && (!endDate || expenseDate <=
endDate);
};

export const filterByCategory = (
  expenses: Expense[] | Expense,
  selectedCategory?: string,
): Expense[] | boolean => {
  if (Array.isArray(expenses)) {
    return expenses.filter(expense => !selectedCategory || expense.category ===
selectedCategory);
  }
  return !selectedCategory || expenses.category === selectedCategory;
};

export const filterByCurrency = (
  expenses: Expense[] | Expense,
  selectedCurrency?: string,
): Expense[] | boolean => {
  if (Array.isArray(expenses)) {
    return expenses.filter(expense => !selectedCurrency || expense.currency ===
selectedCurrency);
  }

  return !selectedCurrency || expenses.currency === selectedCurrency;
};

export const mapCategories = (category: string): string => {
  const mapping = categoryOptions.find(mapping => mapping.value === category);
  return mapping?.name || category;
};
```

Список витрат, побудова та розрахунки для **кругової діаграми**

```
import { useCallback, useEffect, useMemo, useState } from 'react';
import { Chart as ChartJS, ArcElement, Tooltip, Legend } from 'chart.js';
import { Doughnut } from 'react-chartjs-2';
import {
  Stack,
  IconButton,
  List,
  ListItem,
  ListItemText,
  Typography,
  Grid,
  Paper,
  Box,
} from '@mui/material';
import DeleteOutlinedIcon from '@mui/icons-material/DeleteOutlined';
import EditOutlinedIcon from '@mui/icons-material/EditOutlined';

import { useModal } from 'src/modules/ui/modal';
```

```
import { Token } from 'src/modules/app/types';

import { Expense } from '../interfaces';
import { chartBackgroundColor } from '../constants';
import { mapCategories } from '../utils';
import { ExpensesFormEditModal } from './ExpensesFormEditModal';

ChartJS.register(ArcElement, Tooltip, Legend);

interface ExpenseListProps {
  expenses: Expense[];
  token: Token;
  onDelete?: (id: number) => void;
}

export const ExpensesList = ({ expenses, token, onDelete }: ExpenseListProps) => {
  const [expensesId, setExpensesId] = useState<number>();
  const { openModal, open, onClose } = useModal();
  const handleDelete = useCallback((id: number) => () => onDelete && onDelete(id),
[onDelete]);

  const handleEdit = useCallback(
    (id: number) => () => {
      setExpensesId(id);
      openModal();
    },
    [openModal],
  );

  const [categorySums, setCategorySums] = useState<Record<string, number>>({});

  useEffect(() => {
    const updatedCategorySums: Record<string, number> = {};

    expenses.forEach(item => {
      const { category, amount } = item;
      const parsedAmount = parseFloat(`${amount}`);
      const mappedCategory = mapCategories(category);

      if (!isNaN(parsedAmount)) {
        updatedCategorySums[mappedCategory] =
          (updatedCategorySums[mappedCategory] || 0) + parsedAmount;
      }
    });

    setCategorySums(updatedCategorySums);
  }, [expenses]);

  const dataChart = useMemo(() => {
    return {
      labels: Object.keys(categorySums),
      datasets: [
        {
          data: Object.values(categorySums),
          backgroundColor: chartBackgroundColor,
        },
      ],
    };
  }, [categorySums]);

  return (
    <>
      <Box mb={2}>
```

```
        <Typography variant="h2">Список витрат за обраний період</Typography>
        <Typography variant="subtitle2" color="text.secondary">
          За змовчуванням показано валюта "UAH" та поточний місяць. Скористуйтесь
фільтром, щоб
          отримати інші дані.
        </Typography>
      </Box>

      {expenses.length > 0 ? (
        <Grid container spacing={4}>
          <Grid item xs={12} sm={6}>
            <Paper elevation={0} sx={{ p: 3 }}>
              <Doughnut data={dataChart} options={{ responsive: true }} />
            </Paper>
          </Grid>

          <Grid item xs={12} sm={6}>
            <Paper elevation={0} sx={{ p: 3, height: '100%', maxHeight: 570,
overflow: 'auto' }}>
              <List dense>
                {expenses.map(expense => (
                  <ListItem
                    key={expense.id}
                    sx={{
                      '&:not(:last-of-type)': {
                        borderBottom: theme => `1px solid
${theme.palette.grey[300]}`,
                      },
                    }}
                    secondaryAction={
                      <Stack direction="row" spacing={0.5}>
                        <IconButton edge="end" color="info"
onClick={handleEdit(expense.id!)}>
                          <EditOutlinedIcon />
                        </IconButton>
                        <IconButton edge="end" color="error"
onClick={handleDelete(expense.id!)}>
                          <DeleteOutlinedIcon />
                        </IconButton>
                      </Stack>
                    }>
                    <ListItemText
                      primary={
                        <>
                          Витратили:
                          <Typography
                            component="span"
                            variant="body2"
                            color="text.primary"
                            pl={1}
                            fontWeight={700}>
                            {expense.amount} {expense.currency}
                          </Typography>
                        </>
                      }
                      secondary={
                        <>
                          <Typography
                            component="span"
                            variant="body2"
                            color="text.primary"
                            pr={1}
                            fontWeight={500}>
```

```
                            Категорія: {mapCategories(expense.category)}
                          </Typography>
                          {`Дата: ${new Date(expense.date).toLocaleDateString()}`}
                        </>
                      }
                    />
                  </ListItem>
                ))}
              </List>
            </Paper>
          </Grid>
        </Grid>
      ) : (
        <Paper elevation={0} sx={{ p: 3 }}>
          <Typography variant="body2" color="text.secondary">
            Витрат не знайдено. Змініть фільтр або додайте витрати
          </Typography>
        </Paper>
      )}

      {expensesId && (
        <ExpensesFormEditModal open={open} onClose={onClose} token={token}
editId={expensesId} />
      )}
    </>
  );
};
```

Запити до API ендпойнтів для операцій з **витратами**

```
import { apiClient } from 'src/modules/app/configs/axios';
import { Token } from 'src/modules/app/types';

import { Expense } from '../interfaces';

const endpoint = '/expense/expenses';

// Create an expense
export const createExpense = async (expenseData: Expense, token: Token) => {
  try {
    const response = await apiClient.post(endpoint, expenseData, {
      headers: { Authorization: token },
    });
    return response.data;
  } catch (error) {
    throw error;
  }
};

// Edit an expense
export const updateExpense = async ({ id, ...expenseData }: Expense, token: Token)
=> {
  try {
    const response = await apiClient.put(`${endpoint}/${id}`, expenseData, {
      headers: { Authorization: token },
    });
    return response.data;
  } catch (error) {
```

```
    throw error;
  }
};

// Fetch all expenses
export const fetchExpenses = async (token: Token) => {
  try {
    const response = await apiClient.get(endpoint, {
      headers: { Authorization: token },
    });
    return response.data;
  } catch (error) {
    throw error;
  }
};

// Fetch a single expense by id
export const fetchOneExpense = async (id: number, token: Token) => {
  try {
    const response = await apiClient.get(`${endpoint}/${id}`, {
      headers: { Authorization: token },
    });
    return response.data;
  } catch (error) {
    throw error;
  }
};

// Delete an expense
export const deleteExpense = async (id: number, token: Token) => {
  try {
    const response = await apiClient.delete(`${endpoint}/${id}`, {
      headers: { Authorization: token },
    });
    return response.data;
  } catch (error) {
    throw error;
  }
};
```