МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ОДЕСЬКИЙ ДЕРЖАВНИЙ ЕКОЛОГІЧНИЙ УНІВЕРСИТЕТ

Факультет комп'ютерних наук,
управління та адміністрування
Кафедра інформаційних технологій

Кваліфікаційна робота бакалавра

на тему: Розробка інформаційної системи для відстеження балансу на віртуальному гаманці

Виконав студент групи **К-19б**
спеціальності 122 Комп'ютерні науки
Кухта Ілля Олександрович

Керівник **ст. викладач**
Вохменцева Т.Б.

Консультант **д.ф., доцент**
Бучинська І.В.

Рецензент **к.ф.-м.н., професор**
Ковальчук В.В.

Одеса 2023

ЗМІСТ

Перелік умовних позначок ... 5
Вступ ... 6
1 Аналіз проблем обліку та прогнозування персональних витрат ... 8
1.1 Аналіз основних статей витрат і доходів ... 8
1.2 Аналіз методів оцінювання ймовірних обсягів витрат і доходів ... 9
1.3 Порівняльний аналіз існуючих рішень проблеми ... 10
1.4 Аналіз вимог до програмного забезпечення ... 13
2 Проєктування системи моделювання персональних витрат ... 15
2.1 Характеристика обраних інструментів для розробки та проєктування ... 16
2.2 Опис етапів проєктування інформаційної системи ... 20
3 Опис реалізації проєктного рішення ... 23
3.1 Опис розробленої інформаційної системи ... 23
3.2 Аналіз архітектури серверної сторони ... 25
3.2.1 Опис процесу підключення до бази даних ... 27
3.2.2 Опис маршрутів та обробників маршрутів ... 28
3.2.3 Опис контролерів та їх ролі в системі ... 31
3.2.4 Опис та аналіз моделей даних ... 32
3.3 Аналіз архітектури клієнтської сторони ... 34
3.3.1 Опис та характеристика компонентів системи ... 34
3.3.2 Опис сервісного рівня ... 39
3.3.3 Опис структури сторінок інформаційної системи ... 41
3.4 Аналіз структури бази даних ... 46
3.4.1 Опис структури даних користувача ... 47
3.4.2 Опис структури транзакційних даних ... 48
4. Рекомендації щодо вдосконалення ІС ... 49
Висновки ... 52
Перелік джерел посилання ... 54

ПЕРЕЛІК УМОВНИХ ПОЗНАК

ІС - інформаційна система.

БД – база даних.

Веб-додаток – Веб-програма або веб-застосунок.

ДК – діаграма класів.

ДП – діаграма послідовності.

API – Application Programming Interface.

JSON – JavaScript Object Notation.

MVC – Model-View-Controller.

NoSQL – Not Only SQL.

REST – Representational State Transfer.

HTTP – Hypertext Transfer Protocol.

SQL – Structured Query Language.

UI – User Interface.

UML – Unified Modeling Language.

JWT – JSON Web Token.

ВСТУП

Цифрова революція ретельно трансформує всі аспекти життя, включаючи область фінансових взаємодій та транзакцій. В сучасному динамічному світі, що визначається оцифрованістю, транзакції відбуваються дуже швидко. Важливість грошей як обмінного ресурсу та важливого елемента економічних взаємовідносин лише збільшується. Оцифрованість грошей призводить до появи нових можливостей для управління фінансами та викликає проблеми щодо відстеження доходів і витрат.

Обсяг і складність транзакцій зростають, що робить ефективне управління фінансами все більш викликаючим завданням. У такому середовищі важко підтримувати здоровий баланс між доходами та витратами. Надходження заробітної плати через банківські перекази, можливість проведення онлайн-транзакцій та наявність кредитних можливостей створюють широкий спектр можливостей для задоволення потреб.

Ці можливості також вимагають ретельної і точної оцінки фінансової діяльності для уникнення перевитрат і забезпечення фінансової стабільності. Послідовне, всеосяжне і чітке розуміння фінансових показників може значно покращити здатність ефективно управляти фінансами.

Зростання кількості випадків неправильного управління особистими фінансами, таких як неповернення кредитів та перевитрати, свідчить про поширені проблеми при управлінні фінансами. Наслідки такого неправильного управління можуть бути різними - від фінансових труднощів до серйозних юридичних проблем. Це підкреслює важливість ефективного управління особистими фінансами та викликає потребу в надійних та ефективних інструментах для цього.

У цьому контексті набуває актуальності розробка надійного та ефективного трекера особистих доходів та витрат. Виділення якісних рішень в цій області може принести значущі користі до суспільства, поліпшуючи розуміння особистих фінансів і сприяючи більш обґрунтованому прийняттю рішень.

Розробка такого застосунку, що допоможе відстежувати доходи і витрати, ставити фінансові цілі і надавати корисну інформацію для прийняття обґрунтованих фінансових рішень, є центральною ціллю цієї роботи. Існує необхідність в розробці застосунку, що допоможе відстежувати доходи і витрати, ставити фінансові цілі і надавати корисну інформацію для прийняття обґрунтованих фінансових рішень.

Розробка такого застосунку потребує великої кількості роботи, яка включає в себе глибокий аналіз сучасних фінансових систем, потреб користувачів і найкращих практик управління фінансами. Робота буде включати в себе порівняльний аналіз існуючих рішень для управління особистими фінансами, визначення конкретних вимог до нового застосунку, розробку безпечної і ефективної структури бази даних для зберігання даних користувачів і розробку архітектури програмного забезпечення.

Ця робота, безсумнівно, сприятиме збільшенню обізнаності користувачів з питань особистих фінансів та поліпшенню їх фінансового благополуччя. Особисті фінанси тісно пов'язані з економічною стабільністю суспільства в цілому, і кожна людина повинна відчувати контроль над своїми фінансами, щоб досягнути успіху.

З огляду на вище сказане, ця робота має на меті розробку застосунку для відстеження особистих доходів і витрат, що сприятиме покращенню управління особистими фінансами. Наступним кроком цієї роботи буде аналіз вимог до такого застосунку, його розробка та тестування. Результати цієї роботи допоможуть зрозуміти, як може виглядати оптимальний застосунок для відстеження особистих доходів та витрат в сучасному цифровому світі.

Дана кваліфікаційна робота бакалавра, складається з 55 сторінок, 11 рисунків та 9 джерел посилання.

1 АНАЛІЗ ПРОБЛЕМ ОБЛІКУ ТА ПРОГНОЗУВАННЯ ПЕРСОНАЛЬНИХ ВИТРАТ

1.1 Аналіз основних статей витрат і доходів

В основі відстеження особистих фінансів лежать основні категорії доходів і витрат: різні категорії, включаючи заробітну плату, дивіденди, відсотки та інші джерела доходу, складають загальний дохід людини. З іншого боку рівняння – витрати, які варіюються від предметів першої необхідності, таких як житло (іпотека або оренда), їжа, комунальні послуги та транспорт, до предметів розкоші, таких як відпочинок, розваги та особиста розкіш.

Розробка програмного забезпечення для відстеження особистих фінансів вимагає чіткого розуміння специфіки цих категорій, оскільки вони можуть суттєво впливати на те, як програма їх обробляє.

Наприклад, заробітна плата, як правило, є фіксованою і надходить щомісяця, тоді як дивіденди та відсотки можуть коливатися і бути менш передбачуваними. Витрати на житло та комунальні послуги, на відміну від дискреційних витрат, таких як розваги та предмети особистої розкоші, часто є фіксованими і повторюються щомісяця, але можуть значно відрізнятися в інших випадках.

Крім того, класифікація витрат на "потреби" і "бажання" може бути корисним інструментом для користувачів, які прагнуть визначити пріоритети або скоротити витрати. Надійне програмне забезпечення для відстеження особистих фінансів має бути здатним класифікувати, відстежувати та аналізувати всі ці типи доходів і витрат.

Гнучкість кастомізації, що дозволяє користувачам пристосовувати категорії до унікальних потреб, підвищить зручність і привабливість програмного забезпечення.

1.2 Аналіз методів оцінювання ймовірних обсягів витрат і доходів

Ретельне прогнозування потенційних обсягів доходів і витрат є життєво важливим елементом управління особистими фінансами. Цей процес, як правило, здійснюється за допомогою різноманітних методів, кожен з яких має свої унікальні переваги та недоліки.

Для прогнозування доходів ключову роль відіграє передбачуваність джерела доходу. Фіксовані джерела доходу, такі як заробітна плата або пенсії, можуть бути прогнозовані з певним ступенем впевненості завдяки їх передбачуваності та стабільності. З іншого боку, змінні джерела доходу, такі як фріланс, бонуси або інвестиційний прибуток, вимагають більш складного підходу для точного прогнозування.

Статистичні методи, такі як ковзні середні або регресійний аналіз, часто використовуються, щоб врахувати мінливість цих джерел доходу. Що стосується прогнозування витрат, то фіксовані витрати, такі як орендна плата або іпотечні платежі, комунальні послуги та страхові внески, передбачити відносно просто, оскільки вони повторюються і є передбачуваними за своєю природою. Однак змінні витрати, які включають такі категорії, як продукти харчування, розваги та особиста гігієна, є більш складним завданням для прогнозування. У цьому випадку аналіз історичних даних про витрати може дати цінну інформацію. Тенденції та закономірності в цих даних часто демонструють циклічність і тому можуть бути використані для обґрунтованих прогнозів щодо майбутньої поведінки витрат.

Крім того, різні методи бюджетування можуть забезпечити цінну основу для прогнозування видатків. Наприклад, нульове бюджетування, яке передбачає планування доходу, забезпечує детальний і практичний підхід до управління особистими фінансами. В якості альтернативи, правило 50/30/20 пропонує більш спрощений підхід, який передбачає розподіл доходу після сплати податків на потреби (50%), бажання (30%) та заощадження (20%).

Алгоритми машинного навчання стали великим проривом у прогнозуванні доходів і витрат. Використовуючи велику кількість історичних даних, ці алгоритми можуть адаптуватися до індивідуальних моделей витрат і змін у способі життя, що призводить до все більш точних прогнозів. Ця розробка має великі перспективи для майбутнього розвитку рішень для управління особистими фінансами.

1.3 Порівняльний аналіз існуючих рішень проблеми

Сфера інструментів управління особистими фінансами неймовірно різноманітна, і кожна програма пропонує власний унікальний підхід щоб забезпечити ретельну оцінку управління особистими фінансами, важливо провести порівняльний аналіз існуючих рішень у цій галузі. З цієї причини важливо детальніше розглянути два відомі додатки: "Монетний двір" та "Тобі потрібен бюджет (YNAB)".

"Mint", новаторський застосунок у світі управління особистими фінансами, привернув увагу мільйонів користувачів завдяки широкому набору функцій та зручному інтерфейсу. Серед переваг, пов'язаних з цим інструментом, можна виділити наступні:

– інтеграція та автоматизація: однією з видатних особливостей Mint є його здатність консолідувати кілька фінансових рахунків – банківські рахунки, кредитні картки або інвестиційні портфелі (функція автоматизації ефективно класифікує транзакції, надаючи користувачам цілісне уявлення про їхній фінансовий стан у будь-який момент часу);

- налаштовуване бюджетування та відстеження витрат: за допомогою Mint користувачі можуть налаштовувати свої бюджети на основі своїх фінансових цілей і відстежувати витрати відповідно до цих критеріїв (динамічні візуалізації, що ілюструють структуру витрат за різними категоріями та періодами часу, допомагають користувачам приймати обґрунтовані фінансові рішення);
- безкоштовно: мабуть, однією з найпереконливіших переваг Mint є його економічна ефективність (вона пропонує ці комплексні послуги безкоштовно, усуваючи будь-які фінансові бар'єри для потенційних користувачів).

Незважаючи на потужний набір функцій, Mint також має певні недоліки:

- автоматизована категоризація транзакцій: хоча автоматизація полегшує процес, вона іноді призводить до неправильної категоризації транзакцій, вимагаючи від користувачів вносити виправлення вручну незручність, яка може порушити загальний користувацький досвід;
- відсутність конкретного фокусу: Mint представляє огляд фінансової ситуації користувача, але не зосереджується на конкретних аспектах, таких як планування бюджету або заощадження.

На противагу цьому, "You Need a Budget" (YNAB) застосовує особливий підхід до управління особистими фінансами, що відповідає принципам бюджетування з нульовим рівнем витрат. Переваги використання YNAB включають:

- проактивний підхід до бюджетування: YNAB заохочує користувачів "дати кожному долару роботу", маючи на увазі, що вони повинні планувати свій бюджет заздалегідь, перш ніж витрачати кошти (такий підхід виховує фінансову відповідальність, допомагає користувачам жити за коштами та розвиває мислення, орієнтоване на заощадження);

– поглиблені освітні ресурси: YNAB вирізняється своєю прихильністю до навчання користувачів. Він надає вичерпні посібники, інтерактивні вебінари та чуйну підтримку клієнтів (користувачі можуть скористатися цими ресурсами, щоб покращити свої навички управління фінансами та максимально ефективно використовувати інструмент.

Тим не менш, YNAB не позбавлений недоліків):

– початкова крива навчання: впровадження нульового бюджетування, хоча і є потенційно вигідним, може бути складним для новачків
(ця складність може призвести до крутої початкової кривої навчання, яка може відлякувати деяких користувачів);
– абонентська плата: на відміну від Mint, YNAB вимагає абонентської плати, що може "відлякати" потенційних користувачів, особливо тих, хто має обмежений бюджет.

Провівши цей порівняльний аналіз, стає очевидним, що і Mint, і YNAB пропонують значні переваги, але водночас мають певні обмеження. Ідеальне програмне рішення мало б поєднувати комплексний фінансовий менеджмент, який пропонує Mint, та проактивне бюджетування й освітній акцент, який робить YNAB. Водночас, воно також повинно пом'якшити виявлені недоліки, такі як відсутність конкретного фокусу в Mint і крута крива навчання в YNAB.

Кваліфікаційна робота бакалавра спрямований на розробку інструменту управління особистими фінансами, який об'єднає сильні сторони, пристосується до індивідуальних потреб користувачів і запропонує комплексне та ефективне рішення для управління особистими фінансами.

1.4 Аналіз вимог до програмного забезпечення

Оцінюючи динаміку управління особистими фінансами та існуючі інструменти, стає очевидним, що ефективне рішення повинно включати певні основні функції для задоволення потреб користувачів та усунення виявлених обмежень. Вимоги до запропонованого програмного забезпечення для управління особистими фінансами можна умовно поділити на категорії.

Інтеграція та автоматизація є ключовими аспектами програмного забезпечення, що забезпечують безперешкодний зв'язок з багатьма фінансовими установами та автоматизують процес пошуку і класифікації транзакцій. Усуваючи надмірне ручне введення даних, ця функція надає користувачам комплексне уявлення про їхню фінансову ситуацію.

Щоб задовольнити індивідуальні потреби, програмне забезпечення пропонує настроювані варіанти бюджетування. Користувачі можуть створювати бюджети, пристосовані до їхніх конкретних моделей доходів і витрат. Крім того, надійна функція відстеження витрат дозволяє користувачам контролювати та візуалізувати свої витрати за різними категоріями.

Підкреслюючи проактивний підхід до бюджетування, програма заохочує користувачів розподіляти свої доходи до того, як вони будуть витрачені. Це сприяє підвищенню фінансової відповідальності та розвиває мислення, орієнтоване на заощадження.

Поглиблені освітні ресурси, щоб користувачі могли покращити свої навички фінансової грамотності та максимально ефективно використовувати інструмент, програмне забезпечення повинно містити вичерпні посібники та ресурси, що пояснюють концепції особистих фінансів та функціональність програмного забезпечення.

Зручність використання, програмне забезпечення повинно мати інтуїтивно зрозумілий та зручний інтерфейс для полегшення навігації, скорочення початкового періоду навчання та покращення загального користувацького досвіду.

Запропоноване рішення має бути економічно ефективним, щоб усунути фінансові обмеження для потенційних користувачів, сприяючи таким чином широкому впровадженню.

Безпека та конфіденційність даних, необхідна враховуючи чутливий характер фінансових даних, програмне забезпечення повинно відповідати суворим стандартам безпеки та конфіденційності даних, забезпечуючи належний захист інформації користувачів.

Наявність спрощеного та чистого користувацького інтерфейсу (UI) є важливою передумовою для підвищення ергономіки та зручності використання програмного забезпечення. До об'єктів дослідження в цій області відноситься розробка простого та зрозумілого дизайну користувацького інтерфейсу. Інформаційна архітектура повинна бути розроблена таким чином, щоб швидко надавати ключову інформацію, зменшувати безлад і не перевантажувати користувача. Візуальні елементи повинні бути інтуїтивно зрозумілими, з використанням послідовної та чіткої іконографії, типографіки та кольорових схем. Метою є створити бездоганний користувацький досвід, який сприяє легкій навігації та швидкому розумінню функціональності програми. Крім того, програмне забезпечення має бути адаптивним, забезпечуючи зручність його використання на різних типах пристроїв і розмірах екранів.

Таким чином, перераховані вище вимоги формують комплексний план для розробки функціонального, доступного та безпечного інструменту управління особистими фінансами. Це програмне забезпечення має на меті пом'якшити проблеми, притаманні існуючим рішенням, надаючи кращу альтернативу, яка відповідає потребам користувачів в управлінні фінансами та покращує їхній загальний досвід.

2 ПРОЄКТУВАННЯ СИСТЕМИ МОДЕЛЮВАННЯ ПЕРСОНАЛЬНИХ ВИТРАТ

Етап проєктування архітектури системи контролю особистих витрат є критично важливим етапом життєвого циклу системи. Він відіграє вирішальну роль у перетворенні цілей та вимог, визначених на етапі аналізу, у комплексний план, який є основою для розробки системи. Цей процес передбачає ретельне вивчення проблемної області з використанням систематичних стратегій та інструментів для фіксації складних деталей бажаної функціональності.

В основі етапу проєктування лежить побудова структури системи, що виходить за рамки простого розподілу функціональних можливостей і поширюється на вибір відповідних технологій та програмних компонентів. Цей систематичний поділ системи на керовані компоненти забезпечує зручність обслуговування та масштабованість програмного забезпечення, зберігаючи при цьому його узгодженість та послідовність.

Окрім визначення структури системи, на етапі проєктування увага також приділяється деталізації взаємодії між різними компонентами системи.

Це включає визначення комунікаційних інтерфейсів і протоколів обміну даними для забезпечення безперешкодної інтеграції компонентів системи.

Аутентифікація, яка забезпечує безпеку та цілісність даних, також планується на цьому етапі. Вона слугує основним механізмом перевірки особи користувача та забезпечення безпечної комунікації всередині системи.

Етап проєктування також включає положення про контроль версій і стратегії розгортання. Вони необхідні для відстеження змін, управління різними версіями програмного забезпечення та забезпечення безперебійного процесу розгортання.

Невід'ємною частиною проєктування системи є чітко визначена стратегія тестування. Вона окреслює підхід до модульного, інтеграційного та системного тестування, щоб гарантувати, що кожен компонент системи працює так, як очікується.

Цей етап дозволяє виявити потенційні проблеми на ранніх стадіях і надає дорожню карту для майбутніх удосконалень і коригувань. По суті, етап проєктування є ітеративним, що сприяє створенню середовища, сприятливого для безперервного вдосконалення архітектури системи впродовж її розробки.

2.1 Характеристика обраних інструментів для розробки та проєктування

Стратегічний вибір технологій лежить в основі цього проєкту, і було прийнято-рішення використовувати стек MERN, який складається з MongoDB, Express.js, React.js та Node.js. Цей стек забезпечує надійну основу для створення ефективних та масштабованих рішень. Ключовою особливістю архітектури проєкту є розділення фронтенд- та бекенд-компонентів, що забезпечує добре структуровану, зручну в обслуговуванні та надійну систему. Це чітке розмежування, візуально підкріплене супровідною архітектурною схемою, ілюструє складні взаємозв'язки та обов'язки між компонентами (рис.1).

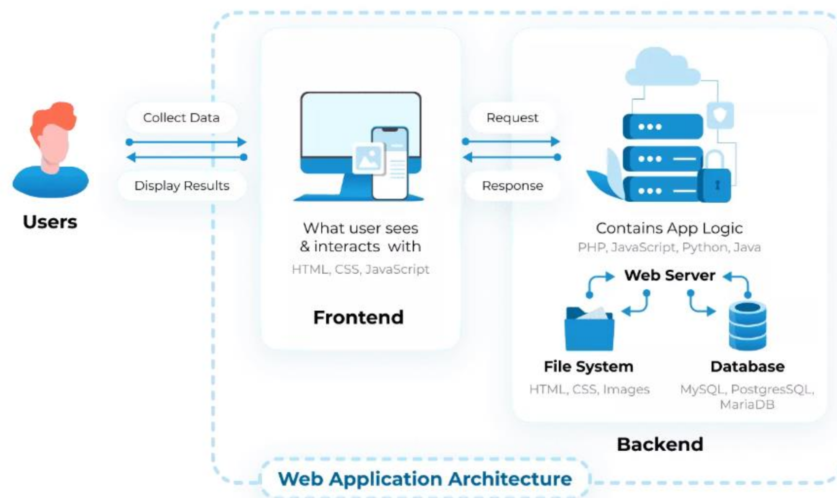

Рисунок 1 – Архітектура веб-додатків

Для розробки компонентів користувацького інтерфейсу (UI) було обрано React.js – бібліотеку JavaScript з відкритим кодом. Її застосування виправдане можливістю створювати складні, адаптивні інтерфейси з високою продуктивністю завдяки алгоритму віртуальної об'єктної моделі документа (DOM). Цей алгоритм мінімізує пряму взаємодію з дорогим DOM, тим самим оптимізуючи загальну продуктивність додатку, що є дуже важливим фактором для інтерактивного додатку для управління особистими фінансами. React.js також підтримується активною спільнотою розробників, як наслідок створено великий репозиторій з готовими до використання компонентів і бібліотек.Таке різноманіття ресурсів дозволяє прискорити розробку, оптимізувати рішення та скоротити час розробки [1].

Для архітектури на стороні сервера було обрано Node.js, що базується на неблокуючій, керованій подіями моделі вводу/виводу та працює на основі JavaScript-рушія Google V8. Ця легка архітектура пропонує чудову продуктивність і масштабованість, що робить її придатною для застосунок реального часу, які працюють на розподілених пристроях. Супровідна блок-схема ефективно демонструє цикл обробки подій Node.js та його переваги представлені на рис.2.

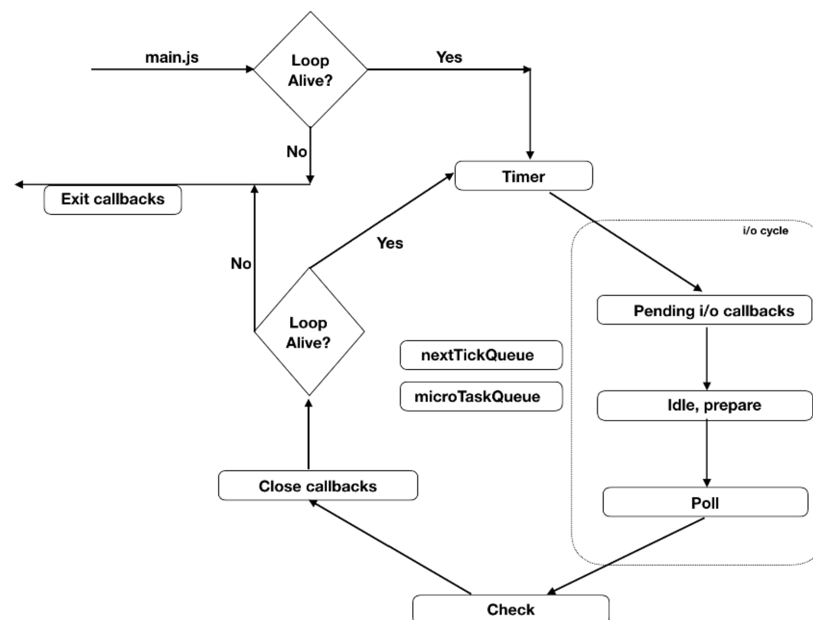

Рисунок 2 – Цикл обробки подій Node.js

Фреймворк Express.js доповнює Node.js в архітектурі на стороні сервера, спрощуючи процес налаштування серверних маршрутів і проміжного програмного забезпечення, обробки запитів і відповідей, а також інтеграції з базою даних MongoDB.

Стандарт JSON Web Tokens (JWT) був включений для забезпечення безпечної передачі вимог між двома сторонами. JWT забезпечують компактний, безпечний для URL-адрес механізм для підтримки безпечних сеансів користувачів на серверах без статусу. Взаємодія між клієнтом, сервером і токенами може бути представлена у вигляді блок-схеми, що візуально демонструє етапи аутентифікації та авторизації користувача [2] (рис.3).

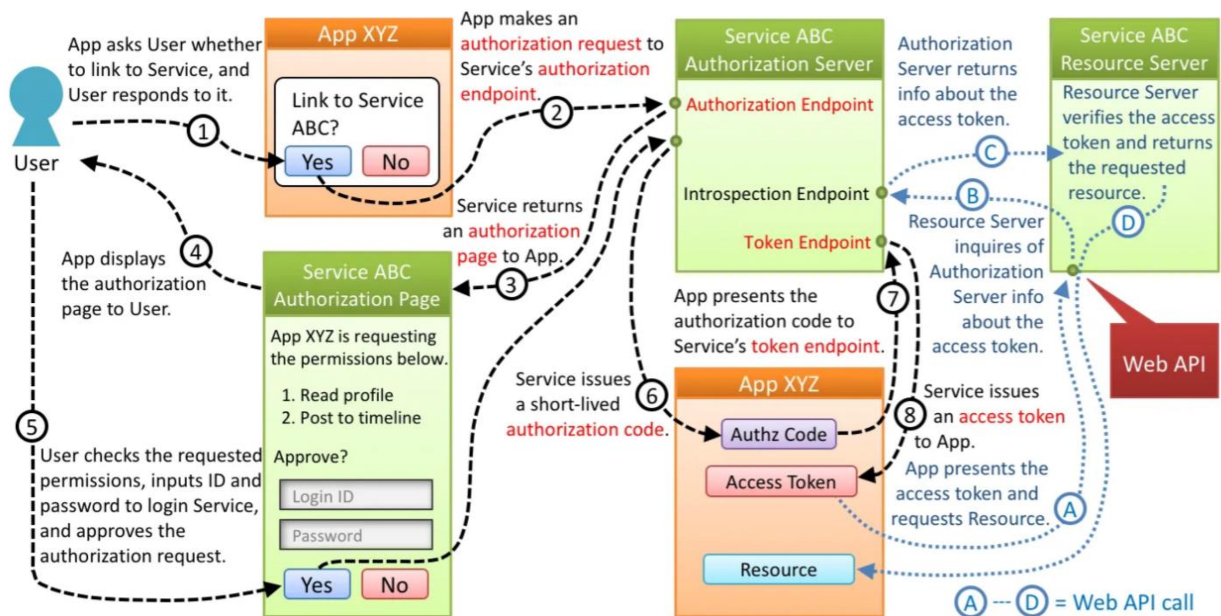

Рисунок 3 – Етапи аутентифікації та авторизації користувача

Git, розподілена система контролю версій, була обрана для управління змінами в кодовій. Притаманна їй децентралізація дозволяє багатьом учасникам одночасно працювати над різними функціями, не порушуючи стабільності основної кодової бази.

Docker використовує модель клієнт-сервер, де клієнт Docker взаємодіє з демоном Docker для керування контейнерами Docker. Ця взаємодія може відбуватися в одній системі або на різних машинах. Зв'язок забезпечується за допомогою REST API, UNIX-сокетів або мережевого інтерфейсу. Docker Compose, варіант клієнта Docker, організовує багатоконтейнерні застосунок. Архітектуру Docker, яка підкреслює оркестровку контейнерів, можна зобразити на схемі, щоб проілюструвати його роль у технологічній екосистемі програми [3] (див.рис.4).

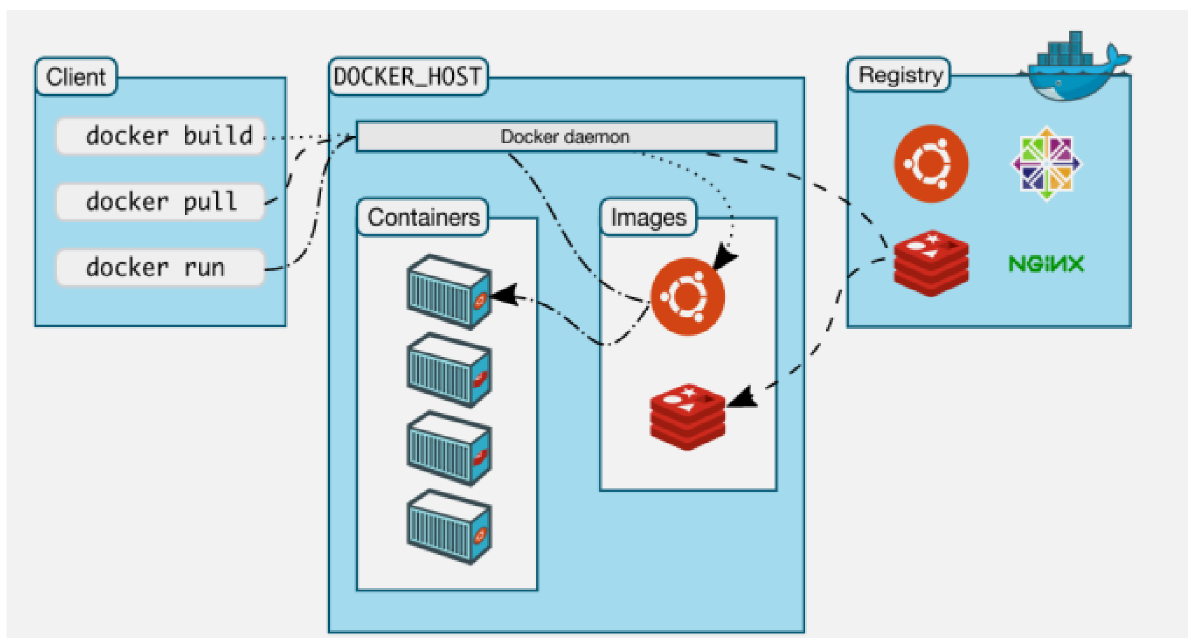

Рисунок 4 – Роль Docker у технологічній екосистемі програми.

Amazon Web Services (AWS), універсальна хмарна платформа, що пропонує широкий спектр послуг. AWS було обрано завдяки її масштабованій інфраструктурі, надійному обслуговуванню та надійним заходам безпеки, що є важливими для розміщення застосунок.

Стратегія тестування включає модульне тестування за допомогою Jest і тестування API за допомогою Postman. Jest, простий у використанні фреймворк для тестування JavaScript, використовується завдяки своїй простоті та підтримці великих додатків. Postman, з іншого боку, надає комплексне середовище для тестування кінцевих точок API, що є критично важливим компонентом для перевірки функціональності на стороні сервера.

2.2 Опис етапів проєктування інформаційної системи

Розробка ефективної, надійної та орієнтованої на користувача інформаційної системи є складним, багатогранним завданням, яке вимагає ретельного та структурованого підходу. Даний проєкт розроблявся за ретельною, поетапною методологією, яка дозволила поглиблено зосередитися на кожному критичному аспекті процесу розробки системи. Цей методичний підхід складався з п'яти окремих етапів:

- аналіз вимог;
- проєктування системи;
- реалізація;
- тестування;
- розгортання.

Кожнен з цих етапів робить унікальний і значний внесок під час розробки проєкту.

Фундаментальною фазою проєкту є аналіз вимог. Ця ключова фаза була зосереджена на розумінні точних потреб та очікувань кінцевих користувачів. Завдяки детальному дослідженню функціональних і нефункціональних вимог, цей етап мав на меті отримати всебічне розуміння призначення системи та потенційних варіантів використання.

Акцент був зроблений на розумінні того, як застосунок може найкраще допомогти користувачам в управлінні особистими фінансами, як наслідок визначено ключові функції, такі як аутентифікація користувача, реєстрація транзакцій, інструменти бюджетування та ресурси фінансового планування. Комплексний аналіз вимог забезпечив відповідність розробки системи потребам користувачів, що має вирішальне значення для підвищення задоволеності користувачів і загальної ефективності системи.

Після цього розпочався етап проєктування системи, на якому було створено концептуальну модель системи. На цьому етапі були прийняті ключові рішення щодо технологічного стеку та архітектурної структури системи. Стек MERN, що складається з MongoDB, Express.js, React.js та Node.js, був обраний через його масштабованість, ефективність роботи та потужну підтримку спільноти [4]. Система була логічно розділена на дві окремі частини: фронтенд, який відповідає за взаємодію з користувачем і бекенд, який відповідає за обробку даних і бізнес-логіку (див.рис.5).

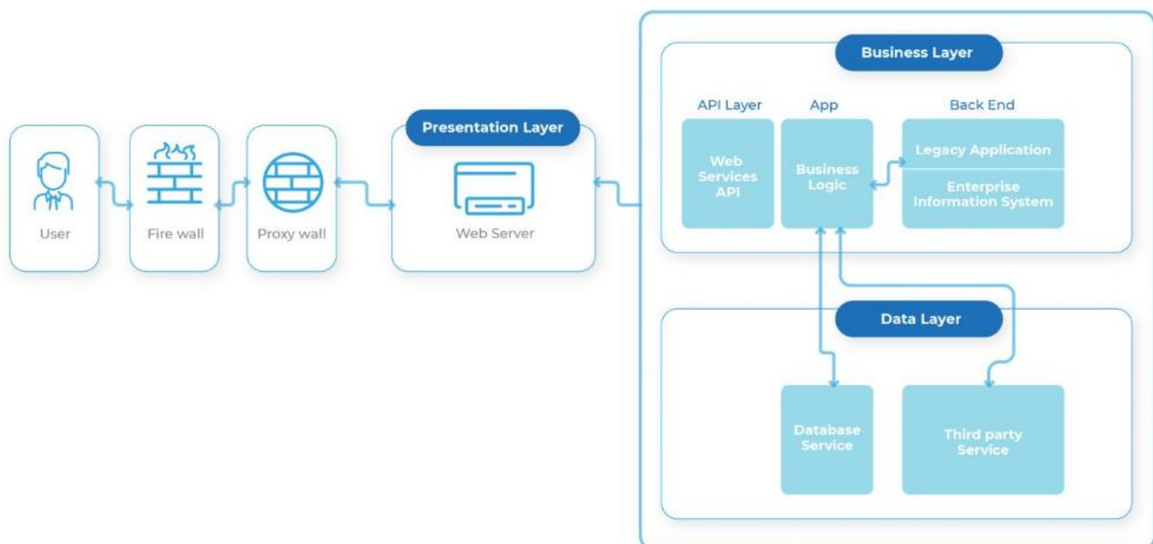

Рисунок 5 – Приклад діаграми для візуалізації архітектури

Маючи чітко визначений дизайн системи, проєкт перейшов у фазу реалізації. На цьому етапі дизайн системи був розроблено шляхом перекладу абстрактного дизайну у функціональний код.

Для розробки інтерфейсу було використано React.js, що забезпечило динамічний та зручний інтерфейс, а поєднання Node.js та Express.js забезпечило надійну функціональність бекенд-частини. JSON Web Tokens (JWT) були реалізовані для безпечної автентифікації користувачів, забезпечуючи надійні механізми контролю доступу [5].

Після завершення етапу впровадження система проходить етап комплексного тестування. Цей етап необхідний для того, щоб переконатися, що система відповідає попередньо визначеним вимогам.

Тестування буде включати обширне модульне тестування з використанням Jest для перевірки функціональності окремих компонентів, а також тестування API з використанням Postman для перевірки операцій на стороні сервера. Очікується, що цей ретельний процес тестування допоможе виявити та вирішити потенційні проблеми до розгортання, що значно підвищить надійність та стійкість системи.

Заключним етапом проєкту є розгортання, після ретельного тестування і затвердженння, застосунок стає доступним для кінцевих користувачів. Amazon Web Services (AWS) було обрано як найбільш підходящу платформу для розгортання застосунка завдяки її масштабованості, надійності та надійним функціям безпеки. Docker буде використовуватися для контейнеризації, забезпечуючи узгоджене і контрольоване середовище для застосунка, таким чином зменшуючи розбіжності між етапами розробки і розгортання.

Слід зазначити, що кожна фаза проєкту, від аналізу вимог до розгортання, відіграває важливу роль в успішному впровадженні застосунку для управління власними фінансами. Запланований методичний підхід дозволить проєкту ефективно орієнтуватися в складнощах розробки системи з метою створення ефективного, надійного та зручного застосунку, який відповідатиме різноманітним потребам користувачів.

3 ОПИС РЕАЛІЗАЦІЇ ПРОЄКТНОГО РІШЕННЯ

3.1 Опис розробленої інформаційної системи

Дана інформаційна система (ІС) є проявом сучасних технологічних досягнень, вміло використаних для задоволення сучасних потреб в управлінні особистими фінансами. Функціональність та простота у використанні поєднуються в ньому, що робить його привабливим для широкого кола користувачів. Перевага ІС для управління фінансами полягає в його широкому спектрі можливостей і функціональності.

Ретельно розроблені та створені для вирішення всіх аспектів управління особистими фінансами, ці функції спрямовані на надання комплексного рішення для кінцевого користувача.

Взаємодія користувача із ІС починається з обов'язкового процесу автентифікації користувача. Вона складається із зручних для користувача сторінками входу та реєстрації, ця система гарантує, що тільки автентифіковані користувачі отримують доступ до власних облікових записів. На сьогоднішній день конфіденційність має першочергове значення, ця функція не тільки забезпечує персоналізований досвід, але й гарантує безпеку та конфіденційність фінансової інформації користувачів.

Після успішної автентифікації користувачі потрапляють до командного центру своїх фінансів – Панелі управління рахунком. Ця інформаційна панель, ретельно продумана для зручності використання, в свою чергу вона відображає фінансовий стан користувача в режимі реального часу. Завдяки чіткій інформації про залишок на рахунку та стрічці останніх транзакцій – вона надає користувачам необхідну інформацію для прийняття обґрунтованих фінансових рішень.

Система управління транзакціями в цій ІС виходить за рамки простого ведення обліку.

Користувачі мають привілеї динамічно реєструвати і класифікувати свої транзакції, що допомагає вести впорядкований облік своїх доходів і витрат. Кожна записана транзакція збагачується конкретними деталями, такими як тип транзакції, сума та відповідна категорія. Така ретельно деталізована система обліку допомагає користувачам виявляти закономірності у своїх витратах.

ІС використовує можливості візуалізації даних для перетворення складних фінансових даних на інтерактивні, легкі для розуміння діаграми. Відображаючи розподіл доходів і витрат користувачів за різними категоріями, функція візуалізації даних спрощує фінансовий аналіз. Вона дозволяє користувачам отримати чітке уявлення про свій фінансовий стан, сприяючи ефективному фінансовому плануванню.

Керуючись принципами мінімалістичного дизайну, користувацький інтерфейс ІС обіцяє користувацький досвід, який є настільки ж плавним, як і візуально привабливим. Інтерфейс гарантує, що кожна взаємодія користувача з ІС є цілеспрямованою, ефективною та збагачуючою. Кожна сторінка в ІС слугує певній меті, забезпечуючи впорядковану подорож користувача. Навігація інтуїтивно зрозуміла, візуальні підказки чіткі, а текстові описи точні. Така філософія дизайну балансує між простотою та ефективністю, сприяючи створенню інтерфейсу, який користувачі вважають простим у використанні та естетично привабливим.

ІС сумісний з більшістю сучасних веб-браузерів, включаючи, але не обмежуючись Google Chrome, Firefox, Safari та Microsoft Edge. Однак стабільне підключення до Інтернету є обов'язковою вимогою для безперебійної роботи програми та надання користувачам безперервного досвіду.

ІС використовує можливості стеку MERN, популярного повностекового JavaScript-рішення. MongoDB слугує основою для зберігання даних, забезпечуючи гнучкість для задоволення мінливих вимог до зберігання даних.

Операціями на стороні сервера допомогають керувати Express.js та Node.js, які обробляють маршрутизацію та HTTP-запити, забезпечуючи плавні та швидкі відповіді сервера. Бібліотека React.js, допомагає створювати динамічні та цікаві користувацькі інтерфейси, вона забезпечує взаємодію з клієнтом у ІС [6].

ІС пройшов процедури тестування з акцентом на забезпечення високого рівня надійності. Комплексні стратегії, включаючи модульне тестування, інтеграційне тестування та системне тестування, були використані для перевірки надійності ІС та його здатності працювати в різних умовах. Ці тести відіграли важливу роль у процесі розробки, допомагаючи виявляти та вирішувати потенційні проблеми і сприяючи створенню надійного, ефективного та зручного інструменту управління фінансами.

Таким чином, ІС є прикладом того, як технології можуть бути використані для створення практичних рішень, які є не тільки ефективними та надійними, але й естетично привабливими та простими у використанні. Це комплексне рішення для управління особистими фінансами, розроблене з урахуванням потреб кінцевого користувача.

3.2 Аналіз архітектури серверної сторони

Важливу роль під час розробки ІС відіграє налаштування бекенду, який слугує фундаментальною основою веб-додатку. Бекенд відповідає за управління системними даними, обробку запитів користувачів та забезпечення безперебійного зв'язку між клієнтською та серверною частинами (рис.6).

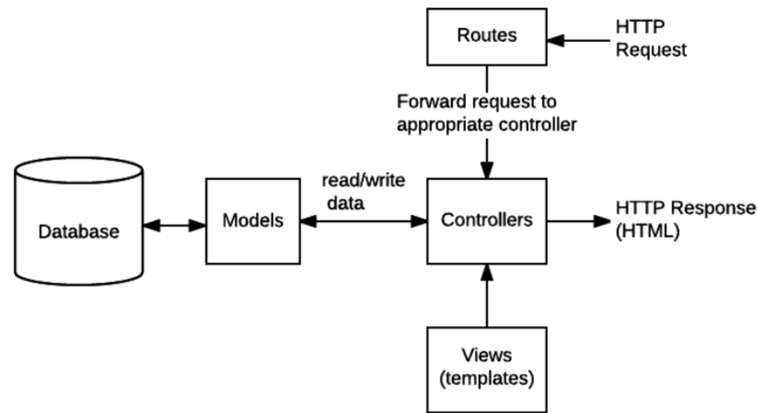

Рисунок 6 – Основний потік даних і те, що потрібно реалізувати при обробці HTTP-запиту/відповіді

Для реалізації бекенду цього проєкту використовується Node.js завдяки його високій продуктивності в обробці операцій на стороні сервера. Для налаштування сервера використовується Express.js, мінімалістичний фреймворк веб-додатків для Node.js. Процес починається зі встановлення необхідних пакетів Node.js та Express.js за допомогою Node Package Manager (NPM) [3]. Далі створюється фреймворк Express.js і налаштовується на прослуховування вказаного порту, таким чином налаштовуючи сервер. Варто зазначити, що фреймворк Express.js і сервер, на якому він працює, є основою бекенду. Відповідний фрагмент коду файлу server.js представлено нище [7].

```
const path = require('path')
const express = require('express')
const dotenv = require('dotenv').config()
const connectDB = require('./config/db')
const port = process.env.PORT || 5000
connectDB()
const app = express()
app.use(express.json())
app.use(express.urlencoded({ extended: false }))
app.use('/api/transactions',          require('./routes/transac-
tionRoutes'))
app.use('/api/seed', require('./routes/seedRoutes'));
app.use('/api/users', require('./routes/userRoutes'));
```

```
// Serve frontend
if (process.env.NODE_ENV === 'production') {
  app.use(express.static(path.join(__dirname,
'../frontend/build')))
  app.get('*', (req, res) =>
    res.sendFile(
      path.resolve(__dirname, '../', 'frontend', 'build', 'in-
dex.html')
    )
  )
} else {
  app.get('/', (req, res) => res.send('Please set to production'))
}
app.listen(port, () => console.log(`Server started on port
${port}`))
```

3.2.1 Опис процесу підключення до бази даних

Після успішної ініціалізації бекенду наступним важливим кроком є підключення до MongoDB. MongoDB – це документно-орієнтована база даних NoSQL, яка пропонує високу продуктивність, високу доступність і легку масштабованість. Вона працює на основі концепції колекцій і документів, забезпечуючи гнучку, JSON-подібну модель даних, яка дозволяє зберігати і обробляти дані різної структури.

Підключення до MongoDB здійснюється за допомогою бібліотеки Mongoose, моделювальника об'єктів MongoDB для Node.js. Mongoose надає просте, засноване на схемах рішення для моделювання даних і включає вбудовані засоби приведення типів, валідації, створення запитів і хуки бізнес-логіки.

Налаштування з'єднання з MongoDB передбачає імпорт бібліотеки Mongoose, визначення URL-адреси MongoDB (яка вказує на розташування бази даних MongoDB) та використання методу connect Mongoose для встановлення з'єднання. Якщо з'єднання встановлено успішно, на консоль буде виведено відповідне повідомлення [8].

Деталі налаштування MongoDB, створення схем і моделей, а також створення запитів за допомогою Mongoose є критично важливими компонентами, які суттєво впливають на функціональність і ефективність програми. Деталі підключення до MongoDB та подальші кроки можна побачити в наступному фрагменті коду db.js файлу:

```
const mongoose = require('mongoose');
mongoose.set('strictQuery', true);
const connectDB = async () => {
  try {
    const conn = await mongoose.connect(process.env.MONGO_URI);
    console.log(`MongoDB Connected: ${conn.connection.host}`);
  } catch (err) {
    console.log(`Error: ${err.message}`);
    process.exit(1);
  }
}
module.exports = connectDB;
```

Фрагмент коду файлу .env:

```
MONGO_URI = mongodb+srv://shin:PsSuB3F42GD13PGM@cluster0.v45ttzf.
mongodb.net/?retryWrites=true&w=majority
```

3.2.2 Опис маршрутів та обробників маршрутів

Ключовим елементом архітектури фінансової ІС є каталог маршрутів, який містить скрипти, що визначають кінцеві точки API та пов'язану з ними функціональність. Кожен файл в каталозі routes відповідає окремому маршруту в ІС і містить обробники маршрутів для різних методів HTTP-запитів.

Файл seedRoutes.js використовується для заповнення бази даних початковими даними. Файл експортує екземпляр Express Router з одним маршрутом – POST-маршрутом на кінцеву точку '/seeduser'. Обробник маршруту, пов'язаний з цією кінцевою точкою, createRandomUserAcc, імпортується з файлу seedController в каталозі контролерів.

Коли відбувається POST-запит до '/seeduser', викликається функція createRandomUserAcc, яка створює випадкові облікові записи користувачів для тестування. Відповідний код виглядає наступним чином:

```
const express = require('express')
const router = express.Router()
const {
   createRandomUserAcc
} = require('../controllers/seedController')

router.post('/seeduser', createRandomUserAcc)
module.exports = router
```

Файл transactionRoutes.js використовується для управління маршрутами, пов'язаними з транзакціями. Цей маршрутизатор експортує кілька кінцевих точок, пов'язаних з різними методами HTTP (GET, POST, DELETE, PUT), кожна з яких пов'язана з відповідними обробниками: getTransactions, addTransaction, updateTransaction, deleteTransaction. Ці обробники, імпортовані з 'transactionController', виконують потрібну операцію в залежності від типу вхідного запиту. Всі ці маршрути є захищеними, тобто вимагають автентифікації користувача, що забезпечується функцією protect middleware:

```
const express = require('express');
const router = express.Router();
const { getTransactions, addTransaction, updateTransaction, deleteTransaction } = require('../controllers/transactionController');
const { protect } = require('../middleware/authMiddleware')

router
.route('/')
.get(protect, getTransactions)
.post(protect, addTransaction);

router
.route('/:id')
.delete(protect, deleteTransaction)
.put(protect, updateTransaction);
```

```
module.exports = router;
```

Файл userRoutes.js керує маршрутами та операціями, пов'язаними з користувачем.

З цього маршрутизатора експортуються три кінцеві точки: POST-маршрут на '/' для реєстрації нових користувачів (registerUser), POST-маршрут на '/login' для входу користувача (loginUser) і GET-маршрут на '/me', який отримує дані про користувача, що увійшов до системи (getMe). Ці маршрути мають відповідні обробники в 'userController'. Маршрут '/me' теж є захищеним маршрутом, оскільки він вимагає автентифікації користувача:

```
const express = require('express')
const router = express.Router()
const {
    registerUser,
    loginUser,
    getMe,
} = require('../controllers/userController')
const { protect } = require('../middleware/authMiddleware')

router.post('/', registerUser)
router.post('/login', loginUser)
router.get('/me', protect, getMe)
module.exports = router
```

Процес налаштування цих файлів маршрутів і відповідних їм функцій контролерів є невід'ємною частиною реалізації функціональності програми. Це дозволяє ІС належним чином реагувати на різні типи HTTP-запитів, таким чином маніпулюючи даними системи і задовольняючи потреби користувача.

Добре структурована маршрутизація покращує ремонтопридатність коду, спрощує процеси налагодження та тестування і, в кінцевому підсумку, сприяє безперебійній роботі користувача.

3.2.3 Опис контролерів та їх ролі в системі

Контролери відіграють важливу роль у програмній архітектурі фінансового веб-додатку. Вони обробляють вхідні запити, взаємодіють з моделями для отримання або маніпулювання даними та надсилають відповідну відповідь клієнту. Директорія контролерів у проєкті містить наступні основні файли.

Файл seedController.js – генерує фальшиві дані користувача для тестування. Контролер містить функцію createRandomUserAcc, яка створює нового користувача з випадково згенерованим ім'ям, електронною поштою та паролем і додає його до бази даних. Функція також створює набір випадкових транзакцій для користувача. Ось короткий фрагмент, що демонструє частину створення користувача:

```
const userName = faker.name.firstName()
const userData = {
  name: userName,
  email: faker.internet.email(userName),
  password: bcrypt.hashSync('123456',12),
};
const user = await UserSchema.create(userData);
```

Файл transactionController.js обробляє операції, пов'язані з фінансовими транзакціями. Функція getTransactions отримує всі транзакції для певного користувача, а функція addTransaction створює нову транзакцію. Ось фрагмент, який показує, як створюється нова транзакція:

```
const { account , text, amount , note } = req.body;
const newTransaction = await Transaction.create({
  account: account,
  category: text,
  amount: amount,
  note: note,
  user: req.user.id
});
```

Файл userController.js керує операціями, пов'язаними з користувачами, такими як реєстрація нового користувача. У функції registerUser він спочатку перевіряє вхідні дані, потім хешує пароль і створює нового користувача в базі даних. Ось фрагмент, що демонструє частину хешування пароля:

```
// Hash password
const salt = await bcrypt.genSalt(10)
const hashedPassword = await bcrypt.hash(password, salt)
```

Фрагменти коду, ілюструють роботу кожного контролера. Контролери допомагають керувати потоком даних програми, гарантуючи, що кожен запит обробляється точно і ефективно.

3.2.4 Опис та аналіз моделей даних

Моделі є фундаментальними архітектурними компонентами веб-додатків, оскільки вони визначають структуру даних для бази даних MongoDB. Вони відповідають за структурування та формування інформації, яку отримує та передає ІС, дозволяючи контролювати та маніпулювати даними відповідно до вимог. У цьому проєкті використовуються дві основні моделі: транзакційна та користувацька.

Файл transactionModel.js описує структуру даних для транзакцій. Кожна транзакція складається з таких елементів: тип рахунку, сума, категорія, примітка, дата транзакції та ідентифікатор користувача. Ця структура забезпечує узгодженість даних, якими ми керуємо, і дає нам передбачувану схему для транзакцій.

Розглянемо наступний фрагмент коду, який демонструє схему транзакції:

```
const transactionSchema = new mongoose.Schema(
  {
  account: {
```

```
        type: String,
        required: [true, 'Please add account type']
      },
      amount: {
        type: Number,
        required: [true, 'Please add a positive or negative number']
      },
      ...
    }
    );
    module.exports = mongoose.model('Transaction', transac-
tionSchema);
```

Цей фрагмент є ядром моделі транзакцій, визначаючи первинні параметри даних і пов'язані з ними вимоги. Викликаючи mongoose.model(), відбувається генерація моделі транзакцій і експорт її для використання в інших частинах програми.

Файл userModel.js визначає структуру даних для користувачів. Кожен користувач має ім'я, електронну пошту та пароль. Забезпечення послідовної та уніфікованої структури для всіх користувачів спрощує операції з даними користувачів і підвищує надійність пов'язаних з ними дій. Нижче наведено фрагмент коду, який визначає схему користувача:

```
const userSchema = new mongoose.Schema(
  {
    name: {
      type: String,
      required: [true, 'Please add a name'],
    },
    email: {
      type: String,
      required: [true, 'Please add an email'],
      unique: true,
    },
    password: {
      type: String,
      required: [true, 'Please add a password'],
    },
  },
```

```
)
module.exports = mongoose.model('User', userSchema)
```

Як видно з вищеописаного коду, визначення моделі користувача слідує підходу, подібному до моделі транзакцій, але з полями даних, специфічними для користувача. Ця схема забезпечує необхідні дані для кожного користувача і узгодженість між усіма записами користувачів.

Таким чином, ці моделі формують основу структури даних застосунку, надаючи необхідні схеми для забезпечення узгодженості та надійності даних.

3.3 Аналіз архітектури клієнтської сторони

3.3.1 Опис та характеристика компонентів системи

Інтерфейс користувача ІС побудований з використанням універсальної природи компонентів у React, а також потужних можливостей управління станами, що надаються Redux. Компоненти в React слугують самодостатніми, багаторазовими одиницями коду, що інкапсулюють сегменти користувацького інтерфейсу (UI) та оптимізують загальну структуру ІС. Завдяки Redux ІС може ефективно керувати та оновлювати свій стан, забезпечуючи безперебійний потік даних між компонентами.

Така організація та інтеграція компонентів і управління станом Redux сприяють створенню добре структурованого репозиторію компонентів програми (див.рис.7).

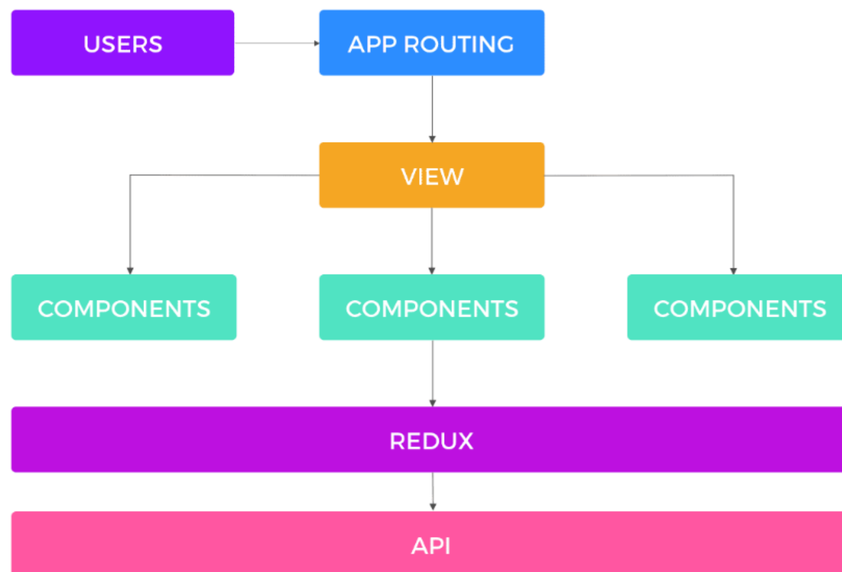

Рисунок 7 – Потік додатку на React.js

Певні ключові компоненти відіграють ключову роль у функціюванні програми, до них відносяться:

– “AddTransaction.js”;
– “Charts.js”;
– “IncomeExpense.js”;
– “Transaction.js”.

Для глибшого розуміння ролі згаданих файлів у системі, необхідно ретельно розглянути специфіку роботи кожного компонента.

Файл AddTransaction.js дозволяє користувачам додавати нові транзакції в систему. Компонент використовує useState, невід'ємну властивість React, щоб керувати локальним станом для декількох факторів, включаючи тип рахунку, суму, категорію та примітку.

Ця система управління станами забезпечує адаптивний користувацький інтерфейс, який миттєво реагує на зміни.

Коли користувач надсилає форму, створюється новий об'єкт транзакції, який надсилається до репозиторію Redux за допомогою addTransaction.

Нижче представлено фрагмент коду компонента AddTransaction:

```
export const AddTransaction = () => {
  const dispatch = useDispatch()
  const [text, setText] = useState('');
  const [account, setAccount] = useState(0);
  const [amount, setAmount] = useState(0);
  const [note, setNote] = useState('');
  const [open, setOpen] = useState(false);
  const onSubmit = e => {
    e.preventDefault();
    const newTransaction = {
      account,
      text,
      amount: +amount,
      note:note
    }
    dispatch(addTransaction(newTransaction));
  }
}
```

У цьому компоненті useState організовує локальний стан полів форми, а useDispatch використовується для відправки дій до сховища Redux. Функція onSubmit створює новий об'єкт транзакції і відправляє його до сховища за допомогою addTransaction, ефективно забезпечуючи взаємодію з користувачем у реальному часі.

Компонент charts.js використовується для представлення фінансових даних у візуальному та інтерактивному форматі. Компонент реалізує useSelector для отримання даних про транзакції зі сховища Redux. Потім компонент форматує ці дані для сумісності з бібліотекою Google Charts. Наступний фрагмент коду з файлу charts.js ілюструє логіку форматування даних:

```
export const TransactionCharts = () => {
    const { transactions } = useSelector(state => state.transac-
tions)
```

```
    const [data, setData] = useState([]);
    const [edata, setEdata] = useState([]);
    const [idata, setIdata] = useState([]);

    useEffect(() => {
        var category = {}
        ...
        for (const item in category) {
            value.push([item,category[item][0],category[item][1]])
            value2.push([item, category[item][1]])
            value3.push([item, category[item][0]])
        }
        setData(value)
        setIdata(value3)
        setEdata(value2)
    }, [transactions])
    ...
}
```

Ця функція ефективно перетворює дані транзакції у формат, придатний для бібліотеки візуалізації діаграм. Вона використовує useSelector для доступу до даних транзакції та useState для керування станом даних, що візуалізуються.

Цей компонент в першу чергу відповідає за відображення загальних доходів і витрат для користувача. Використовуючи useSelector, він отримує масив транзакцій зі сховища Redux і обчислює загальний дохід і витрати шляхом ітерації по масиву і підсумовування сум. Фрагмент з файлу IncomeExpense.js, що ілюструє це, виглядає наступним чином:

```
export const IncomeExpense = () => {
       const { transactions } = useSelector(state => state.transac-
tions);
       const amounts = transactions.map(transaction => transaction);

       const income = amounts
         .filter(item => item.account === 'income')
         .reduce((acc, item) => (acc += item.amount), 0)
         .toFixed(2);

       const expense = (
```

```
    amounts.filter(item   =>   item.account   ===   'expense').re-
duce((acc, item) => (acc += item.amount), 0) *
    -1
  ).toFixed(2);
  ...
}
```

У цьому компоненті ефективно використовуються методи filter і reduce для обчислення загального доходу і витрат.

Файл Transaction.js відповідає за відображення окремих транзакцій та надання інтерфейсу для маніпуляцій з ними. Користувачі можуть видаляти або редагувати транзакції за допомогою кнопок, пов'язаних з кожною транзакцією. Дія deleteTransaction виконується, коли ініціюється операція видалення, в результаті чого транзакція видаляється зі сховища.

Ось фрагмент коду компонента Transaction.js:

```
export const Transaction = ({ transaction }) => {
  const dispatch = useDispatch()
  const sign = transaction.account === 'expense' ? '-' : '+';
  const [modalIsOpen, setIsOpen] = useState(false);
  const openModal = () => {
    setIsOpen(true);
  }
  const closeModal = () => {
    setIsOpen(false);
  }
}
```

Цей компонент використовує useState для керування видимістю модального вікна, що використовується для редагування транзакцій. При взаємодії з кнопкою видалення виконується дія deleteTransaction, яка видаляє відповідну транзакцію з магазину.

Таким чином, інтерфейс ІС є модульним, з різними компонентами, що відповідають за різні аспекти користувацького інтерфейсу. Ефективно використовуючи можливості React та Redux, ІС підтримує високий рівень чуйності та інтерактивності, забезпечуючи бездоганний користувацький досвід.

3.3.2 Опис сервісного рівня

У надійній фінансовій програмі управління транзакціями є фундаментальним. Цей процес вимагає виконання різноманітних операцій, таких як створення, отримання, оновлення та видалення транзакцій. Щоб полегшити ці операції, проєкт використовує два основні компоненти: файл transactionService.js і файл transactionSlice.js. Ці файли служать різним цілям, але гармонійно працюють разом, щоб забезпечити безперебійну роботу користувача і зберегти цілісність даних.

Файл transactionService.js, що є частиною сервісного рівня, відіграє центральну роль у забезпеченні зв'язку з бекендом. Він робить HTTP-запити до відповідних кінцевих точок, причому кожна функція відповідає певній CRUD-операції.

```
// transactionService.js
// Create new transaction
const addTransaction = async (transactionData, token) => {
  // HTTP request configuration
  const response = await axios.post(API_URL, transactionData, con-
fig);
  return response.data
}

// Get user transactions
const getTransactions = async (token) => {
  // HTTP request configuration
  const response = await axios.get(API_URL, config)
  return response.data
}
```

Інкапсулюючи запити API в ці сервісні функції, підтримується читабельність коду, забезпечується легку обробку помилок і дотримується принципів абстракції та модульності.

Однак сервісні операції хоч і полегшують комунікацію з бекендом, але потрібен механізм для обробки стану на стороні клієнта, щось що може відстежувати. Наприклад, поточні транзакції або те, чи виконується операція транзакції в даний момент, чи призвела вона до помилки. Саме тут з'являється файл transactionSlice.js, який відповідає за управління станом транзакцій Redux.

```
// transactionSlice.js
// Create new transaction
export const addTransaction = createAsyncThunk(
  'transactions/add',
)
// Initial state and reducers
export const transactionSlice = createSlice({
  name: 'transaction',
  initialState,
  })
```

Цей фрагмент слідує шаблону Redux Toolkit, надаючи дії для відправки і редуктор для обробки оновлень стану на основі цих дій. Він інтегрує операції transactionService.js і відправляє їх як асинхронні потоки. Це означає, що він може обробляти повний життєвий цикл операції: відправляти дії перед початком операції, після її успішного завершення і в разі помилки. У редукторі вони представлені у вигляді станів "очікує", "виконано" та "відхилено".

І transactionService.js, і transactionSlice.js відіграють ключову роль в архітектурі проєкту:

- забезпечення цілісності даних: односпрямований потік даних і чіткі переходи станів дозволяють легко відстежувати стан застосунку в будь-який час (такий рівень передбачуваності є перевагою при роботі з фінансовими даними, де точність і надійність мають першорядне значення);
- розподіл обов'язків: кожен файл має власну сферу відповідальності – transactionService.js відповідає за зв'язок з бекендом, а transactionSlice.js керує станом на стороні клієнта (таке розділення покращує читабельність коду, що полегшує підтримку та розширення проєкту);
- масштабованість: ця архітектура створена для зростання (у міру розвитку ІС, додавання нових функцій та операцій передбачає створення нових сервісних функцій та розширення зрізу Redux – без необхідності серйозної переробки існуючого коду).

Слід зауважити, transactionService.js і transactionSlice.js складають основу системи управління транзакціями. Вони являють собою поєднання різних принципів і практик розробки програмного забезпечення, що призвело до створення надійної, зручної для обслуговування і масштабованої архітектури.

3.3.3 Опис структури сторінок інформаційної системи

У сучасних фінансових програмах управління та контроль різних функціональних можливостей, призначених для користувача, здебільшого зосереджені в окремих, спеціально створених модулях. Одним з таких основних модулів у ІС є папка "Сторінки". Ця папка є важливим гвинтиком у механізмі ІС, керуючи безліччю переглядів, які формують користувацький досвід. У цій папці знаходяться важливі файли Login.js, Register.js і Dashboard.js, кожен з яких вносить свій унікальний внесок у функціональність ІС.

Використовуючи можливості React, Redux та різноманітних кастомних і сторонніх компонентів, ці файли слугують для побудови плавної та безперервної подорожі користувача.

Шлюз автентифікації Login.js – це портал, через який користувачі отримують доступ до ІС. Ретельне налаштування управління станом керує введенням даних користувачем, а useState відповідає за локальне зберігання стану. З сильним акцентом на підтримці безперебійного потоку, створений механізм відправляє дії Redux для входу користувача і ефективно переходить на головну сторінку після успішної автентифікації.

```
// Excerpt from Login.js
const { email, password } = formData;
const navigate = useNavigate();
const dispatch = useDispatch();

useEffect(() => {
  if (isSuccess || user) {
    navigate('/')
  }
  dispatch(reset())
}, [user, isSuccess, navigate, dispatch])
dispatch(login(userData))
```

Цей фрагмент показує витяг даних форми з локального стану, відправку дії login() і ретельну обробку перенаправлення в разі успішного входу (рис.8).

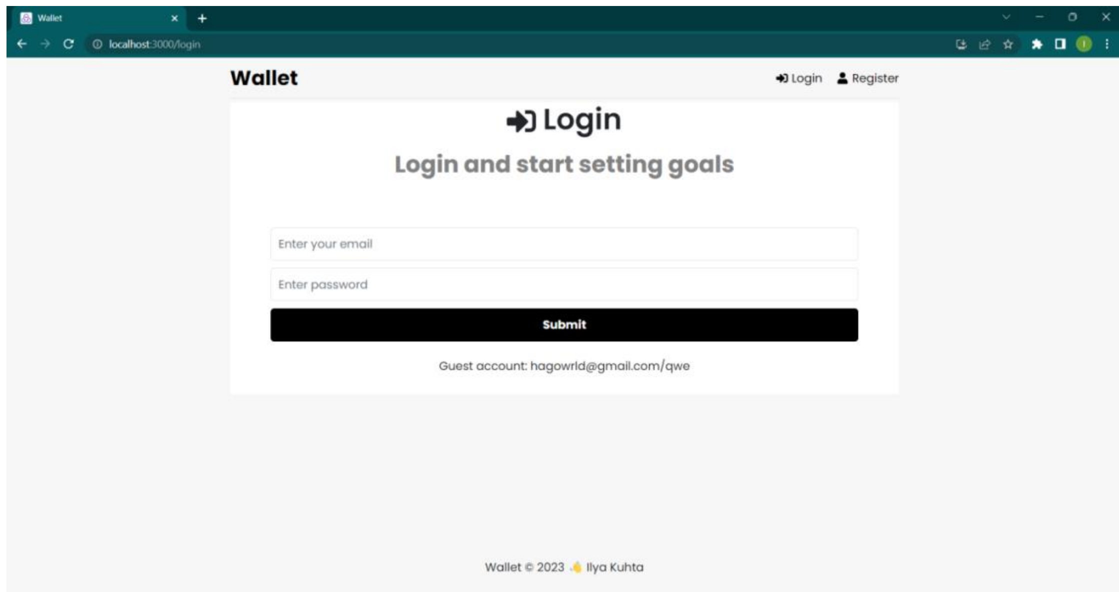

Рисунок 8 – Сторінка Log in

Спираючись на фундамент, закладений Login.js, Register.js робить ще один крок вперед, вітаючи нових користувачів і проводячи їх через процес реєстрації.

Файл Register.js розширює можливості свого попередника, додаючи поле для підтвердження пароля, ретельно перевіряючи, що паролі, введені користувачем, ідеально збігаються, перш ніж виконати реєстраційну дію (див.рис.9).

```
// Excerpt from Register.js
const { name, email, password, password2 } = formData;
const navigate = useNavigate();
const dispatch = useDispatch();

if (password !== password2) {
  toast.error('Passwords do not match')
} else {
  const userData = { name, email, password };
  dispatch(register(userData))
}
```

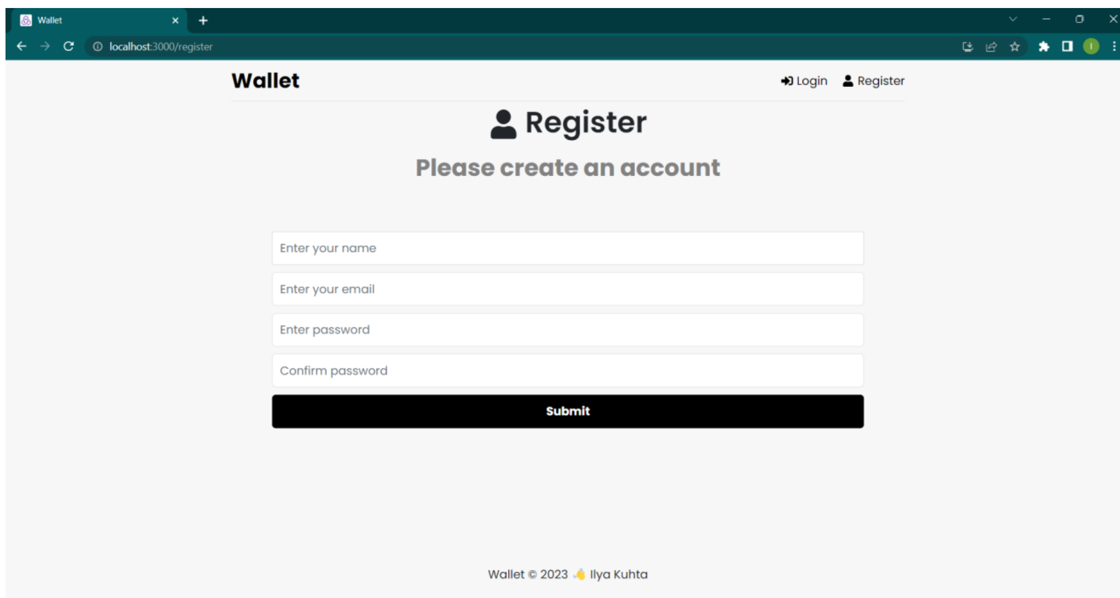

Рисунок 9 – Сторінка Register

Виділений сегмент коду підкреслює важливість підтвердження пароля під час реєстрації користувача і демонструє процес залучення користувача через дії Redux.

Dashboard.js, як основний інтерфейс ІС, з'являється після успішної аутентифікації користувача. Цей файл виконує роль центру ІС, інтегруючи кілька кастомних компонентів, таких як Баланс, Доходи-Витрати, TransactionList і AddTransaction (рис.10) та розділ Charts (рис.11).

```
useEffect(() => {
  if (!user) {
    navigate('/login')
  }
  dispatch(getTransactions())
}, [user, navigate, dispatch])
if (isLoading) {
  return <Spinner />
}
{
  chart && <TransactionCharts  />
}
```

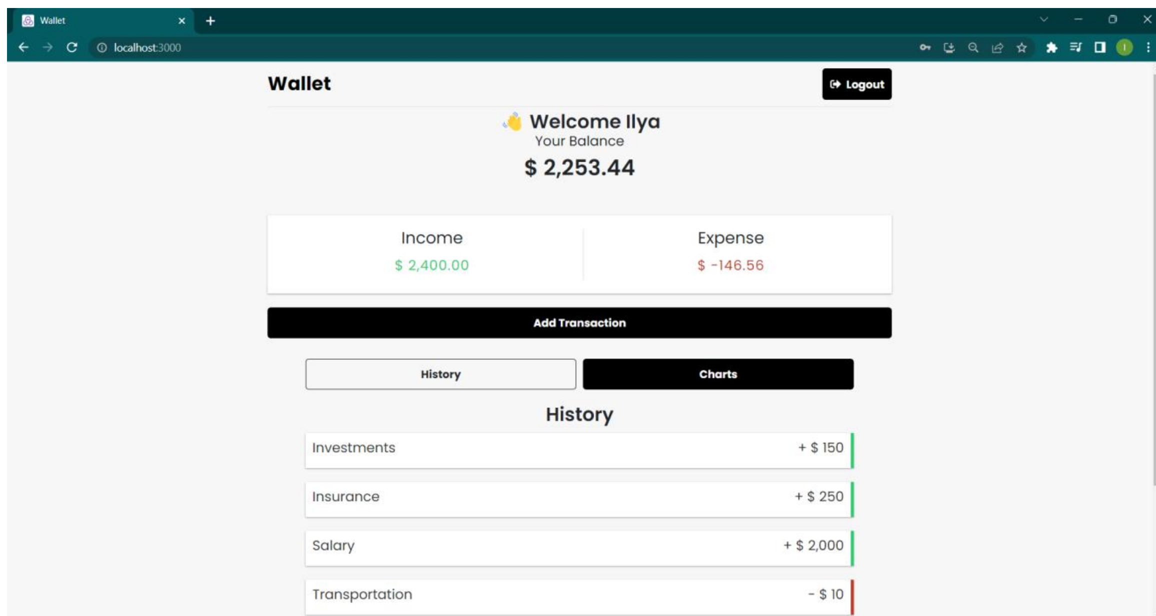

Рисунок 10 – Сторінка Dashboard

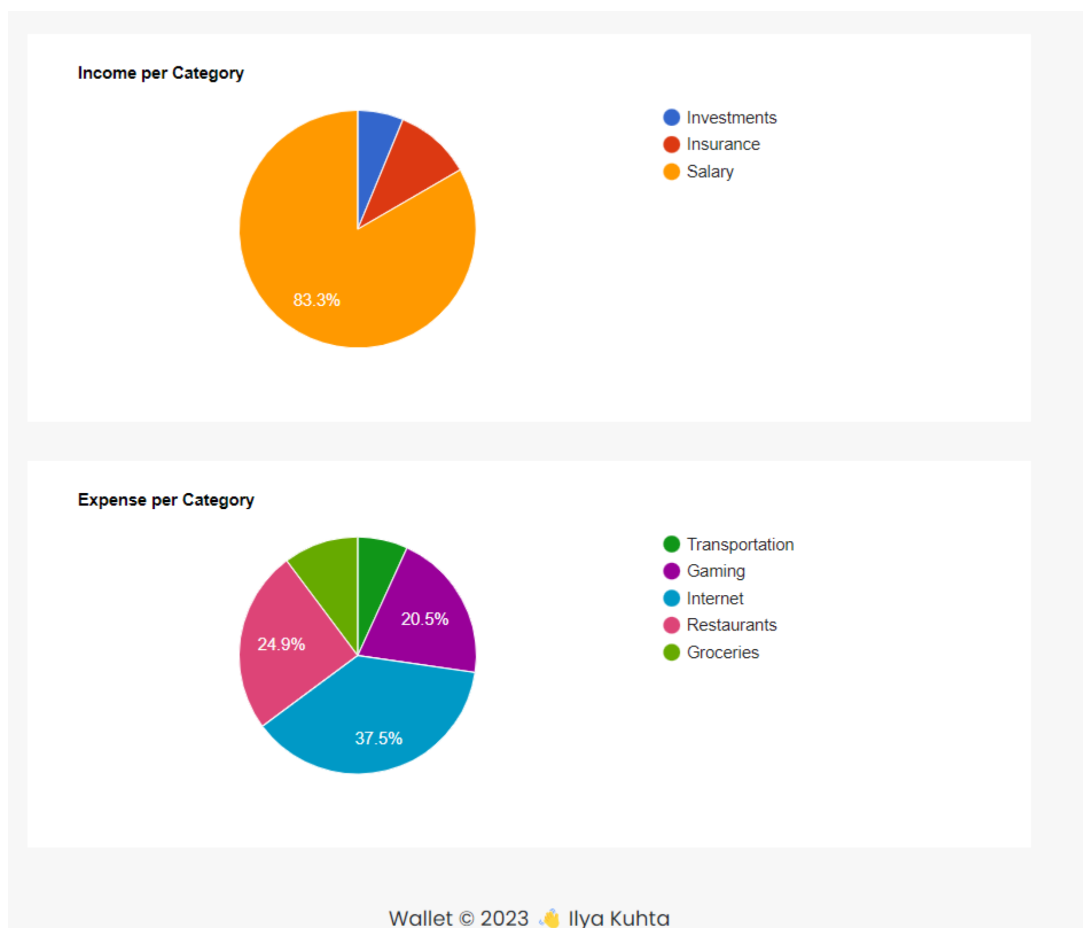

Рисунок 11 – Розділ Charts

useEffect у цьому фрагменті призначений для запуску дії getTransactions при завантаженні сторінки і перенаправлення на вхід, якщо немає автентифікованих даних користувача. Стан isLoading визначає стан відображення компонента завантаження спіннера. Згодом, вибір діаграми транзакцій або списку транзакцій визначає вигляд відображення, що додає інтерактивності додатку.

Мінімалістичний дизайн ІС з основним акцентом на дані забезпечує чистий, візуально привабливий інтерфейс. Простота навігації має першорядне значення, що сприяє ефективному виконанню завдань користувачами.

3.4 Аналіз структури бази даних

Структура бази даних є ключовим елементом, який має значний вплив на розробку та ефективність роботи будь-якої програми. Вона вимагає ретельної уваги до кількох аспектів, таких як тип бази даних, дизайн схеми та складні взаємозв'язки між різними об'єктами. Ці елементи об'єднуються для покращення функціональності, ефективності та масштабованості ІС.

У цьому проєкті в якості основи системи зберігання даних було обрано MongoDB. Як документно-орієнтована база даних NoSQL, MongoDB пропонує переваги гнучкості, масштабованості та структури без схем – ідеальний вибір для динамічних ІС.

3.4.1 Опис структури даних користувача

Важливою сутністю при проєктуванні схеми ІС є сутність користувача.

Нижче представлено детальний опис структури та обґрунтування кожного поля:

– ‘_id’: це унікальний ідентифікатор, який автоматично генерується MongoDB, щоб гарантувати, що кожен користувач може бути однозначно ідентифікований в системі;
– ‘name’: це поле зберігає ім'я користувача і є важливою частиною персоналізованого користувацького досвіду;
– ‘email’: поле зберігає адресу електронної пошти користувача (вона використовується не тільки як засіб зв'язку, але й для унікальної ідентифікації);
– ‘password’: паролі користувачів зберігаються у вигляді хешів для додаткової безпеки (цей метод гарантує, що необроблені дані пароля користувача ніколи не існують безпосередньо в базі даних, що зменшує потенційні вразливості);
– ‘createdAt’ і ‘updatedAt’: ці поля міток часу відстежують створення і останнє оновлення кожного запису користувача (вони надають цінну інформацію про шаблони активності користувачів);
– ‘__v’: це внутрішній ключ версії, який використовується MongoDB для відстеження змін, забезпечення цілісності даних та вирішення конфліктів.

Типове представлення сутності користувача виглядає наступним чином:

```
{
   "_id": "645d2c62115d3cb7eef73e16",
   "name": "shin",
   "email": "usemail@gmail.com",
   "password":"$2a$10$q9S3pySjYOTjkHhdNW/9iu3zDOEQywZWpm6CtE4izot-
   zjxqTtWHv.",
   "createdAt": "2023-05-11T17:56:50.825+00:00",
```

```
    "updatedAt": "2023-05-11T17:56:50.825+00:00",
    "__v": 0
}
```

Сутність Користувач, з її полями та функціями, призначена для забезпечення безпечної та персоналізованої роботи користувачів.

3.4.2 Опис структури транзакційних даних

Щоб забезпечити ефективний інструмент для управління особистими фінансами, ця програма також використовує складну сутність "Транзакція". Вона поділяється на два типи: витрати і доходи. Кожен з цих типів відповідає за унікальний аспект особистих фінансів. Витрати – транзакції за цією класифікацією представляють собою відтік грошей з рахунку користувача. Кожному запису присвоюється категорія, щоб краще зрозуміти і відстежити структуру витрат. Крім того, поле для приміток дозволяє користувачам надавати додатковий контекст. Нижче наведено фрагмент коду видаткової транзакції:


```
{
    "_id": "6474cd2ba79f6ce2d51679df",
    "account": "expense",
    "amount": 55,
    "category": "Internet",
    "note": "HomeNet",
    "user": "6474cce9a79f6ce2d51679d4",
    "incurred_on": "2023-05-29T16:04:59.446+00:00",
    "__v": 0
}
```


Дохід – ці транзакції представляють гроші, що надходять на рахунок користувача. Як і у випадку з витратами, операції з доходами поділяються на категорії для кращого управління фінансами, а поле примітки доступне для додаткової інформації. Нижче представлено фрагмент коду транзакції "Дохід":

```
{
    "_id": "6474cd92a79f6ce2d51679f1",
    "account": "income",
    "amount": 150,
    "category": "Investments",
    "note": "",
    "user": "6474cce9a79f6ce2d51679d4",
    "incurred_on": "2023-05-29T16:06:42.104+00:00",
    "__v": 0
}
```


Кожна транзакція, незалежно від того, чи є вона доходом або витратою, містить важливу інформацію про кожну фінансову подію. Ця інформація включає тип (дохід або витрата), суму, категорію, пов'язаного користувача і позначку часу виникнення (incurred_on). Важливо, що дизайн схеми в MongoDB забезпечує значний ступінь гнучкості.

Додаткові поля можуть бути легко додані до схеми користувача або транзакції в майбутньому, якщо це буде потрібно, без необхідності повної реструктуризації бази даних. Ця здатність адаптуватися і розширюватися гарантує, що ІС буде готовий розвиватися і відповідати мінливим вимогам з часом.

4. РЕКОМЕНДАЦІЇ ЩОДО ВДОСКОНАЛЕННЯ ІС

Враховуючи динамічний характер технологій та вподобань користувачів, дуже важливо оцінити потенційні вдосконалення, які можна впровадити в поточну систему фінансового менеджменту. Така оцінка допоможе зберегти конкурентну перевагу програми та забезпечити її відповідність майбутнім потребам користувачів.

У цьому розділі описано три додаткові функції, які потенційно можуть розширити функціональність програми, а саме: інтеграція з фінансовими послугами, впровадження вбудованого калькулятора та включення конвертера валют. Ці функції будуть розглянуті з точки зору їх ключових концепцій, потенційних переваг та міркувань щодо майбутньої інтеграції.

Синхронізація з фінансовими послугами: для подальшої автоматизації ІС можна передбачити синхронізацію з іншими фінансовими послугами, такими як банківські рахунки, кредитні картки та цифрові платіжні сервіси. Цей зв'язок прокладе шлях до точного відображення фінансового стану користувача в режимі реального часу, зменшуючи таким чином ймовірність помилок, що виникають при ручному введенні даних. Таке розширення змінить спосіб взаємодії користувачів зі своїми фінансовими даними. Завдяки прямій синхронізації з фінансовими установами ІС може надавати консолідоване та актуальне уявлення про транзакції, залишки на рахунках та іншу відповідну фінансову інформацію.

Реалізація цієї функції вимагатиме стратегічного партнерства з банками або інтеграції з агрегаторами фінансових даних, які пропонують надійні API. Критично важливим у цьому процесі є суворе дотримання стандартів конфіденційності та безпеки даних, щоб забезпечити відповідність міжнародним нормам, таким як GDPR та PCI DSS [9].

Інтеграція калькулятора в ІС необхідна для того, щоб забезпечити безперебійну роботу користувача, інтеграція калькулятора в програму має практичну перевагу. Ця функція усуває необхідність перемикатися між ІС під час виконання обчислень, таким чином збагачуючи взаємодію користувача з програмою фінансового менеджменту. Фактор зручності лежить в основі надання цієї функції. Інтеграція калькулятора в програму дозволяє користувачам виконувати фінансові розрахунки, пов'язані з транзакціями, бюджетуванням або фінансовим плануванням, не перериваючи робочий процес. Процес інтеграції цієї функції вимагає створення зручного інтерфейсу калькулятора з урахуванням принципів дизайну інтерфейсу користувача (UI) та користувацького досвіду (UX). Водночас, внутрішня частина програми повинна враховувати логіку розрахунків, що може вимагати розробки алгоритмів для виконання різних арифметичних операцій.

Інтеграція конвертера валют, для мандрівників або тих, хто займається міжнародними операціями, може виявитися корисною.

Ця функція має на меті полегшити конвертацію валют у режимі реального часу, гарантуючи, що користувачі матимуть точне уявлення про свій фінансовий стан у різних валютах. Практична реалізація конвертера валют вимагатиме інтеграції з надійним API обмінного курсу, що надасть додатку доступ до поточних і точних курсів конвертації між різними валютами.

Розробка цієї функції включає розробку інтуїтивно зрозумілого інтерфейсу конвертера та внутрішньої логіки для обробки конвертації валют і взаємодії з API обмінного курсу.

Таким чином, ці заплановані функції роблять значний внесок у розширення поточних можливостей ІС. Вони мають потенціал для підвищення рівня залученості та взаємодії користувачів, тим самим покращуючи перспективи зростання застосунку на конкурентному ринку. Ретельне планування і реалізація цих функцій має першочергове значення для удосконалення застосунку і збереження його ринкових позицій в умовах постійного розвитку технологічного прогресу.

ВИСНОВКИ

В ході виконання кваліфікаційної роботи бакалавра було проведено глибокий аналіз тонкощів відстеження та прогнозування особистих витрат. Дослідження показало, що традиційні методи управління особистими фінансами часто є некоректними та громіздкими, що призводить до неефективного використання часу та неоптимальних результатів. Цей аналіз підкреслив потребу в комплексному, спрощеному та персоналізованому інструменті, який би допоміг керувати фінансовими справами.

Було розроблено комплексний застасунок для управління фінансами. Такі ключові функції, як відстеження витрат, категоризація, бюджетування та прогнозна аналітика, були ретельно інтегровані. Він не лише допоможе спростити управління особистими фінансами, але й дати можливість користувачам стратегічно планувати своє фінансове майбутнє. Справжня цінність ІС полягає в його потенціалі для надання глибоких прогнозів, які, в свою чергу, можуть значно покращити фінансовий добробут його користувачів.

ІС був побудований з використанням сучасного технологічного стеку, що складається з MongoDB, Express.js, React та Node.js. Ці технології відомі своєю масштабованістю, ефективністю та універсальністю, що робить їх перспективною основою для побудови ІС. Такий вибір технологій сприяє швидким циклам розробки, простоті обслуговування та надійному фундаменту, що є важливими елементами для забезпечення довговічності та довгострокової надійності застосунку.

Особливу увагу було приділено структурі бази даних, яка є життєво важливим елементом у створенні будь-якого програмного рішення. Для цього проєкту було використано MongoDB, дуже гнучку та масштабовану документно-орієнтовану базу даних NoSQL. Схема бази даних для сутностей користувачів і транзакцій була розроблена таким чином, щоб інкапсулювати всі необхідні точки даних, забезпечуючи при цьому найвищий рівень безпеки і конфіденційності даних користувачів.

Заглядаючи в майбутнє, можна відзначити ряд цікавих функцій, які можна було б включити у застосунок. Можливість пов'язати ІС з різними фінансовими послугами, такими як банківські рахунки та платіжні сервіси, може призвести до автоматизації процесів, що значно покращить загальний користувацький досвід. Додавання вбудованого калькулятора та конвертера валют надає ще один рівень функціональності для користувачів, особливо тих, хто часто подорожує або має потребу у виконанні численних фінансових розрахунків.

Таким чином, цей проєкт пропонує рішення розповсюдженої проблеми і втілює трансформаційний підхід до управління особистими фінансами.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Соснін О. П., Бондар В. М. Основи програмування на JavaScript. Львів: Видавництво Львівської політехніки, 2018. 448 с.
2. Stubblebine, T. Regular Expression Pocket Reference, 2nd Edition. Sebastopol: O'Reilly Media, 2007. 128 p. URL: https://www.oreilly.com/library/view/regular-expression-pocket/9780596802837/ (дата звернення: 15.03.2023).
3. Мельник М.В., Солдатенко В.М., Луговий В.І. Основи алгоритмізації та програмування: навчальний посібник. К.: Національний авіаційний університет, 2017. 276 с.
4. Gamma, E., Helm, R., Johnson, R., Vlissides, J. Design Patterns: Elements of Reusable Object-Oriented Software. New Jersey: Addison-Wesley Professional, 1994. 395 p. URL: https://www.pearson.com/us/higher-education/program/Gamma-Design-Patterns-Elements-of-Reusable-Object-Oriented-Software/PGM2184.html (дата звернення: 24.03.2023).
5. George Ornbo. Sams Teach Yourself Node.js in 24 Hours. Indianapolis: Sams, 2013. 464 p. URL: https://www.informit.com/store/sams-teach-yourself-node.js-in-24-hours-9780672335952 (дата звернення: 21.03.2023).
6. Eddy W., Arjun C. Learning Node.js: A Hands-On Guide to Building Web Applications in JavaScript, 2nd Edition. New Jersey: Addison-Wesley Professional, 2016. 320 p. URL: https://www.pearson.com/us/higher-education/program/Challa-Learning-Node-js-A-Hands-On-Guide-to-Building-Web-Applications-in-Java-Script-2nd-Edition/PGM279117.html (дата звернення: 17.03.2023).

7. Martin, R. C. Clean Code: A Handbook of Agile Software Craftsmanship. New Jersey: Prentice Hall, 2008. 464 p. URL: https://www.pearson.com/us/higher-education/program/Martin-Clean-Code-A-Handbook-of-Agile-Software-Craftsmanship/PGM63937.html (дата звернення: 22.03.2023).
8. Chodorow, K. MongoDB: The Definitive Guide, 3rd Edition. Sebastopol: O'Reilly Media, 2020. 514 p. URL: https://www.oreilly.com/library/view/mongodb-the-definitive/9781491954461/ (дата звернення: 14.03.2023).
9. Zichermann, G., Cunningham, C. Gamification by Design: Implementing Game Mechanics in Web and Mobile Apps. Sebastopol: O'Reilly Media, 2011. 210 p. URL: https://www.oreilly.com/library/view/gamification-by-design/9781449396753/ (дата звернення: 19.03.2023).