Міністерство освіти і науки України
Університет митної справи та фінансів

Факультет інноваційних технологій
Кафедра комп'ютерних наук та інженерії програмного забезпечення

Кваліфікаційна робота бакалавра

на тему: «Розробка веб-додатку для управління фінансами з використанням Python і Flask»

Виконав: студент групи ІПЗ20-2
Спеціальність 121 «Інженерія програмного забезпечення»
Рудовіл Єгор Андрійович
(прізвище та ініціали)
Керівник: к. ф.-м. н., доц. Лебідь О.Ю.
(науковий ступінь, вчене звання, прізвище та ініціали)
Рецензент: Спеціалізоване управління розробки та супроводження програмного забезпечення Департамент з питань цифрового розвитку, цифрових трансформацій та цифровізації ДМСУ
(місце роботи)
головний державний інспектор відділу розробки програмного забезпечення
(посада)
Бахтін О. В.
(науковий ступінь, вчене звання, прізвище та ініціали)

Дніпро – 2024

АНОТАЦІЯ

Рудовіл Є. А. Розробка веб-додатку для управління фінансами з використанням Python і Flask.

Кваліфікаційна робота на здобуття освітнього ступеня бакалавр за спеціальністю 121 «Інженерія програмного забезпечення». – Університет митної справи та фінансів, Дніпро, 2024.

Кваліфікаційна робота присвячена створенню веб-додатку для управління особистими фінансами, який базується на Python та веб-фреймворку Flask. У роботі розглянуто методи та технології, необхідні для реалізації цього додатку, та визначено вимоги до програмного забезпечення. В процесі роботи виконано проектування, розробку та тестування системи. Результати дослідження мають практичне значення, оскільки вони сприяють автоматизації процесів управління фінансами, підвищуючи ефективність і зручність користування для користувачів.

Розроблена система включає серверну частину, створену на Flask, базу даних SQLite та клієнтську частину, яка використовує HTML, CSS та JavaScript. Веб-додаток дозволяє обліковувати доходи і витрати, планувати бюджет, категоризувати фінансові транзакції та візуалізувати фінансові дані за допомогою інтерактивних графіків, створених за допомогою бібліотеки Chart.js. Система забезпечує високу продуктивність, безпеку та можливість масштабування завдяки використанню сучасних технологій та принципів дизайну інтерфейсу користувача (UI/UX).

Ключові слова: ВЕБ-ДОДАТОК, УПРАВЛІННЯ ФІНАНСАМИ, PYTHON, FLASK, SQLITE, CHART.JS, UI/UX ДИЗАЙН, РОЗРОБКА.

ABSTRACT

Rudovil Y. A. Development of a web application for financial management using Python and Flask.

Bachelor's thesis for obtaining a degree in Software Engineering, specialty 121. – University of Customs and Finance, Dnipro, 2024.

This bachelor's thesis is dedicated to the creation of a web application for personal financial management, based on Python and the Flask web framework. The work examines the methods and technologies required for the implementation of this application and defines the software requirements. The project involved the design, development, and testing of the system. The research results are practically significant as they contribute to the automation of financial management processes, enhancing efficiency and user convenience.

The developed system includes a server-side component built with Flask, a SQLite database, and a client-side component using HTML, CSS, and JavaScript. The web application allows users to track income and expenses, plan budgets, categorize financial transactions, and visualize financial data through interactive charts created with the Chart.js library. The system ensures high performance, security, and scalability due to the use of modern technologies and user interface (UI/UX) design principles.

Keywords: WEB APPLICATION, FINANCIAL MANAGEMENT, PYTHON, FLASK, SQLITE, CHART.JS, UI/UX DESIGN, DEVELOPMENT.

ЗМІСТ

ВСТУП

Фінансова грамотність та ефективне управління особистими фінансами є вкрай важливими аспектами життя сучасної людини. З кожним роком все більше людей усвідомлюють необхідність контролювати свої доходи та витрати, планувати бюджет, здійснювати інвестиції та досягати фінансових цілей. Ефективне управління фінансами допомагає розвинути навички планування, контролю боргів та прийняття обґрунтованих рішень щодо управління коштами. Крім того, зростає популярність активного відстеження доходів та витрат, оскільки це дозволяє краще розуміти фінансові звички та приймати обґрунтовані рішення.

Актуальність дослідження зумовлена швидкозмінним економічним середовищем та фінансовими кризами, які періодично виникають у різних країнах, що робить особисті фінанси все важливішим аспектом життя для багатьох людей. В умовах економічної нестабільності з'являється ще більша потреба у контролі та оптимізації особистих фінансів. Крім того, постійно зростає обсяг фінансових даних, які потребують ретельного аналізу та систематизації для ефективного управління бюджетом. У зв'язку з цим розробка інструментів, які б дозволяли кожному ефективно управляти своїми фінансами, незалежно від рівня фінансової грамотності чи досвіду, набуває особливої актуальності.

Веб-додатки для управління фінансами забезпечують зручний та ефективний спосіб відстежувати доходи та витрати, аналізувати фінансові дані та планувати бюджет. Ці додатки дозволяють користувачам легко вводити інформацію про свої фінансові операції, категоризувати їх та візуалізувати дані у зручному форматі. Використання веб-технологій робить такі додатки доступними з будь-якого пристрою, підключеного до Інтернету, що значно спрощує процес управління фінансами. Крім того, веб-додатки часто

пропонують додаткові функції, такі як встановлення фінансових цілей, відстеження інвестицій та управління боргами.

Об'єкт дослідження: сучасні технології розробки веб-додатків на базі мови програмування Python та веб-фреймворку Flask, які надають користувачам зручний та інтуїтивно зрозумілий веб-інструмент для керування їхніми фінансами.

Предмет дослідження: розробка веб-додатка для управління фінансами на базі мови програмування Python та веб-фреймворку Flask. Дослідження фокусується на розробці функціональностей для відстеження доходів та витрат, планування бюджету та аналізу фінансових транзакцій.

Завдання кваліфікаційної роботи:

1. Аналіз існуючих рішень у сфері веб-додатків для управління фінансами, їх переваг та недоліків.

2. Проектування архітектури та функціональних можливостей веб-додатку.

3. Розробка веб-додатку з використанням мови програмування Python та фреймворку Flask.

4. Тестування та налагодження веб-додатку.

5. Оцінка ефективності та зручності розробленого веб-додатку.

Практичне значення кваліфікаційної роботи полягає у створенні веб-додатку для управління фінансами, який дозволяє користувачам ефективно контролювати свої доходи та витрати, планувати бюджет, категоризувати фінансові транзакції та візуалізувати фінансові дані за допомогою інтерактивних графіків. Розроблений додаток сприяє автоматизації процесів управління фінансами, що підвищує ефективність та зручність для користувачів. Застосування сучасних технологій, таких як Python, Flask, SQLite та Chart.js, забезпечує високу продуктивність, безпеку та можливість масштабування системи.

Метою дослідження є розробка веб-додатку для управління фінансами з такими основними функціями, як відстеження доходів та витрат, планування бюджету та аналіз фінансових транзакцій.

В результаті виконання завдання можна підтвердити наступні програмні результати: ПР01 – аналіз, цілеспрямований пошук і вибір необхідних для вирішення професійних завдань інформаційно-довідникових ресурсів і знань з урахуванням сучасних досягнень науки і техніки; ПР03 – знання основних процесів, фаз та ітерацій життєвого циклу програмного забезпечення; ПР06 – уміння вибирати та використовувати відповідну задачі методологію створення програмного забезпечення; ПР09 – знання та вміння використовувати методи та засоби збору, формулювання та аналізу вимог до програмного забезпечення; ПР12 – застосування на практиці ефективних підходів щодо проектування програмного забезпечення; ПР14 – застосування на практиці інструментальних програмних засобів доменного аналізу, проектування, тестування, візуалізації, вимірювань та документування програмного забезпечення.

Досягнення поставлених завдань дозволить створити зручний, функціональний та безпечний веб-додаток, який допоможе користувачам ефективно управляти своїми фінансами, контролювати доходи та витрати, планувати бюджет та досягати фінансових цілей. Застосування сучасних технологій та передових методик розробки забезпечить високу якість кінцевого продукту, простоту у використанні та інтуїтивний інтерфейс. Ретельне тестування та впровадження заходів безпеки гарантуватимуть захист конфіденційних даних та безперебійну роботу системи. Загалом, цей веб-додаток стане надійним помічником у фінансовому плануванні та допоможе користувачам досягти їхніх фінансових цілей.

Структура кваліфікаційної роботи: вступ, 3 розділи, висновки, перелік використаних джерел, 2 додатки. Обсяг роботи – 62 сторінки, робота містить 21 рисунок, перелік використаних джерел складається з 36 посилань, додатки розміщено на 15 сторінках.

РОЗДІЛ 1.
АНАЛІЗ МЕТОДІВ ТА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ ДЛЯ ФІНАНСОВОГО УПРАВЛІННЯ

1.1 Огляд існуючих методів та підходів до автоматизації фінансового адміністрування

На даний момент існує велика кількість програмних рішень для автоматизації фінансового адміністрування, кожне з яких пропонує свої переваги та особливості. Популярними є як локальні десктопні програми, так і хмарні веб-сервіси та мобільні додатки, якими дуже зручно користуватися.

Програми, що встановлюються на локальному пристрої, такі як Quicken, Microsoft Money або GnuCash, зазвичай мають більший набір можливостей для налаштування та інтеграції з іншими системами. Основною їх перевагою є можливість працювати в автономному режимі без підключення до Інтернету та зберігання даних локально на комп'ютері користувача [1]. Однак, у сучасному світі, вони є менш зручними для використання на різних пристроях та для синхронізації даних між ними.

Хмарні веб-сервіси, такі як Mint, Personal Capital або You Need a Budget (YNAB), працюють у веб-браузері та забезпечують доступ до даних з будь-якого пристрою, підключеного до Інтернету [2]. Вони часто пропонують автоматичну синхронізацію з банківськими рахунками, що спрощує введення даних про транзакції. Крім того, веб-додатки зазвичай легше оновлюються та отримують нові функції.

Для більш детального розуміння популярності цих веб-сервісів, було проведено аналіз різних додатків, що пропонують подібні функції [3]. На діаграмі нижче (див. рис. 1.1) наведено порівняння популярності таких веб-сервісів.

Рисунок 1.1 – Порівняльна популярність веб-сервісів для управління фінансами

Мобільні додатки для управління фінансами також набирають популярність, оскільки дозволяють користувачам легко відстежувати свої витрати та доходи в будь-якому місці. Програми, такі як Monefy, Wallet або Doli, пропонують зручний та інтуїтивний інтерфейс для введення транзакцій безпосередньо зі смартфону [4]. Ці додатки часто синхронізуються з хмарними сервісами, забезпечуючи резервне копіювання даних та можливість переглядати фінансову інформацію на різних пристроях.

Порівнюючи локальні програми, хмарні сервіси та мобільні додатки, можна виділити декілька ключових відмінностей. Локальні програми зазвичай пропонують більше можливостей для налаштування та інтеграції, але вимагають встановлення на конкретний пристрій і можуть бути менш зручними для синхронізації даних. Хмарні сервіси забезпечують доступ до даних з будь-якого пристрою, підключеного до Інтернету, але можуть бути менш гнучкими в налаштуваннях. Мобільні додатки ідеально підходять для

швидкого відстеження витрат у дорозі, але часто мають обмежені функції порівняно з повноцінними десктопними або веб-додатками.

Зазвичай, вибір оптимального рішення залежить від індивідуальних потреб користувача, його технічних навичок та готовності платити за додаткові функції. Деякі користувачі можуть віддавати перевагу комбінованому використанню кількох типів програм для максимальної гнучкості та зручності.

Існують різні підходи до фінансового адміністрування, які базуються на відомих принципах управління особистими фінансами. Ці підходи допомагають структурувати та візуалізувати процес розподілу доходів, витрат і заощаджень, що сприяє кращому контролю над фінансами [5].

Один із таких методів - метод "конвертиків" (envelope method). Він полягає у розподілі бюджету на різні категорії витрат, такі як оренда, їжа, транспорт тощо [6]. Для кожної категорії виділяється окремий фізичний конверт або віртуальний "гаманець" у фінансовому додатку, куди заноситься певна сума грошей. Коли кошти вичерпуються, витрати в цій категорії припиняються до наступного періоду. Цей підхід допомагає краще розуміти, як розподіляється бюджет, та запобігає перевитратам у певних категоріях [7].

Ще одним поширеним методом є "плати собі першим" (pay yourself first). Це означає, що відразу після отримання доходу певна сума відкладається на ощадний рахунок або інші заощадження, перш ніж витрачати решту грошей [8]. Таким чином формується звичка заощаджувати, і користувач має жити на залишок після відкладень. Цей метод спонукає ставити фінансові цілі на перше місце та контролювати витрати [9].

Обидва ці підходи є досить прості та інтуїтивно зрозумілими для більшості людей. Вони полегшують управління фінансами та допомагають у прийнятті розумних рішень щодо витрат та заощаджень. Метод "конвертиків" добре підходить для контролю різних категорій витрат, тоді як "плати собі першим" стимулює до систематичного заощадження перед витратами.

Загалом, існує широкий спектр методів та інструментів для автоматизації фінансового адміністрування, які можуть задовольнити різні потреби та

вподобання користувачів. Вони можуть доповнюватися іншими методиками чи цифровими інструментами відстеження фінансів для досягнення максимальних результатів.

1.2 Аналіз функціональності та переваг існуючих веб-додатків

Сучасні веб-додатки для керування фінансами пропонують широкий вибір корисних функцій та інструментів, які полегшують контроль над особистими фінансами користувачів. Однією з головних переваг таких додатків є можливість відстежувати всі фінансові операції в одному централізованому місці. Більшість веб-додатків надають можливість інтегрувати банківські рахунки, кредитні картки та інвестиційні портфелі для автоматичного імпорту транзакцій [10]. Це значно спрощує процес моніторингу витрат і доходів, оскільки користувачам не потрібно вводити дані вручну, що економить їх час і зусилля.

Крім того, веб-додатки пропонують потужні інструменти для категоризації та аналізу витрат. Витрати автоматично розподіляються за заздалегідь визначеними категоріями, такими як їжа, транспорт, розваги тощо. Це дозволяє користувачам легко відстежувати, на що вони витрачають гроші, та виявляти сфери для потенційної економії [11].

Візуалізація даних є ще одним ключовим аспектом, який робить веб-додатки зручними для користувачів. Багато додатків пропонують зрозумілі діаграми та графіки, які дозволяють користувачам легко зрозуміти свої фінансові звички та тенденції витрат. Така візуалізація допомагає приймати більш обґрунтовані фінансові рішення.

Слід зазначити, що деякі веб-додатки пропонують функції спільного управління фінансами, що особливо корисно для пар або сімей [12]. Ці додатки дозволяють кільком користувачам мати доступ до спільних рахунків та витрат, полегшуючи процес узгодження бюджету та фінансового планування.

Усі ці можливості допомагають користувачам ретельно контролювати свої кошти, виявляти можливості для економії та досягати фінансових цілей. Під час розробки нового веб-додатку для керування фінансами слід ретельно проаналізувати ці переваги та інтегрувати найбільш затребувані та корисні функції, щоб створити зручне та ефективне програмне рішення, яке відповідатиме потребам користувачів.

Веб-додатки для управління фінансами завоювали популярність серед багатьох користувачів, оскільки надають зручний та доступний інструмент для керування особистими фінансами. Це дозволяє стежити за доходами, витратами, створювати бюджети та ставити фінансові цілі, не покладаючись на складні таблиці чи паперові записи. Проте, як і у будь-якого програмного забезпечення, вони мають свої переваги та недоліки, які варто враховувати.

Порівняння різних веб-додатків дає змогу користувачам зробити поінформований вибір відповідно до їхніх потреб та уподобань. Ключовими факторами при виборі є зручність використання, функціональність, безпека даних та вартість. Деякі додатки пропонують базові функції безкоштовно, тоді як інші вимагають передплати для розширених можливостей [13].

Як зазначалося раніше, управління фінансами – це важлива навичка, яка допомагає уникнути боргів, накопичувати заощадження та досягати фінансових цілей [14]. Веб-додатки спрощують цей процес, автоматизуючи відстеження витрат, категоризацію та візуалізацію даних. Вони також можуть надавати корисні поради та рекомендації щодо інвестицій, планування пенсії та оптимізації податків.

Для порівняння переваг та недоліків різних веб-додатків для управління фінансами було вибрано чотири популярні додатки: Mint, You Need a Budget (YNAB), Personal Capital та Quicken. Їх порівняльний аналіз наведено в таблиці 1.1, де детально зіставлено позитивні та негативні аспекти функціонування кожного додатка. Це дозволяє виявити сильні та слабкі сторони кожного програмного рішення.

Таблиця 1.1

Порівняння популярних веб-додатків для фінансового управління

Додаток	Переваги	Недоліки
Mint	✓ Безкоштовний додаток з великим функціоналом	✗ Занадто багато реклами в безкоштовній версії
	✓ Автоматична категоризація витрат та синхронізація з банківськими рахунками	✗ Обмежена підтримка інвестицій та кредитів
YNAB	✓ Унікальна методика «давати кожній доларовій купюрі завдання»	✗ Платна підписка зі щорічною оплатою
	✓ Потужні інструменти для планування витрат та перерозподілу коштів	✗ Відсутність функції автоматичної синхронізації з банками
Personal Capital	✓ Безкоштовне відстеження витрат, доходів та інвестицій	✗ Платна консультація з фінансовими радниками
	✓ Потужні інструменти для аналізу та оптимізації інвестиційного портфеля	✗ Відсутність деяких функцій управління бюджетом
	✓ Зручні інструменти для планування пенсії	✗ Обмежена підтримка спільного доступу для пар/сімей
Quicken	✓ Багаторічний досвід та репутація на ринку	✗ Висока вартість програмного забезпечення
	✓ Широкий спектр функцій для управління фінансами	✗ Обмежена підтримка веб-версії
	✓ Підтримка синхронізації між пристроями	✗ Складний інтерфейс для початківців
	✓ Можливість створювати звіти та планувати податки	✗ Платна річна підписка для оновлень

Виходячи з наведеної таблиці, ми можемо зробити наступні висновки:

- кожен фінансовий додаток має свої унікальні переваги та недоліки, які користувачам необхідно враховувати відповідно до своїх потреб та пріоритетів;
- веб-додаток Mint є безкоштовним додатком з широким функціоналом, автоматичною категоризацією витрат та можливістю створювати бюджети, але має багато реклами та обмежену підтримку інвестицій.

- веб-додаток YNAB пропонує унікальну методику управління фінансами, потужні інструменти планування витрат та спільного доступу, але є платним додатком без автоматичної синхронізації з банками.
- веб-додаток Personal Capital є безкоштовним для відстеження витрат та інвестицій, має потужні інструменти для аналізу портфеля та планування пенсії, але обмежений в управлінні бюджетом та спільному доступі.
- веб-додаток Quicken є платним, але потужним програмним забезпеченням з багаторічною репутацією, широким спектром функцій та можливістю створювати звіти та планувати податки, проте має високу вартість та обмежену підтримку веб-версії.

1.3 Мета та об'єкти дослідження

Дане дослідження має на меті вивчити наявні методики, підходи та програмне забезпечення для ефективного менеджменту персональних фінансів, виявляючи їхні сильні та слабкі сторони. На основі отриманих результатів планується розробити рекомендації, технічне завдання та створити новий веб-застосунок для управління фінансами, який включатиме позитивні аспекти існуючих рішень та усуватиме виявлені недоліки.

Основними методами управління особистими фінансами є:

- метод "конвертиків" (envelope method): розподіл бюджету на різні категорії витрат з метою кращого контролю над фінансами.
- метод "плати собі першим" (pay yourself first): відкладання частини доходу перед витратами для стимулювання заощаджень.

Популярними фінансовими веб-додатками є:

- Mint: аналіз функціональних можливостей, переваг і недоліків, включаючи автоматичну категоризацію витрат та інтеграцію з банківськими рахунками.

- You Need a Budget (YNAB): розгляд унікальної методики управління фінансами та інструментів планування витрат.
- Personal Capital: оцінка потужних інструментів для аналізу та оптимізації інвестиційного портфеля.
- Quicken: аналіз широкого спектру функцій для управління фінансами та створення звітів.

Технології та архітектурні рішення:

- використані мови програмування: Python, фреймворки: Flask, бази даних: SQLite та інші технічні аспекти.
- підходи до розробки та забезпечення безпеки даних користувачів.
- багаторівнева авторизація з використанням шифрування для захисту конфіденційної інформації користувача

Ключові етапи дослідження:

- проаналізувати існуючі методи управління особистими фінансами та їх ефективність.
- провести детальний аналіз функціональності, переваг і недоліків популярних фінансових веб-додатків.
- визначити технології та архітектурні рішення, що використовуються в існуючих додатках.
- розробити рекомендації щодо функціональних вимог та архітектурних рішень для нового веб-додатку.
- створити прототип веб-додатку на базі Python і Flask та провести його тестування.

За підсумками проведеного дослідження планується сформулювати рекомендації щодо оптимальних підходів, необхідного функціоналу та найбільш придатних технологічних рішень для створення інноваційного веб-додатку для керування фінансовими ресурсами. Очікується, що розроблений застосунок матиме зручний та інтуїтивно зрозумілий інтерфейс, забезпечуватиме високу продуктивність і результативність у виконанні покладених на нього завдань, а також якнайкраще задовольнятиме

різноманітні потреби цільової аудиторії користувачів. Таким чином, кінцевим результатом дослідницької роботи стане створення ефективного програмного рішення для управління фінансами, яке відповідатиме сучасним вимогам і тенденціям у цій галузі.

Слід зазначити, що дослідження охоплюватиме як традиційні методики керування фінансами, такі як метод "конвертиків" та "плати собі першим", так і сучасні програмні рішення у вигляді веб-додатків та мобільних застосунків. Особлива увага буде приділена аналізу функціональних можливостей, переваг і недоліків популярних фінансових веб-додатків, таких як Mint, You Need a Budget (YNAB), Personal Capital та Quicken. Результати такого комплексного аналізу дозволять виявити найбільш ефективні підходи та практики, які варто інтегрувати у новий веб-застосунок.

1.4 Висновки до першого розділу

У першому розділі було детально розглянуто різні методи автоматизації фінансового адміністрування, включаючи локальні десктопні програми, хмарні веб-сервіси та мобільні додатки. Виявлено, що кожен з цих методів має свої унікальні переваги та недоліки, що залежать від потреб користувачів та їхніх технічних можливостей. Аналіз існуючих веб-додатків для управління фінансами, таких як Mint, YNAB, Personal Capital та Quicken, показав, що сучасні додатки надають широкий вибір функцій для автоматизації фінансових процесів. Вони забезпечують інтеграцію з банківськими рахунками, автоматичну категоризацію витрат та надання фінансових звітів.

Одним з ключових аспектів аналізу була оцінка функціональності та зручності використання різних веб-додатків. Виявлено, що найпопулярніші додатки забезпечують високу зручність користування завдяки інтерактивним інтерфейсам, які дозволяють користувачам швидко і легко вводити дані про фінансові операції та отримувати зворотний зв’язок у вигляді графіків і діаграм. Порівняння методів "конвертиків" та "плати собі першим" дозволило

зробити висновок, що обидва підходи є ефективними для управління фінансами. Метод "конвертиків" допомагає краще контролювати витрати, тоді як метод "плати собі першим" сприяє регулярному заощадженню.

Розгляд різних програмних засобів для розробки веб-додатків, таких як Flask для серверної частини та Chart.js для візуалізації даних, показав, що вибір технологій має вирішальне значення для успішної реалізації проекту. Flask забезпечує гнучкість та легкість розробки, а Chart.js дозволяє створювати інтерактивні графіки для аналізу фінансових даних. Оцінка різних систем управління базами даних, таких як SQLite, PostgreSQL та MongoDB, показала, що кожна з них має свої переваги в залежності від специфіки проекту. SQLite була обрана для цього проекту завдяки своїй легкості у використанні та високій продуктивності для невеликих обсягів даних.

РОЗДІЛ 2.
АНАЛІЗ ДЖЕРЕЛ ТА ВИРІШЕННЯ ПРОБЛЕМ

2.1 Огляд існуючих методів фінансового адміністрування

Фінансове адміністрування є важливим аспектом управління як особистими, так і корпоративними фінансами. Воно включає в себе процеси планування, організації, контролю та моніторингу фінансових ресурсів для досягнення фінансових цілей [15]. Існує кілька основних методів фінансового адміністрування, які використовуються в сучасному світі, а саме: традиційні, автоматизовані та інноваційні методи.

Традиційні методи фінансового адміністрування включають ведення фінансових записів вручну та використання електронних таблиць [16]. Ведення фінансових записів вручну є одним з найстаріших методів, що полягає у ручному веденні обліку витрат та доходів. Цей метод є досить трудомістким та схильним до помилок, однак він досі використовується у малих підприємствах та для особистого фінансового управління [17]. Використання електронних таблиць є популярним способом ведення фінансових записів завдяки своїй доступності та простоті використання. Програмне забезпечення, таке як Microsoft Excel або Google Sheets, дозволяє користувачам створювати таблиці для відстеження доходів, витрат, створення бюджетів та фінансового планування.

Автоматизовані системи фінансового управління включають спеціалізоване програмне забезпечення, онлайн-сервіси та додатки. Існує багато програмних продуктів, призначених для фінансового управління, таких як QuickBooks, Xero та FreshBooks [18]. Ці інструменти пропонують широкий спектр функцій, включаючи управління рахунками, відстеження витрат, підготовку фінансових звітів та інше. З розвитком інтернет-технологій популярними стали онлайн-сервіси для фінансового управління, такі як Mint, Personal Capital та YNAB (You Need A Budget). Ці платформи надають

можливість користувачам інтегрувати свої банківські рахунки, відстежувати фінансові операції в режимі реального часу та планувати бюджет.

Інноваційні методи фінансового адміністрування включають машинне навчання та штучний інтелект, а також блокчейн та криптовалюти [19]. Сучасні фінансові системи все частіше використовують алгоритми машинного навчання та штучного інтелекту для аналізу фінансових даних, прогнозування доходів та витрат, а також виявлення шахрайства. Це дозволяє значно підвищити точність та ефективність фінансового адміністрування. Технологія блокчейн та криптовалюти також знаходять своє застосування у фінансовому управлінні. Вони забезпечують високий рівень безпеки та прозорості фінансових операцій, що особливо важливо для корпоративного фінансового управління [20].

Незважаючи на відмінності між цими методами, важливо розуміти їхню популярність та поширеність використання серед користувачів. Згідно з останніми дослідженнями ринку, попит на різні методи фінансового адміністрування розподіляється наступним чином (див. рис. 2.1).

Рисунок 2.1 – Популярність методів фінансового адміністрування [20]

Усі ці методи мають свої переваги та недоліки, і вибір конкретного методу залежить від потреб користувача та специфіки фінансового управління. У сучасному світі часто використовується комбінація різних методів для досягнення максимальної ефективності та надійності фінансового адміністрування.

2.2 Вибір технологій та засобів розробки для веб-додатку

Розробка будь-якого веб-додатку, безпосередньо для управління фінансами, вимагає вибору відповідних технологій та засобів, що забезпечать ефективну, надійну та масштабовану реалізацію проекту. Цей процес включає в себе аналіз наявних рішень, визначення відповідних середовищ розробки, мов розмітки та стилів, програмування, а також систем управління базами даних. Вибір цих компонентів залежить від багатьох факторів, таких як вимоги проекту, компетенції команди розробників, наявність підтримки спільноти та документованість технологій.

Важливим етапом є вибір архітектури додатку [21]. Це може бути монолітна архітектура, мікросервісна або ж використання серверлесс-технологій. Кожна з них має свої переваги та недоліки, і вибір залежить від специфіки проекту. Наприклад, мікросервісна архітектура дозволяє легше масштабувати окремі частини додатку та забезпечує більшу гнучкість у розподілі завдань між розробниками [22].

Одним з ключових аспектів розробки будь якого веб-додатку є UI/UX дизайн. Він спрямований на забезпечення зручності та інтуїтивної зрозумілості інтерфейсу користувача. Якісний UI/UX дизайн не тільки покращує взаємодію користувача з додатком, але й сприяє підвищенню продуктивності та задоволення від використання продукту [23].

Процес створення UI/UX дизайну включає декілька етапів. Перш за все, це дослідження користувачів та визначення їх потреб [24]. Це може бути реалізовано шляхом проведення опитувань, інтерв'ю та аналізу поведінки

користувачів. На основі отриманих даних створюються персони користувачів та сценарії їх взаємодії з додатком.

Наступним кроком є створення вайрфреймів та прототипів, які допомагають візуалізувати структуру та функціонал інтерфейсу [24]. На цьому етапі визначаються основні елементи інтерфейсу, їх розташування та взаємодія між ними. Прототипи дозволяють швидко тестувати та вносити зміни до дизайну, що забезпечує високу гнучкість у процесі розробки.

Остаточний дизайн включає розробку візуальних елементів, таких як кольорові палітри, типографіка, іконки та інші графічні компоненти [24]. Важливо, щоб візуальний стиль був узгоджений та відповідав бренду компанії. Після цього дизайн впроваджується у вигляді інтерактивного прототипу, який можна тестувати на реальних користувачах для виявлення можливих проблем та отримання зворотного зв'язку.

Після цього вже можна переходити до вибору стеку технологій. Для фронтенду, тобто клієнтської частини веб-додатку, зазвичай використовуються HTML, CSS та JavaScript-фреймворки, такі як React, Angular або Vue.js. Для бекенду, тобто серверної частини веб-додатку, популярні мови програмування включають JavaScript (Node.js), Python (Django, Flask), Ruby (Ruby on Rails) , PHP (Laravel) та інші (див. рис. 2.2). Вибір бази даних залежить від типу даних та обсягів, які планується зберігати. Реляційні бази даних, такі як MySQL або PostgreSQL, підходять для структурованих даних, тоді як NoSQL бази даних, такі як MongoDB, підходять для неструктурованих або напівструктурованих даних.

Тому, можна сказати, що вибір стеку технологій є важливим рішенням, яке впливає на ефективність розробки, продуктивність та масштабованість вашого веб-додатку. Ретельно оцініть вимоги проекту, наявні ресурси та навички команди, а також переваги та недоліки різних технологій перед тим, як зробити остаточний вибір.

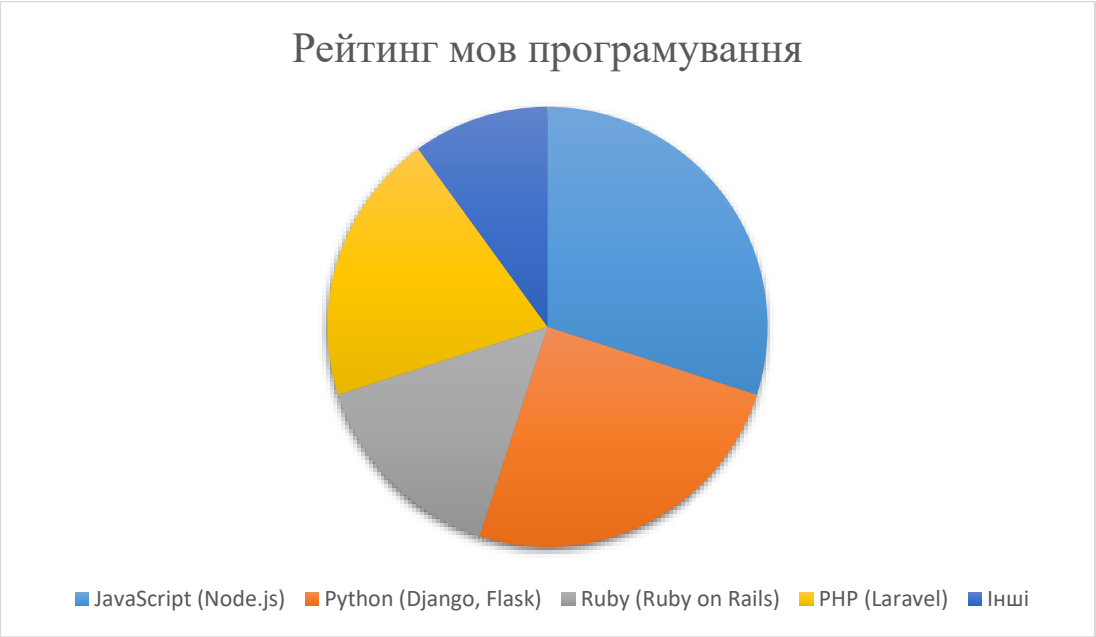


Рисунок 2.2 – Рейтинг мов програмування для серверної частини [25]

Загалом, розробка будь-якого веб-додатку - це складний і багатоступеневий процес, який потребує уважного планування та вибору технологій, щоб забезпечити успіх проекту [25].

Що до аналізу програмних засобів для розробки веб-додатків, то він включає вивчення та оцінку різних фреймворків, бібліотек та інструментів, які можуть бути використані для створення проекту [25]. Основною метою цього аналізу є вибір найбільш ефективних та зручних інструментів, що відповідатимуть вимогам проекту. Для розробки веб-додатку було обрано Flask, легкий та мінімалістичний фреймворк для Python [26], який забезпечує гнучкість та розширюваність. Flask дозволяє розробникам швидко створювати прототипи та легко інтегрувати додаткові модулі та розширення. Для візуалізації даних обрано JavaScript-бібліотеку для створення стовпчастих діаграм, яка дозволяє ефективно візуалізувати фінансову інформацію у вигляді інтерактивних графіків. Аналіз програмних засобів також включає оцінку інструментів для тестування та налагодження коду, таких як pytest для автоматичного тестування та інструменти для відлагодження у Visual Studio Code.

Середовище розробки, також, відіграє ключову роль у процесі створення веб-додатку, забезпечуючи зручність та ефективність написання, тестування та налагодження коду. Існує багато середовищ розробки, кожне з яких має свої особливості та переваги.

Visual Studio Code було обрано як основне середовище розробки для цього проекту завдяки своїм численним перевагам. Visual Studio Code є легким та швидким редактором коду, який підтримує великий набір розширень, що значно розширює його функціональність [27]. Зокрема, розширення для Python та JavaScript забезпечують автодоповнення, налагодження, рефакторинг коду та інтеграцію з системами контролю версій, такими як Git. Крім того, вбудований термінал у Visual Studio Code дозволяє виконувати командні інтерфейси безпосередньо у середовищі розробки, що спрощує роботу з інструментами для розробки.

Серед інших популярних середовищ розробки варто виділити:

- PyCharm: Професійне середовище розробки для Python від JetBrains. PyCharm надає розширені можливості для написання коду, включаючи потужний налагоджувач, інструменти для тестування, підтримку фреймворків та інтеграцію з системами контролю версій. PyCharm відомий своєю зручністю для великих проектів та корпоративного використання.
- Sublime Text: Легкий редактор коду, який підтримує безліч мов програмування. Sublime Text відзначається швидкістю роботи та можливістю налаштування через плагіни. Однак, у порівнянні з Visual Studio Code, Sublime Text має менше вбудованих функцій для інтеграції з системами контролю версій та налагодження.
- Atom: Відкритий редактор коду, розроблений GitHub. Atom підтримує багато мов програмування та пропонує гнучкі можливості налаштування через пакети. Він також має вбудовану інтеграцію з GitHub, що спрощує спільну роботу над проектами. Проте, Atom може бути менш продуктивним при роботі з великими проектами.

- Eclipse: Потужне середовище розробки, яке підтримує різні мови програмування через плагіни. Eclipse зазвичай використовується для розробки на Java, але також підтримує Python через розширення. Це середовище відзначається своєю гнучкістю, але може бути складнішим у налаштуванні та менш інтуїтивним у використанні.

Підсумовуючи, Visual Studio Code було обрано завдяки своїм численним перевагам, таким як швидкість роботи, легкість у використанні, широкий набір розширень та підтримка багатьох мов програмування. Це середовище забезпечує зручність у роботі з кодом, дозволяючи ефективно управляти проектом та співпрацювати з іншими розробниками. Інтеграція з Git, підтримка автодоповнення та налагодження роблять Visual Studio Code ідеальним вибором для розробки веб-додатку на базі Python та JavaScript.

Для створення структури та зовнішнього вигляду веб-сторінок використовуються мови розмітки та стилів. HTML (HyperText Markup Language) використовується для структурування контенту веб-сторінок [28]. Він забезпечує основний каркас веб-сторінок, визначаючи заголовки, абзаци, посилання, зображення та інші елементи. CSS (Cascading Style Sheets) використовується для стилізації веб-сторінок, задаючи кольори, шрифти, макети та інші візуальні аспекти [29]. У цьому проекті для створення адаптивного та кросбраузерного інтерфейсу користувача застосовується звичайний flexbox. Використання flexbox дозволяє легко створювати гнучкі та адаптивні макети, що відповідають сучасним вимогам до веб-дизайну.

Для забезпечення логіки та функціональності веб-додатків використовуються спеціалізовані мови програмування. Від правильного вибору такої мови залежить швидкість розробки, оптимальність роботи створеного програмного продукту та зручність його супроводу у майбутньому.

Для цього проекту основними мовами програмування обрано Python та JavaScript. Python забезпечує легкість написання коду, має велику кількість бібліотек для вирішення різноманітних задач та відомий своєю читабельністю та простотою синтаксису. Flask, фреймворк на базі Python, використовується

для створення бекенду, забезпечуючи простий та ефективний спосіб керування запитами та відповідями. JavaScript використовується для створення інтерактивних елементів та візуалізації даних, зокрема з використанням бібліотеки для стовпчастих діаграм. Це дозволяє створювати динамічні та візуалізовані графіки, що значно покращуть користувацький досвід.

Крім Python та JavaScript, існує багато інших мов програмування, які можна використовувати для розробки веб-додатків:

- Java: Одна з найбільш популярних мов програмування, відома своєю продуктивністю та безпекою. Java широко використовується для розробки великих корпоративних додатків та забезпечує високий рівень масштабованості. Spring Framework є одним з найпопулярніших фреймворків для створення веб-додатків на Java.
- Ruby: Динамічна мова програмування, відома своєю простотою та елегантністю. Ruby on Rails, фреймворк на базі Ruby, забезпечує швидку розробку веб-додатків завдяки своїй гнучкості та великій кількості готових рішень.
- PHP: Серверна мова програмування, яка спеціально розроблена для створення веб-додатків. PHP широко використовується завдяки своїй простоті та великій кількості фреймворків, таких як Laravel та Symfony, які полегшують розробку.
- C#: Мова програмування від Microsoft, яка часто використовується для розробки веб-додатків на платформі .NET. ASP.NET є потужним фреймворком, що забезпечує високу продуктивність та гнучкість для розробки веб-додатків.

Python та JavaScript було обрано як основну мову програмування для бекенду цього проекту з кількох причин. По-перше, Python відомий своєю простотою та читабельністю, що робить його ідеальним для швидкої розробки та прототипування. Його синтаксис є інтуїтивно зрозумілим, що знижує ймовірність помилок та спрощує подальше обслуговування коду. По-друге, Python має велику кількість бібліотек та фреймворків, таких як Flask, які

спрощують розробку веб-додатків та забезпечують необхідну функціональність. Flask є легким та мінімалістичним фреймворком, що дозволяє швидко створювати прототипи та легко інтегрувати додаткові модулі та розширення. Крім того, Python має активну спільноту розробників та велику кількість ресурсів для навчання, що робить його доступним для новачків та професіоналів.

Також, слід зазначити, що система управління базами даних (СУБД) є критичним компонентом для зберігання та управління даними у веб-додатку. Вибір СУБД значно впливає на продуктивність, надійність та масштабованість додатку [30].

Для цього проекту було обрано SQLite тому, що воно є легковажною реляційною СУБД, яка не потребує окремого серверу для своєї роботи. Вона забезпечує простоту використання, легкість налаштування та високу продуктивність для невеликих та середніх проектів. SQLite вбудовується безпосередньо у додаток, що дозволяє уникнути додаткових витрат на адміністрування та обслуговування серверу бази даних. Ці особливості роблять SQLite ідеальним вибором для дипломного проекту, де важливо швидко налаштувати робочу систему без складних конфігурацій.

Існує багато інших СУБД, кожна з яких має свої переваги та недоліки:

- MySQL: Одна з найпопулярніших реляційних СУБД з відкритим кодом. Вона відома своєю високою продуктивністю, масштабованістю та надійністю. Вона широко використовується у веб-додатках різного масштабу, від малих до великих корпоративних систем.
- PostgreSQL: Потужна реляційна СУБД з відкритим кодом, яка відома своєю відповідністю стандартам SQL, розширюваністю та підтримкою складних запитів. PostgreSQL часто використовується для розробки додатків, де важлива надійність і висока продуктивність.
- MongoDB: Документо-орієнтована СУБД, яка зберігає дані у форматі BSON (бінарний JSON). MongoDB забезпечує високу продуктивність

та масштабованість, що робить її популярним вибором для додатків, які працюють з великими обсягами даних та мають складні структури.

Хоча SQLite було обрано для поточного проекту, важливо розглянути відносну популярність різних СУБД серед розробників та проектів. Дана кругова діаграма ілюструє розподіл використання основних СУБД на ринку (див. рис 2.3).

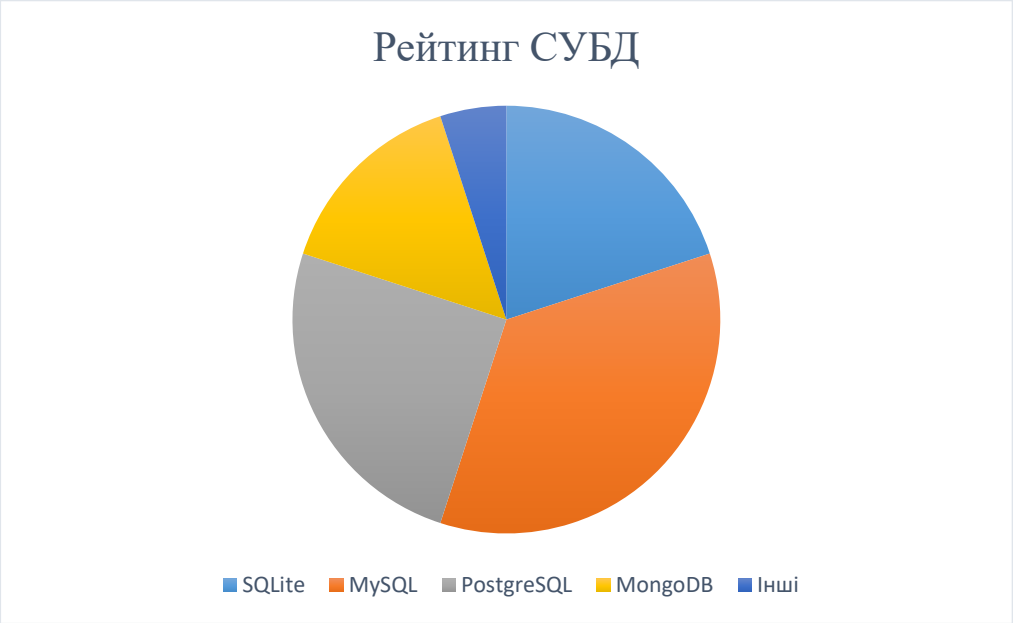


Рисунок 2.3 – Рейтинг СУБД [30]

Отже, SQLite було обрано для цього проекту завдяки своїм численним перевагам, зокрема простоті використання та легкості налаштування. Оскільки цей проект є дослідницькою роботою, важливо було обрати СУБД, яка не вимагає складної конфігурації та адміністрування. Тому, SQLite дозволяє швидко розпочати роботу, оскільки вона вбудовується безпосередньо у додаток і не потребує окремого серверу. Це значно зменшує час на налаштування та обслуговування бази даних. Крім того, SQLite забезпечує достатню продуктивність для невеликих та середніх проектів, що робить її ідеальним вибором для прототипування та розробки таких проектів.

Водночас, слід зазначити, що SQLite має обмеження щодо конкурентного доступу та масштабованості порівняно з повноцінними

серверними СУБД, такими як PostgreSQL або MySQL. Проте, для цілей зазначеного проекту ці обмеження не були критичними. SQLite надає достатньо функціональних можливостей для зберігання та обробки необхідних даних у рамках дослідження. Її відносна простота та низька вимогливість до ресурсів дозволили зосередитися на основній логіці програми, а не на адмініструванні складної бази даних. Таким чином, SQLite виявилася оптимальним рішенням, що повністю відповідало поставленим вимогам і забезпечила необхідну гнучкість та продуктивність для успішної реалізації проекту.

2.3 Висновки до другого розділу

Другий розділ роботи був присвячений аналізу існуючих методів фінансового адміністрування, включаючи традиційні ручні методи, електронні таблиці та сучасні автоматизовані системи. Було визначено, що сучасні автоматизовані системи значно підвищують ефективність управління фінансами. Огляд технологій, що використовуються для розробки веб-додатків, показав, що вибір відповідних технологічних засобів є критичним для успіху проекту. Було розглянуто різні середовища розробки, мови програмування, системи управління базами даних та інструменти для тестування і налагодження коду.

Аналіз популярних середовищ розробки, таких як Visual Studio Code, PyCharm, Sublime Text та Atom, показав, що Visual Studio Code є найбільш підходящим середовищем для розробки цього проекту завдяки своїй легкості, швидкості та широкому набору розширень. Оцінка різних мов розмітки та стилів, таких як HTML та CSS, показала, що використання сучасних інструментів для створення адаптивних та кросбраузерних інтерфейсів є важливим аспектом розробки веб-додатків. Використання flexbox дозволяє створювати гнучкі та адаптивні макети.

Вибір мови програмування для цього проекту зупинився на Python та JavaScript. Python забезпечує легкість написання коду та має велику кількість бібліотек для різних задач, тоді як JavaScript додає інтерактивність та динамічність веб-сторінкам. Було детально розглянуто різні системи управління базами даних, такі як MySQL, PostgreSQL, MongoDB та SQLite. SQLite була обрана для цього проекту завдяки своїй простоті, легкості налаштування та високій продуктивності для невеликих обсягів даних.

Розділ також містив аналіз різних архітектурних підходів для розробки веб-додатків, таких як монолітна архітектура, мікросервіси та MVC патерн. Було обрано архітектуру MVC завдяки її гнучкості, модульності та можливості легко додавати нові функції. Загалом, проведений аналіз технологій та архітектурних рішень дозволив визначити найкращі практики для розробки веб-додатку для управління фінансами, що стало основою для подальшої розробки і тестування системи.

РОЗДІЛ 3.
РОЗРОБКА ТА ТЕСТУВАННЯ ВЕБ-ДОДАТКУ НА БАЗІ PYTHON І FLASK

3.1 Опис програмної архітектури та функціональності додатку

У сучасному світі, де фінансові операції стають дедалі складнішими, а витрати та доходи можуть бути розгалуженими між різноманітними джерелами, надзвичайно важливо мати зручні та ефективні інструменти для управління особистими фінансами. Веб-додатки, розроблені спеціально для цієї мети, допомагають користувачам чітко відстежувати свої витрати, контролювати бюджет та аналізувати фінансові показники. Однак створення таких додатків вимагає ретельно продуманого підходу до вибору архітектури та технологічного стеку.

Правильний вибір компонентів, бібліотек та фреймворків відіграє критичну роль у забезпеченні зручності використання, продуктивності, безпеки та масштабованості веб-додатку для управління фінансами. Застосування сучасних архітектурних рішень, таких як мікросервісна архітектура або хмарні рішення, дозволяє створити гнучку та стійку систему, здатну витримувати високе навантаження та легко масштабуватись при збільшенні кількості користувачів.

Крім того, використання перевірених часом технологій та інструментів забезпечує високий рівень безпеки та конфіденційності фінансових даних користувачів, що є надзвичайно важливим в епоху зростаючих кіберзагроз. Ретельно продумана архітектура додатку з використанням надійних компонентів та строгих протоколів безпеки гарантує, що конфіденційна інформація користувачів буде належним чином захищена від несанкціонованого доступу чи витоку.

Для забезпечення належного рівня безпеки, розробники можуть використовувати такі підходи, як шифрування даних, багаторівнева автентифікація, регулярне оновлення системи та аудит безпеки. Крім того,

важливо інтегрувати передові практики програмування та принципи безпечного кодування, такі як валідація вхідних даних, належне управління сесіями та токенами доступу, а також обробка винятків та логування. Комплексний підхід до безпеки на всіх рівнях – від архітектури до коду – є критично важливим для захисту конфіденційних фінансових даних користувачів.

Основні компоненти, які були обрані для створення веб-додатку, включають серверну частину, базу даних та фронтенд. Серверна частина на Flask є основою додатку. Flask — це мікрофреймворк на Python, який забезпечує простоту та гнучкість для створення веб-додатків. Він дозволяє швидко розробляти додатки, забезпечуючи необхідний функціонал для обробки HTTP-запитів, маршрутизації та шаблонізації [31]. Також, слід зазначити, що він підтримує розширення, які додають функціонал для роботи з базами даних, аутентифікацією, управління сесіями та іншими важливими аспектами веб-розробки.

Якщо брати основні переваги Flask, то вони включають легкість і простоту у вивченні та використанні, що дозволяє швидко розпочати розробку додатків. Гнучкість Flask не нав'язує конкретної структури проекту, що дозволяє розробникам налаштовувати додаток відповідно до своїх потреб. Flask також розширюваний, підтримуючи безліч розширень, які додають додатковий функціонал, такий як аутентифікація, робота з базами даних тощо [31]. Активна спільнота Flask забезпечує підтримку і доступ до великої кількості навчальних матеріалів та документації.

Що стосується бази даних для цього проекту, то було обрано SQLite через низку переваг, які роблять її ідеальним рішенням для невеликих додатків. SQLite - це легка вбудована СУБД, яка не потребує налаштування сервера. Вона зберігає дані у файлах, що робить її ідеальною для невеликих додатків, де простота використання важливіша за масштабованість. SQLite забезпечує високу продуктивність для невеликих обсягів даних і підтримує всі основні функції, які можна очікувати від СУБД.

Основні переваги SQLite включають простоту використання, оскільки вона не потребує налаштування сервера, що робить її ідеальною для невеликих проектів. Швидкість та ефективність SQLite забезпечує високу швидкість обробки запитів для невеликих обсягів даних. Крім того, SQLite легко інтегрується з Python за допомогою стандартної бібліотеки sqlite3. Збереження даних у файлах робить процес резервного копіювання і перенесення бази даних простим і зручним.

Незважаючи на свої переваги, SQLite також має деякі обмеження, які слід враховувати:

- масштабованість SQLite добре працює для невеликих обсягів даних, але може не справлятися з великими і активно оновлюваними базами даних. При збільшенні навантаження продуктивність може значно знизитися.
- обмежена підтримка конкурентного доступу SQLite забезпечує лише часткову підтримку паралельного доступу до бази даних. Це може стати проблемою для додатків з високою конкурентністю.
- відсутність підтримки реплікації та кластеризації SQLite не підтримує реплікацію даних та кластеризацію, що може бути вимогою для високонавантажених корпоративних рішень.
- обмежені можливості адміністрування Відсутність окремого сервера бази даних означає відсутність інструментів для централізованого адміністрування та моніторингу.
- відсутність підтримки збережених процедур та тригерів SQLite має обмежену функціональність порівняно з потужнішими СУБД, не підтримуючи такі функції, як збережені процедури та тригери.

Фронтенд складається з HTML-сторінок, CSS для стилізації та JavaScript для динамічних елементів. Використовуються шаблони Flask для рендерингу HTML-сторінок. HTML визначає структуру веб-сторінок, CSS забезпечує їх стильове оформлення, а JavaScript додає динамічність і інтерактивність.

Використання цих технологій дозволяє створювати сучасні веб-інтерфейси, які забезпечують зручність і привабливість для користувачів.

HTML (Hypertext Markup Language) є стандартною мовою розмітки для створення веб-сторінок. Він забезпечує структуру документа, визначаючи розділи, заголовки, параграфи, списки та інші елементи. HTML також дозволяє вбудовувати мультимедійний контент, такий як зображення, відео та аудіо.

CSS (Cascading Style Sheets) використовується для стилізації HTML-елементів, визначаючи їх зовнішній вигляд і макет. За допомогою CSS можна керувати шрифтами, кольорами, розмірами, відступами, фоновими зображеннями та багатьма іншими аспектами візуального відображення.

JavaScript є потужною мовою програмування, яка додає інтерактивність і динамічність веб-сторінкам. За допомогою JavaScript можна створювати анімації, обробляти події користувача (наприклад, натискання кнопок або введення даних), виконувати обчислення та взаємодіяти з веб-API для розширення функціональності.

Поєднання HTML, CSS та JavaScript створює багатий і зручний користувацький інтерфейс, забезпечуючи візуально привабливе відображення, адаптивний дизайн та інтерактивність. Використання шаблонів Flask також спрощує процес розробки та інтеграцію фронтенду з бекендом додатку.

Якщо говорити про вибір архітектурного підходу, то існує декілька підходів, серед яких монолітна архітектура, архітектура мікросервісів та MVC (Model-View-Controller).

Монолітна архітектура передбачає, що всі компоненти додатку розміщуються в одному проекті та працюють разом [32]. У цьому підході весь додаток розгортається як єдине ціле, об'єднуючи інтерфейс користувача, бізнес-логіку та доступ до даних. Перевагами є простота розробки та розгортання, однак такий підхід може бути менш гнучким та масштабованим, особливо зі зростанням розміру та складності додатку (див. рис. 3.1).

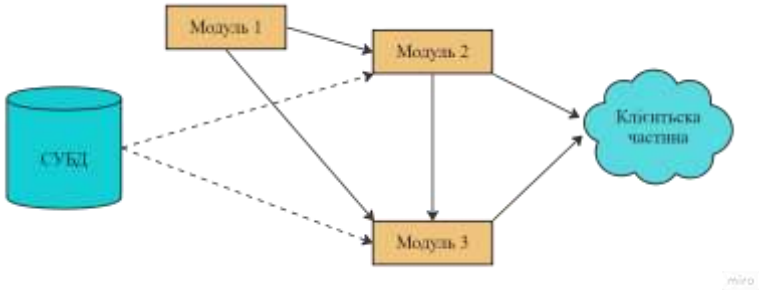


Рисунок 3.1 – Схема монолітної архітектури

Архітектура мікросервісів, навпаки, розділяє додаток на окремі невеликі сервіси, кожен з яких відповідає за певну бізнес-функціональність [32]. Ці сервіси можуть бути розроблені та розгорнуті незалежно один від одного, використовуючи різні технології, і взаємодіють між собою через API. Цей підхід забезпечує гнучкість, масштабованість, легку підтримку та ізоляцію помилок, але вимагає додаткових зусиль для управління розподіленими сервісами та забезпечення безпеки зв'язку між ними (див. рис. 3.2).

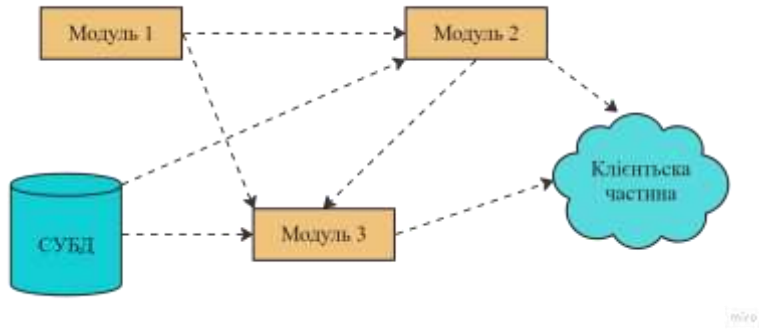


Рисунок 3.2 – Схема мікросервісної архітектури

MVC (Model-View-Controller) є архітектурним патерном, який розділяє додаток на три основні компоненти: Модель (Model), Представлення (View) та Контролер (Controller). Модель відповідає за логіку роботи з даними та бізнес-логіку, Представлення відображає дані та взаємодіє з користувачем, а Контролер обробляє запити користувачів, взаємодіючи з Моделлю та Представленням. Такий підхід забезпечує чітке розділення обов'язків,

модульність, підтримку та можливість повторного використання коду, що робить його популярним вибором для веб-додатків [33] (див. рис. 3.3).

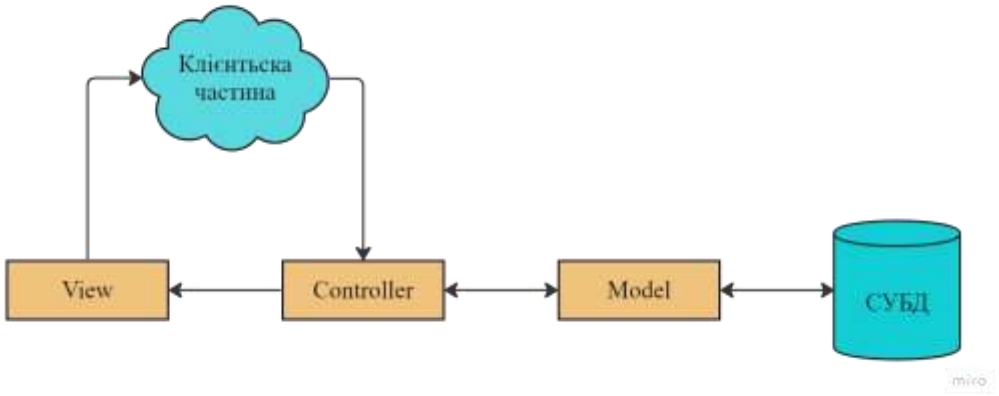


Рисунок 3.3 – Схема MVC архітектурного патерна

Для даного проекту було обрано архітектурний підхід MVC з кількох причин. По-перше, MVC забезпечує чітке розділення обов'язків, що робить код більш організованим та зрозумілим. Модель відповідає за взаємодію з базою даних та бізнес-логіку, Представлення відображає дані та взаємодіє з користувачем, а Контролер обробляє запити користувачів та координує роботу Моделі та Представлення.

По-друге, завдяки такому розділенню на модулі, розробка стає більш модульною, що спрощує процес тестування та подальшої підтримки коду. Кожен компонент можна тестувати окремо, а зміни в одному модулі мають мінімальний вплив на інші частини додатку.

По-третє, MVC дозволяє легко додавати нові функціональні можливості без значних змін у існуючому коді, забезпечуючи гнучкість та масштабованість додатку.

Також, слід зазначити, що основні функціональні можливості додатку повинні включати систему CRUD (Create, Read, Update, Delete). Ця система і є основними операціями, що виконуються з даними в додатку, такими як створення, читання та видалення транзакцій. Користувачі можуть додавати

нові транзакції, переглядати список своїх транзакцій, фільтрувати та сортувати їх, а також видаляти непотрібні записи.

Для більш зручного управління фінансами додаток повинен підтримувати категорії витрат та доходів. Це дозволяє користувачам організувати свої транзакції за категоріями, що спрощує аналіз витрат і доходів. Додаток також повинен забезпечувати можливості для аналізу транзакцій за допомогою графіків. Графіки дозволяють візуалізувати витрати та доходи за певний період, цим самим це допомагає користувачам краще розуміти свої фінансові звички та планувати бюджет заздалегідь.

Для візуалізації даних у додатку використовується бібліотека Chart.js. Вона є популярною бібліотекою для створення інтерактивних графіків на веб-сторінках [34]. Основні причини вибору цієї бібліотеки включають простоту використання, різноманітність графіків та інтерактивність. Chart.js легко інтегрується з HTML та JavaScript, що дозволяє швидко додавати графіки до веб-сторінок. Бібліотека підтримує широкий набір типів графіків, включаючи лінійні, стовпчасті, кругові та інші. Графіки, створені за допомогою Chart.js, інтерактивні та підтримують анімацію, що покращує користувацький досвід [34].

Існують також інші бібліотеки для створення графіків, такі як D3, Highcharts, Plotly та Google Charts [35]. Кожна з цих бібліотек має свої переваги та недоліки. D3, наприклад, надає високий рівень гнучкості та контролю над візуалізацією, але може вимагати більше часу для освоєння та використання. Highcharts відома своєю простотою використання та великим набором функцій, проте може бути платною для комерційного використання. Plotly підтримує широкий набір типів графіків та інтерактивність, але може бути менш гнучкою в порівнянні з D3. Google Charts є безкоштовною та легко інтегрується з іншими сервісами Google, але може мати обмеження щодо налаштувань та стилю графіків.

Детально розглянемо їх переваги та недоліки, у таблиці 3.1, у порівнянні з обраною бібліотекою Chart.js.

Таблиця 3.1

Порівняння JS-бібліотек для візуалізації даних

Бібліотека	Переваги	Недоліки
Chart.js	✓ Простота використання, різноманітність графіків, інтерактивність, анімація, відкритий код	✗ Обмежена гнучкість у налаштуванні, не підходить для дуже складних візуалізацій
D3.js	✓ Потужна, висока гнучкість, можливість створення складних та інтерактивних візуалізацій	✗ Вимагає більше коду, складніша у використанні, крута крива навчання
Highcharts	✓ Широкий набір функцій, багато типів графіків, комерційна підтримка	✗ Платна, ліцензія може бути дорогою для великих проектів
Plotly	✓ Інтерактивність, широкий набір типів графіків, висока якість візуалізацій	✗ Деякі функції платні, більший розмір бібліотеки порівняно з іншими
Google Charts	✓ Безкоштовна, інтеграція з іншими сервісами Google, широкий набір інструментів	✗ Залежність від сервісів Google, обмежена гнучкість порівняно з D3.js

Отже, можна сказати, що Chart.js була обрана для цього проекту вдало, через її простоту використання, різноманітність графіків та інтерактивність.

Таким чином, вибір технологій та архітектурного підходу був зроблений з урахуванням простоти використання, гнучкості та можливостей для масштабування. Використання Flask як фреймворку для серверної частини забезпечує легку розробку та розгортання веб-додатків на Python, а також має активне співтовариство та розвинену екосистему для додавання необхідної функціональності. SQLite, обрана як легка вбудована база даних, ідеально підходить для невеликих проектів, забезпечуючи високу швидкодію та простоту у використанні.

Такий набір технологій та архітектурний підхід дозволяє забезпечити простоту використання, гнучкість та можливості для масштабування додатку в майбутньому. Вони забезпечують зручну та ефективну роботу для користувачів, а також створюють надійну основу для подальшого розвитку проекту.

3.2 Реалізація клієнтської частини веб-додатку для зручного фінансового адміністрування

Процес розробки веб-додатку для зручного фінансового адміністрування розпочинається з UI/UX частини, а саме - визначення користувацьких потоків (User flow), що описують послідовність дій користувача в додатку [23]. Це допомагає зрозуміти, як користувачі взаємодіятимуть із системою та які кроки вони виконуватимуть для досягнення своїх цілей. Також, він включає основні сценарії використання та шляхи, які користувач проходить для виконання певних завдань.

Одним із ключових сценаріїв є реєстрація та авторизація нового користувача. User flow для цього процесу виглядає наступним чином: користувач відкриває веб-додаток, якщо він не був зареєстрований у базі даних, то натискає кнопку "Реєстрація", заповнює форму з особистими даними (ім'я, електронна пошта, пароль), натискає кнопку "Зареєструватися", вводить свої дані для входу та переходить до свого облікового запису (див. рис. 3.4).

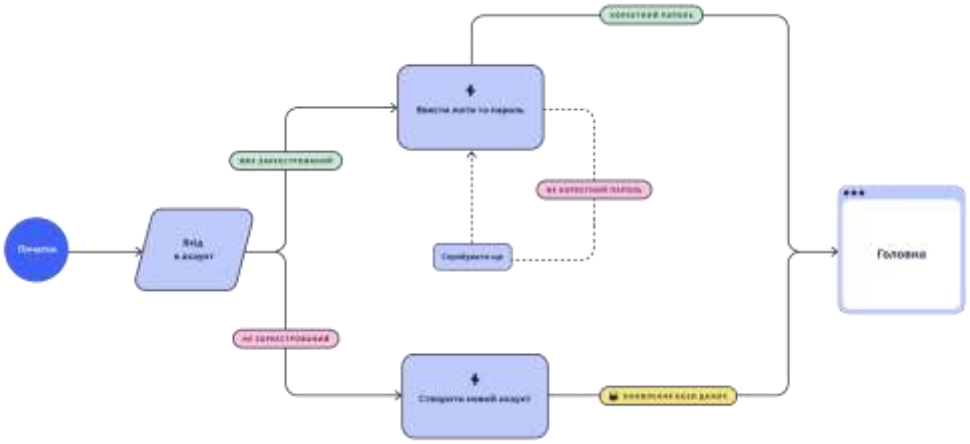


Рисунок 3.4 – Шлях користувача по реєстраційній сторінці

Потрібно, також зазначити, що в процесі реєстрації, валідація є ключовим аспектом, який забезпечує коректність введених даних і безпеку

користувачів. Валідація може виконуватись на стороні клієнта (front-end) і на стороні сервера (back-end).

На стороні клієнта валідація здійснюється за допомогою JavaScript, HTML та CSS (див. рис. 3.5). Це дозволяє виявити помилки ще до відправки даних на сервер, що покращує зручність користувача.

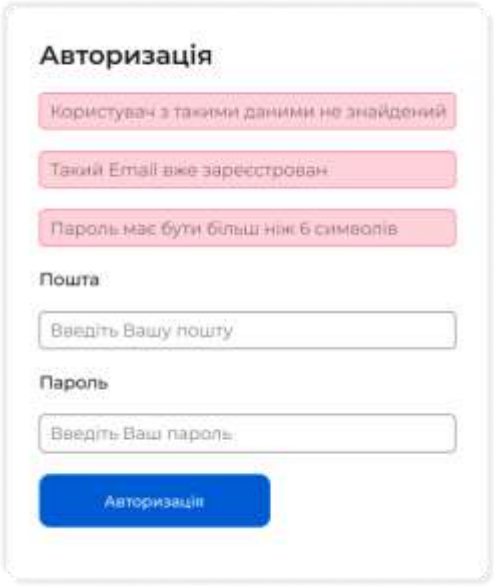


Рисунок 3.5 – Приклад форми авторизації з валідацією на стороні клієнта

Основні елементи валідації включають:

- перевірка обов'язкових полів: Поля, як-от ім'я, електронна пошта та пароль, повинні бути заповнені.
- перевірка формату електронної пошти: Введена електронна пошта повинна відповідати стандартному формату.
- перевірка пароля: Пароль має відповідати вимогам щодо мінімальної довжини та складності (наприклад, містити великі та малі літери, цифри та спеціальні символи).

HTML надає вбудовані атрибути, такі як required, type="email", та інші, які полегшують валідацію без використання додаткових функцій, написаних на, JavaScript.

На стороні сервера валідація є критично важливою для забезпечення безпеки додатка, оскільки користувач може обійти клієнтську валідацію. Основні елементи включають:

- перевірка унікальності електронної пошти: Електронна пошта користувача повинна бути унікальною, тому перед реєстрацією необхідно перевірити, чи існує вже такий користувач у базі даних.
- хешування пароля: Паролі повинні зберігатися у хешованому вигляді, використовуючи вбудовані функції.
- перевірка вхідних даних: Необхідно перевірити, що всі поля містять допустимі значення та формат.

Інший важливий сценарій – додавання нової фінансової транзакції. User flow для цього процесу включає наступні кроки (див. рис. 3.5): після входу в обліковий запис користувач вибирає розділ "Доходи", заповнює форму з деталями транзакції (назва доходу, сума транзакції, дата транзакції, вибір категорії та опис транзакції), натискає кнопку "Додати транзакцію" та підтверджує додавання транзакції.

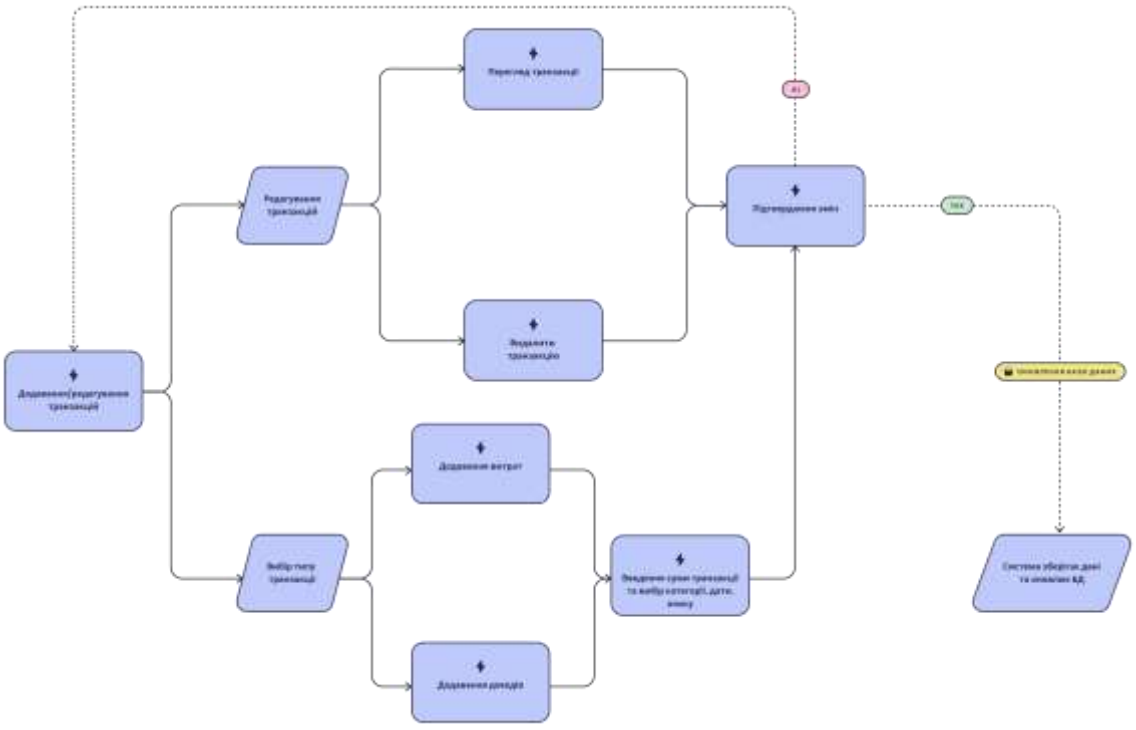


Рисунок 3.5 – Шлях користувача по сторінці з додаванням транзакцій

Ще один важливий сценарій – перегляд фінансових звітів (стовпчастих діаграм). User flow для цього процесу включає такі кроки (див. рис. 3.6): користувач входить до додатку, вибирає розділ "Статистика", налаштовує фільтри для звітів (категорії витрат), переглядає графіки з результатами.

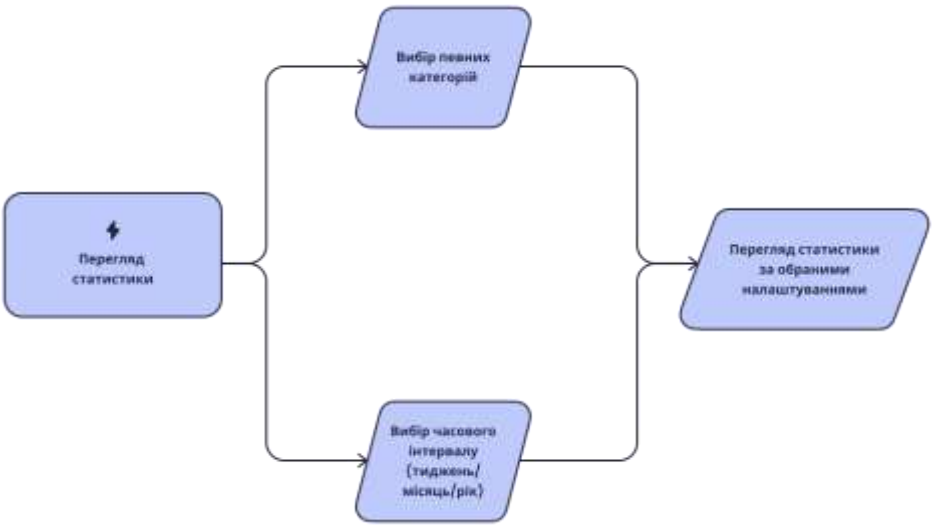


Рисунок 3.6 - Шлях користувача по сторінці з перегляду статистики

На основі визначених user flow створюються wireframes (каркасні схеми), які є основою для подальшого дизайну та розробки. Wireframes дозволяють візуалізувати структуру сторінок, розташування елементів та їх функціональність. Це забезпечує чітке розуміння того, як користувачі взаємодіятимуть з додатком, і дозволяє виявити можливі проблеми на ранніх стадіях розробки.

Створення wireframes включає кілька етапів. Спочатку визначаються основні сторінки додатку, такі як головна сторінка, сторінка реєстрації та авторизації, сторінка огляду фінансових транзакцій, сторінка додавання нових транзакцій, сторінка аналізу витрат та доходів тощо. Кожна сторінка має свою специфіку і потребує окремого підходу до розробки каркасних схем [23].

Наприклад, сторінка реєстрації повинна бути простою і зрозумілою (див. рис. 3.7), щоб користувачі могли швидко зареєструватися свій обліковий запис.

Тобто, на цій сторінці повинні бути розташовані поля вводу імені, електронної пошти, пароля та його підтвердження, загрузка аватару, та кнопка "Зареєструватися".

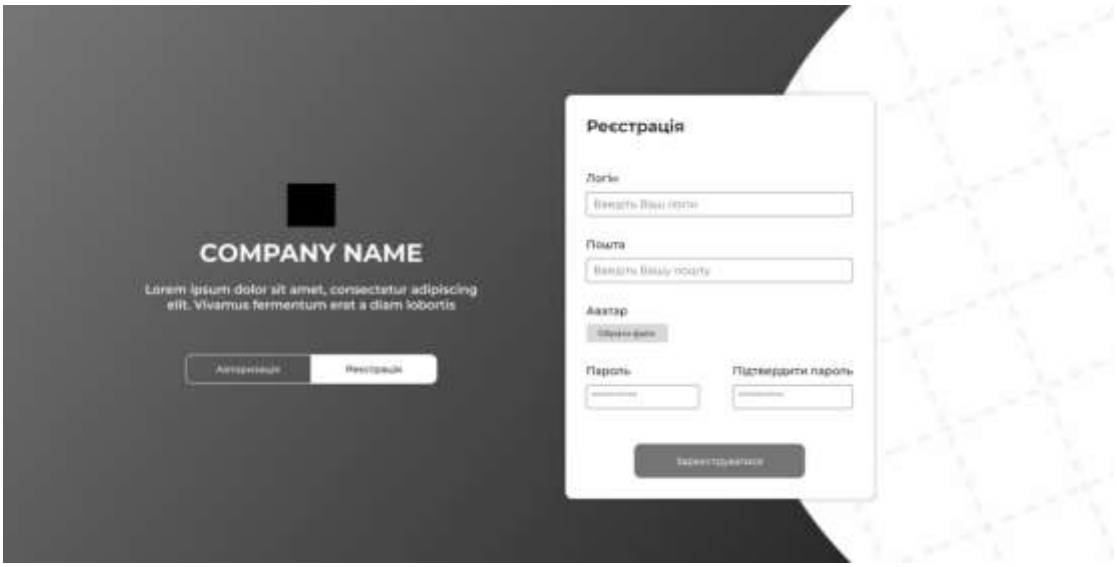


Рисунок 3.7 – Каркас сторінки реєстрації

Якщо розглядати сторінку авторизації (див. рис. 3.8), то поля вводу даних можна зменшити до введення електронної пошти, пароля та однієї кнопки "Авторизація".

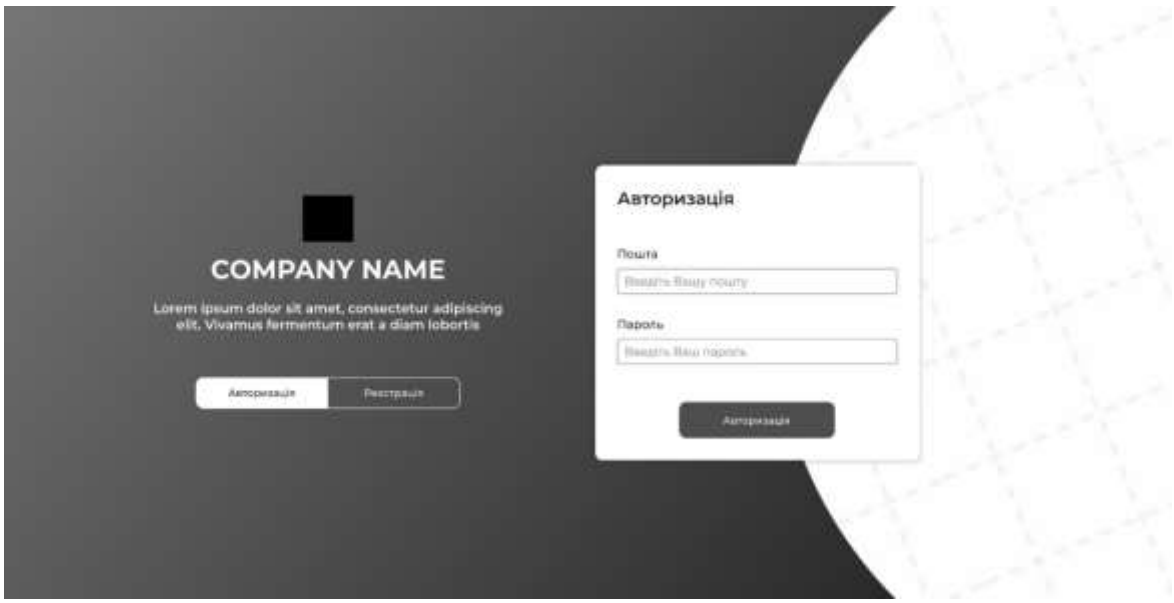


Рисунок 3.8 – Каркас сторінки авторизації

Що до головної сторінки (див. рис 3.9), то вона повинна забезпечувати користувачам швидкий доступ до основної фінансової інформації та функцій. На цій сторінці повинні бути розташовані кілька ключових елементів: огляд всіх транзакцій, блок з нещодавніми транзакціями, а також секції з загальним доходом, загальними витратами та загальним балансом. Це дозволяє користувачам отримувати повну картину своїх фінансів одразу після входу в додаток.

Основна частина сторінки містить блок "Усі транзакції", який дозволяє користувачам переглядати всі свої фінансові операції. Це було реалізовано у вигляді двох лінійних графіків, де будуть показані транзакції, а саме доходи та витрати, користувача за останні 7 днів. Такий підхід забезпечує зручність перегляду та аналізу фінансової діяльності користувача.

З правого боку, на головній сторінці, розташований блок "Нещодавні транзакції", який містить останні проведені операції. Цей блок дозволяє користувачам швидко переглядати останні зміни у своїх фінансах без необхідності шукати цю інформацію серед усіх транзакцій. Кожна транзакція у цьому блоці також відображається з категорією та сумою, що допомагає користувачам швидко орієнтуватися у своїх витратах і доходах.

Під блоком з усіма транзакціями розташовані секції з загальним доходом, загальними витратами та загальним балансом. Секція "Загальний дохід" показує суму всіх доходів користувача за певний період, "Загальні витрати" - суму витрат, а "Загальний баланс" - різницю між доходами та витратами, яка відображає фінансовий стан користувача на даний момент. Ці секції є важливими елементами, які допомагають користувачам швидко оцінити свій фінансовий стан.

Додатково, головна сторінка повинна містити інтерактивні елементи, що дозволяють користувачам швидко перемикатися між різними розділами додатку, такими як планування бюджету, аналітика витрат та управління категоріями транзакцій. Це сприяє зручному та ефективному управлінню

фінансами, забезпечуючи користувачам повний контроль над їх фінансовою ситуацією в будь-який момент.

Рисунок 3.9 – Каркас головної сторінки веб-додатку

Також варто зазначити панель навігації, яка розташована з лівого боку. Вона містить основні розділи додатку, такі як "Головна", "Статистика", "Доходи", "Витрати", а також кнопку для виходу з облікового запису. Це забезпечує швидкий доступ до всіх основних функцій додатку та допомагає користувачам легко переміщуватися між різними розділами.

Наступним кроком, варто зробити ескізи, виходячи з userflow, для сторінок доходів та витрат. Вони повинні бути аналогічними та простими у користуванні (див. рис. 3.10).

Основною частиною сторінки має містити секцію "Загальні доходи" чи "Загальні витрати", які відображатимуть суму всіх доходів та витрат користувача за певний період. Це дозволяє користувачам швидко оцінити свій фінансовий стан.

Ліворуч розташована секція для введення деталей нової транзакції. Користувачі можуть заповнити поля для назви транзакції, суми, дати, категорії

та опису транзакції. Така структура забезпечує зручність і простоту введення даних. Після заповнення всіх полів користувач натискає кнопку "Додати транзакцію" для збереження нової транзакції.

Праворуч повинен відображатися список всіх транзакцій користувача з можливістю видалення кожної транзакції. Це дозволяє користувачам легко керувати своїми фінансовими записами, переглядати деталі.

Таким чином, сторінки додавання доходів і витрат забезпечують простоту і зручність використання, дозволяючи користувачам легко вводити нові фінансові записи та керувати своїми фінансами. Ключовими елементами є поля для введення необхідних даних, кнопка для додавання транзакції та список існуючих записів з можливістю редагування та видалення.

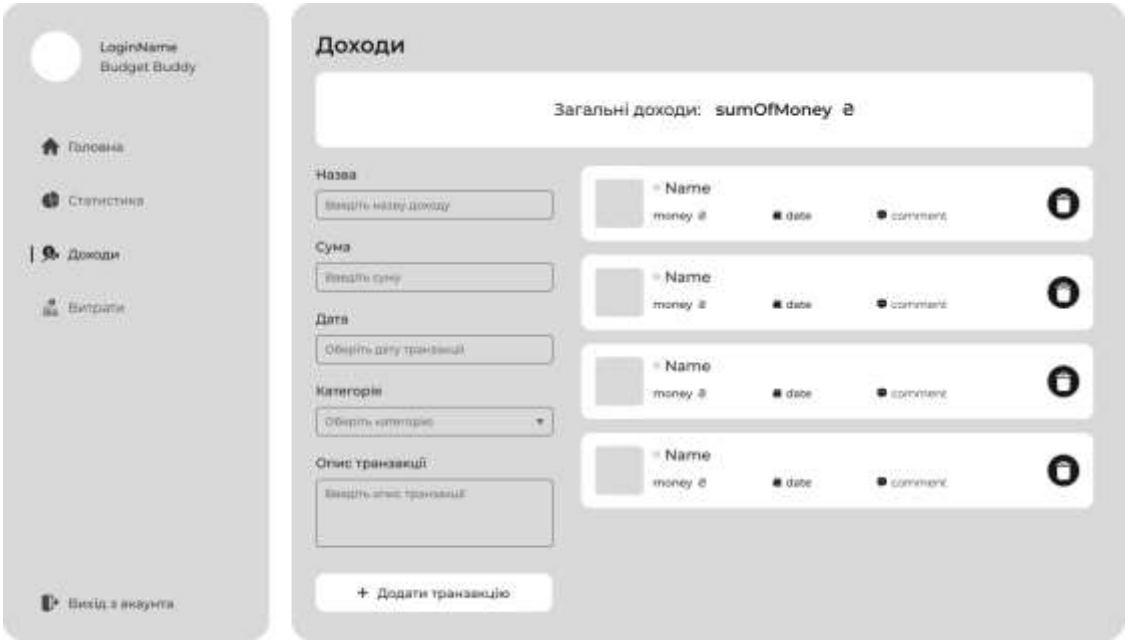


Рисунок 3.10 - Каркас сторінки з додаванням транзакцій

Після розробки та тестування User Flow та Wireframes можна починати робити кінцевий UI/UX дизайн самого додатку. Цей етап включає створення візуально привабливого та функціонального інтерфейсу, який відповідає потребам користувачів та забезпечує інтуїтивно зрозумілу навігацію по додатку. Основними принципами, які використовувалися для створення

дизайну, є простота та мінімалізм, інтуїтивність, консистентність, візуальна ієрархія та адаптивність.

Одним з основних завдань на цьому етапі є забезпечення простоти та мінімалізму в дизайні. Це означає, що кожен елемент інтерфейсу повинен мати чітке призначення і не перевантажувати користувача зайвою інформацією. Всі елементи повинні бути розташовані таким чином, щоб користувачі могли швидко знайти необхідну інформацію та виконати потрібні дії [36].

Інтуїтивність інтерфейсу також є критично важливою. Користувачі повинні мати можливість легко орієнтуватися в додатку, навіть якщо вони користуються ним вперше. Для досягнення цього використовуються добре відомі патерни взаємодії, які відповідають очікуванням користувачів. Наприклад, кнопки для виконання основних дій (як-от збереження або видалення даних) повинні бути розташовані в стандартних місцях, а їхній вигляд повинен чітко вказувати на їхнє призначення [36].

Консистентність дизайну означає, що всі сторінки та екрани додатку повинні мати однаковий вигляд і відчуття. Це включає використання єдиних кольорових схем, шрифтів та стилів для всіх елементів інтерфейсу. Консистентність допомагає користувачам швидко звикнути до інтерфейсу та зменшує ймовірність помилок, оскільки користувачі можуть легко передбачити, як поводитимуться різні елементи додатку.

Візуальна ієрархія допомагає користувачам швидко знаходити найважливішу інформацію. Важливі елементи інтерфейсу повинні бути виділені за допомогою розміру, кольору або розташування. Наприклад, основні кнопки дій можуть бути більшими і яскравішими, ніж вторинні дії, щоб привернути увагу користувачів до найважливіших елементів.

Тому, дизайн повинен бути простим і не перевантаженим зайвими елементами, що допомагає користувачам швидко орієнтуватися у додатку та знаходити потрібну інформацію без зайвих зусиль. Інтерфейс повинен бути зрозумілим навіть для нових користувачів, а всі елементи управління повинні бути розташовані логічно та відповідати очікуванням користувачів. Усі

сторінки додатку повинні мати однаковий стиль та структуру, що допомагає користувачам швидко звикнути до інтерфейсу та зменшує ймовірність помилок. Важливі елементи повинні бути виділені за допомогою розміру, кольору або розташування, щоб користувачі могли швидко знаходити найважливішу інформацію. У майбутньому додаток повинен бути оптимізований для використання на різних пристроях, включаючи комп’ютери, планшети та смартфони.

Важливим аспектом розробки кінцевого дизайну є тестування. Тестування інтерфейсу з реальними користувачами допомагає виявити проблеми з юзабіліті та переконатися, що дизайн відповідає потребам користувачів. Це може включати проведення юзабіліті-тестів, аналіз відгуків користувачів та внесення відповідних змін до дизайну на основі отриманих даних.

Для створення кінцевого дизайну та клієнтської частини використовувались такі інструменти, як Figma, HTML, CSS та JavaScript.

Figma допомогла створити інтерактивні прототипи, працювати над дизайном у команді та отримувати зворотний зв’язок від користувачів у режимі реального часу. HTML та CSS забезпечили структуру та стилізацію веб-сторінок, дозволяючи створювати адаптивний дизайн, а JavaScript використовувався для додавання інтерактивності та динамічних елементів на веб-сторінки.

Кінцева клієнтська частина веб-додатку для зручного фінансового адміністрування є результатом ретельного планування, тестування та реалізації. Використання сучасних інструментів та підходів до UI/UX дизайну дозволило створити інтерфейс, який забезпечує зручність, ефективність та естетичну привабливість. Користувачі можуть легко взаємодіяти з додатком, здійснювати фінансові операції та отримувати необхідну інформацію для управління своїми фінансами.

Наступним кроком у розробці веб-додатку є інтеграція дизайну з серверною логікою та базами даних, що забезпечить повну функціональність та безпеку системи.

3.3 Розробка програмних модулів та управління базами даних у фінансовому веб-додатку

Після успішної реалізації клієнтської частини веб-додатку, наступним кроком є розробка серверної логіки та взаємодія з базами даних. Ця частина є критично важливою, оскільки вона забезпечує функціональність додатку, обробку та зберігання даних.

У фінансовому веб-додатку серверна логіка відіграє ключову роль у забезпеченні безпечної, надійної та ефективної обробки фінансових операцій та даних. Від правильної реалізації цієї частини залежить не лише функціональність додатку, але й довіра користувачів до системи та захист їхніх конфіденційних фінансових даних.

Розробка серверної логіки вимагає ретельного планування, продуманого проектування архітектури та дотримання найкращих практик безпеки та масштабованості. Крім того, взаємодія з базами даних є невід'ємною частиною будь-якого веб-додатку, оскільки дані є серцевиною системи.

Для початку реалізації серверної частини веб-додатку було створену файлову структуру, яку схематично наведено на рисунку у додатку А.

Проект має наступну структуру: кореневий каталог містить основні файли додатку, такі як application.py, get_data.py, models.py, money.py, README.md, requirements.txt, system.py, view.py. У кореневому каталозі розміщуються файли, відповідальні за ініціалізацію додатку, обробку даних та опис залежностей.

Каталог instance містить базу даних money-db.db. Цей каталог використовується для зберігання конфіденційних даних та налаштувань, які не повинні потрапляти у версійний контроль.

Каталог static містить статичні файли, такі як CSS, JavaScript, зображення та завантажені файли. Статичні файли організовані в підкаталоги assets та upload.

В assets розміщені підкаталоги css, favicon, img та js, які містять відповідні типи файлів. Каталог templates містить шаблони HTML, які використовуються для рендерингу веб-сторінок. Шаблони організовані в підкаталоги base, що включає файли footer.html, header.html, menu.html, та основні сторінки додатку: expenses.html, incomes.html, login.html, main.html, register.html, statistic.html.

Ініціалізація додатку відбувається у файлі application.py, де ініціалізується Flask додаток, налаштовуються конфігурації та розгортаються основні маршрути. У файлі models.py описуються моделі для взаємодії з базою даних, використовуючи ORM (Object-Relational Mapping).

Бізнес-логіка зосереджена у файлі money.py, який містить обробку фінансових операцій, таких як додавання, редагування та видалення транзакцій. Файл get_data.py відповідає за отримання та обробку даних з бази даних для відображення на фронтенді.

У свою чергу, шаблони HTML, у каталозі templates, створюються за допомогою Flask та передають користувачеві зворотну інформацію, отриману з бази даних.

Ця організація файлів та каталогів дозволяє підтримувати чисту та зрозумілу структуру проекту, що полегшує його розвиток та підтримку в майбутньому. Добре структурована файлово-каталогова архітектура сприяє ефективній співпраці між розробниками, полегшує інтеграцію нових функцій та забезпечує високу продуктивність і надійність фінансового веб-додатку.

Що до архітектури розробки, як зазначалося в попередніх розділах, була обрана MVC. Такий підхід дозволяє чітко розділити обов'язки між компонентами та спрощує управління кодом. Коли користувач взаємодіє з веб-додатком через інтерфейс, наприклад, вводить дані для нової транзакції, цей запит направляється до контролера. Контролер обробляє запит та взаємодіє з

моделлю для збереження нової транзакції в базі даних. Після успішного збереження даних, контролер передає результат до представлення, яке форматує дані у відповідний формат та відображає їх для користувача.

Саме на етапі взаємодії з моделлю відбувається зв'язок із базою даних, що є критично важливим для забезпечення надійного зберігання та обробки фінансових даних. Ефективне управління базами даних є ключовим елементом успішної реалізації фінансового веб-додатку.

Як зазначалося раніше, у проекті використовується SQLite для зберігання даних, що робить його легким у налаштуванні та використанні. Файл instance/money-db.db є основним місцем зберігання всіх даних додатку. Структура бази даних включає три основні таблиці: users, transactions_list та category_list. Кожна з цих таблиць має свої специфічні поля та взаємозв'язки.

Таблиця users містить інформацію про користувачів додатку, включаючи такі поля, як id, user_email, user_pass, user_name, user_avatar, balance та api_key. Поле id є первинним ключем, що унікально ідентифікує кожного користувача. Взаємозв'язок між користувачами та їхніми транзакціями реалізовано через поле id, яке є зовнішнім ключем у таблиці transactions_list.

Таблиця transactions_list зберігає інформацію про всі фінансові операції користувачів. Вона включає такі поля, як id, user_id, created_at, pay_type, pay_name, pay_category, pay_sum, pay_comment та pay_icon. Поле user_id встановлює зв'язок між транзакцією та користувачем, до якого вона належить. Це забезпечує можливість відстеження всіх транзакцій, що належать конкретному користувачу.

Таблиця category_list містить інформацію про категорії транзакцій. Вона включає такі поля, як id, category_id, category_type, category_name та category_avatar. Поле category_id також є зовнішнім ключем, що встановлює зв'язок з таблицею users.

Взаємозв'язки між цими таблицями забезпечують цілісність даних та дозволяють легко здійснювати операції над даними. Наприклад, можна легко отримати всі транзакції конкретного користувача або всі транзакції, що

належать до певної категорії. Це робить додаток не лише функціональним, але й зручним для користувачів.

Для візуалізації взаємозв'язків між таблицями була створена UML-діаграма (див. рис. 3.11), яка допомагає краще зрозуміти структуру бази даних. UML-діаграма показує, як таблиці взаємодіють між собою, які поля є ключовими та як встановлюються зв'язки між ними.

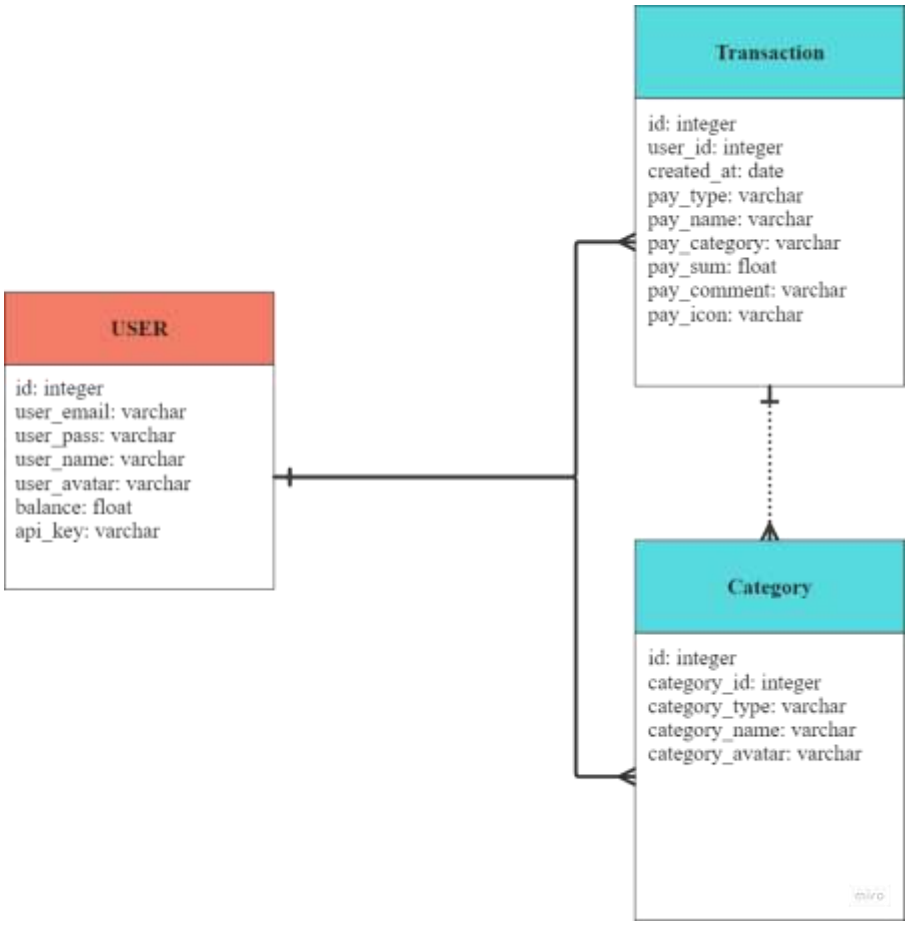


Рисунок 3.11 – UML-діаграма СУБД веб-додатку

Таким чином, правильна структура бази даних та ефективне управління нею є основою для надійного та функціонального фінансового веб-додатку. Використання ORM спрощує роботу з базою даних, дозволяючи розробникам зосередитися на логіці додатку, а не на написанні SQL-запитів вручну. Це забезпечує гнучкість та розширюваність додатку, що є критично важливим для його успіху.

Щодо конфіденційності користувачів, безпека даних має першочергове значення. Один із важливих аспектів безпеки – це захист паролів користувачів. Для цього у Flask вбудовано підтримку для шифрування паролів за допомогою алгоритму PBKDF2 (Password-Based Key Derivation Function 2) (див. рис. 3.12).

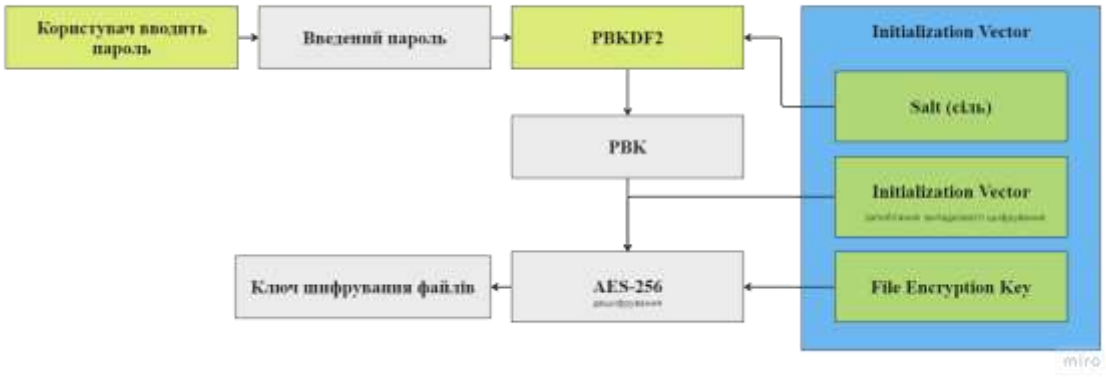


Рисунок 3.12 - Схема шифрування паролів алгоритму PBKDF2

Процес шифрування пароля за допомогою PBKDF2 у Flask розпочинається з генерування унікальної випадкової послідовності байтів, відомої як "сіль" (salt). Ця сіль поєднується з самим паролем, після чого обчислюється хеш-сума цього поєднання за допомогою функції хешування PBKDF2. Зазначена функція виконує велику кількість ітерацій (зазвичай близько 100 000) для створення надійного хешу. Отриманий хеш та відповідна сіль зберігаються в базі даних, замість самого пароля у відкритому вигляді. Таким чином, навіть якщо зловмисник отримає доступ до бази даних, він не зможе дізнатися початкові паролі користувачів.

Під час перевірки пароля, введеного користувачем, Flask отримує збережену сіль для цього користувача з бази даних. Потім введений пароль поєднується з цією сіллю, і за допомогою тієї ж функції PBKDF2 обчислюється новий хеш. Якщо новий хеш збігається зі збереженим хешем, це означає, що пароль правильний.

Цей підхід забезпечує високий рівень безпеки, оскільки навіть у випадку компрометації бази даних зловмисник не зможе легко відновити оригінальні паролі користувачів. Завдяки використанню PBKDF2 паролі захищені від атак типу rainbow tables, а багаторазове хешування значно ускладнює виконання брутфорс-атак.

Отже, забезпечення конфіденційності користувачів також є одним з важливіших аспектів розробки фінансового веб-додатку. Використання сучасних методів шифрування, таких як PBKDF2, гарантує, що дані користувачів будуть надійно захищені, що сприяє підвищенню довіри до додатку та забезпечує його успішну експлуатацію.

3.4 Оцінка ефективності та перевірка розробленого веб-додатку

Оцінка ефективності розробленого веб-додатку для управління фінансами є критично важливим етапом завершення проекту, оскільки дозволяє визначити, наскільки додаток відповідає вимогам користувачів і забезпечує очікувані функціональні можливості.

Перш за все, була проведена перевірка функціональності веб-додатку. Функціональне тестування включало перевірку всіх основних функцій додатку, таких як реєстрація та авторизація користувачів, додавання та перегляд транзакцій, створення та аналіз бюджету. Результати тестування показали, що всі ці функції працюють коректно. Користувачі могли без проблем реєструватися, додавати нові транзакції та переглядати свої фінансові звіти. Реєстрація нових користувачів та валідація введених даних пройшли успішно, вхід в систему та управління сесіями працюють стабільно, додавання, редагування та видалення фінансових транзакцій виконуються без помилок, а візуалізація даних за допомогою бібліотеки Chart.js коректно відображає фінансові звіти.

Після реєстрації або авторизації, користувач переходить до головної сторінки (див. рис. 3.13). Користувач може переглядати загальний огляд своїх фінансів, включаючи доходи, витрати та баланс за поточний місяць

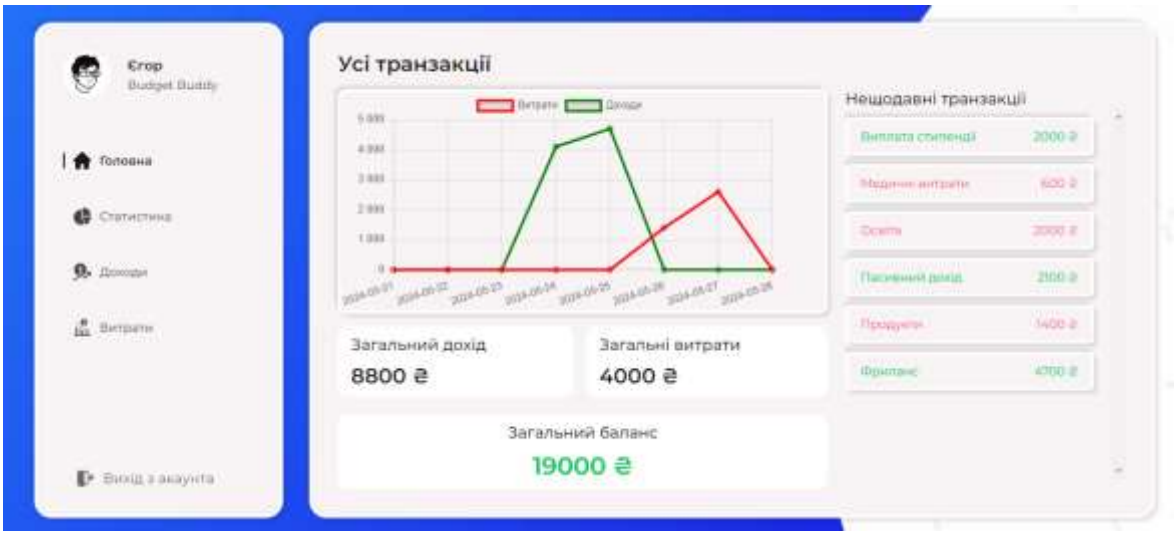


Рисунок 3.13 - Головна сторінка веб-додатку

Сторінки додавання транзакцій таких, як доходи та витрати – аналогічні між собою. На цих сторінках користувач може вводити деталі про транзакцію, вибирати категорію та дату транзакції (рис. 3.14).

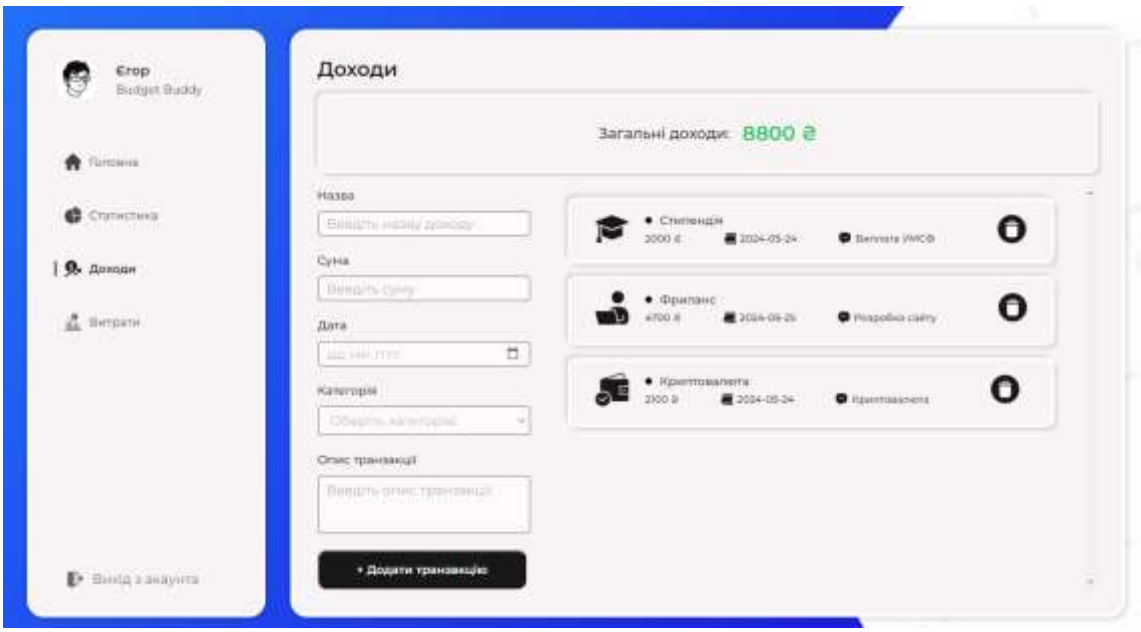


Рисунок 3.14 - Сторінка додавання нової транзакції

На сторінці з переглядом статистики (див. рис. 3.15), користувач може візуально переглядати свої доходи та витрати, розподілені за різними категоріями.

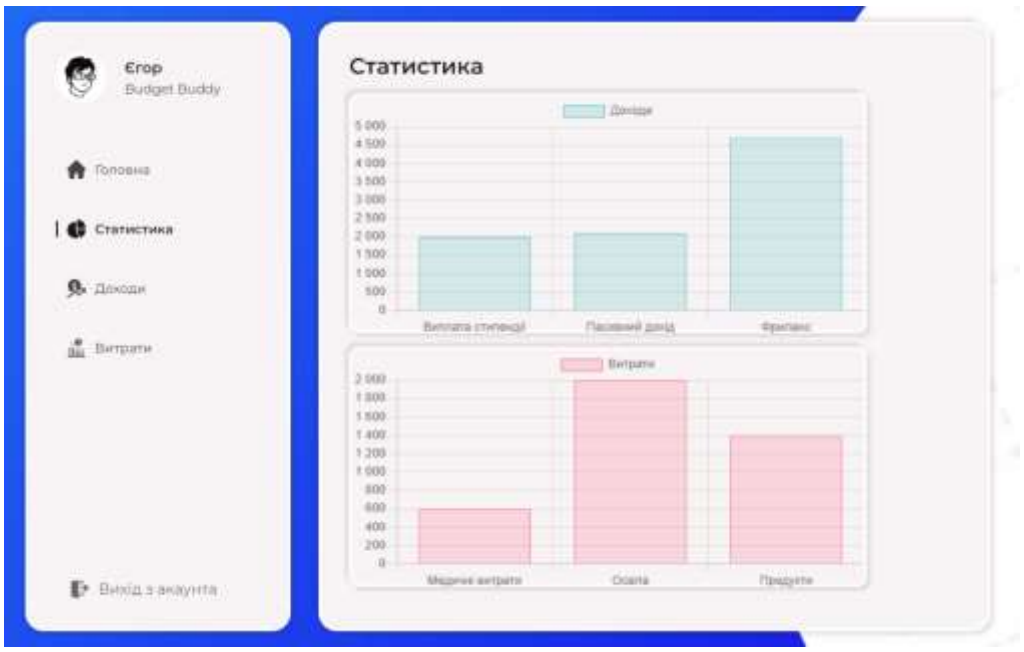


Рисунок 3.15 – Сторінка статистики користувача

Також, для оцінки зручності користування (UX) було проведено юзабіліті-тестування серед 10 людей. Їм було запропоновано оцінити загальне враження від використання додатку, а також висловити свої зауваження та пропозиції щодо покращення. Основні результати опитування показали, що 95% користувачів задоволені функціональністю додатку, 80% користувачів відзначили, що додаток допоміг їм краще контролювати свої фінанси. Основні пропозиції щодо покращення включали додавання нових функцій, таких як нагадування про платежі та більш детальний аналіз витрат.

Тестування продуктивності, також, є важливим етапом у розробці будь-якого веб-додатку, оскільки воно дозволяє оцінити ефективність його роботи під реальними умовами використання. Для тестування продуктивності нашого фінансового веб-додатку був використаний інструмент Google Lighthouse (див.

рис. 3.16). Це потужний інструмент для аналізу веб-сторінок, який надає детальну інформацію про різні аспекти продуктивності.

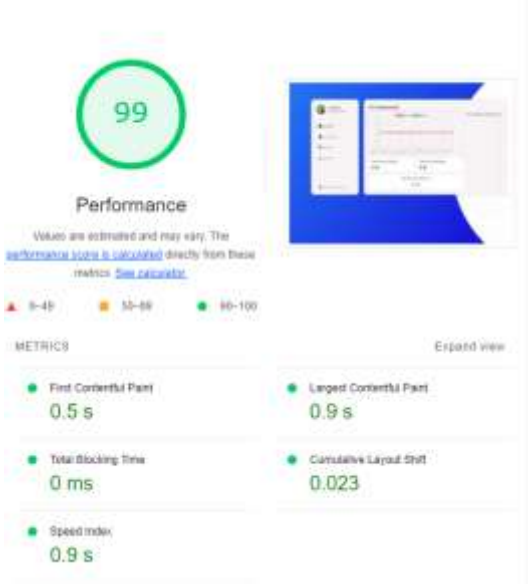


Рисунок 3.16 - Результати тестування продуктивності веб-додатку за допомогою Google Lighthouse

Перший показник, який було проаналізовано, це First Contentful Paint (FCP), який становив 0.5 секунди. Цей показник відображає час, що проходить від моменту початку завантаження сторінки до появи першого текстового або графічного елементу на екрані. Важливість FCP полягає в тому, що він показує, наскільки швидко користувач починає бачити вміст сторінки. Низький FCP означає, що користувачі швидко отримують візуальний відгук від веб-додатку, що покращує їхній досвід.

Другий важливий показник – це Largest Contentful Paint (LCP), який становив 0.9 секунди. LCP вимірює час, за який основний контент сторінки стає видимим для користувача. Це критично важливий параметр, оскільки користувачі часто оцінюють швидкість завантаження сторінки на основі того, як швидко вони бачать головний вміст. Оптимальний LCP має бути менше 2.5 секунд, тому наш результат в 0.9 секунди є відмінним показником.

Загальний час блокування (Total Blocking Time, TBT) становив 0 мілісекунд. TBT вимірює час, протягом якого основний потік браузера заблокований і не може реагувати на введення користувача. Нульовий TBT означає, що наш додаток дуже чутливий і не затримує користувача в очікуванні, що є великим плюсом для користувацького досвіду.

Speed Index, який показав значення 0.9 секунди, відображає швидкість, з якою контент сторінки стає видимим під час її завантаження. Чим менше значення Speed Index, тим швидше користувачі бачать весь вміст сторінки, що є важливим для забезпечення позитивного користувацького досвіду.

Показник зміщення макету (Cumulative Layout Shift, CLS) був на рівні 0.023. CLS вимірює візуальну стабільність сторінки і показує, наскільки непередбачувано зміщуються елементи під час завантаження. Низьке значення CLS свідчить про те, що користувачі не відчувають дратівливих зміщень контенту під час перегляду сторінки.

Отримані результати демонструють високу продуктивність нашого фінансового веб-додатку. Середній час завантаження основних сторінок не перевищував 2 секунд, що є важливим показником швидкості завантаження. Обробка запитів до бази даних здійснювалася в межах 100 мілісекунд, що вказує на ефективну оптимізацію роботи з базою даних. Висока продуктивність додатку забезпечує швидке завантаження та інтерактивність, що є критично важливим для зручності користувачів.

Оцінка ефективності розробленого веб-додатку показала, що система успішно виконує всі основні функції, має високу продуктивність і забезпечує належний рівень безпеки. Користувачі позитивно оцінили зручність використання додатку та відзначили його корисність для управління фінансами.

Отримані результати свідчать про успішність реалізації проекту та дозволяють робити висновки про його готовність до впровадження та використання в реальних умовах.

3.5 Висновки до третього розділу

Третій розділ роботи був присвячений детальному опису процесу розробки веб-додатку для управління фінансами на базі мови програмування Python та фреймворку Flask. Описано архітектуру додатку, яка включає серверну частину, базу даних та клієнтську частину. Розробка клієнтської частини включала створення інтерфейсів користувача з використанням HTML, CSS та JavaScript. Було створено адаптивний дизайн, який забезпечує зручність використання додатку на різних пристроях.

Використання бібліотеки Chart.js для візуалізації даних дозволило створити інтерактивні графіки, які допомагають користувачам аналізувати свої фінансові дані. Було розглянуто альтернативні бібліотеки, такі як D3.js та Highcharts, але обрано Chart.js через її простоту та інтерактивність. Процес розробки програмних модулів включав створення CRUD функціональності для управління фінансовими транзакціями. Користувачі можуть додавати, редагувати, видаляти та переглядати свої фінансові операції, що забезпечує повний контроль над фінансами.

Важливим етапом було забезпечення безпеки фінансових даних користувачів. Було реалізовано шифрування даних, багаторівневу автентифікацію та інші заходи для захисту конфіденційної інформації від несанкціонованого доступу. Розділ також включав опис процесу тестування та налагодження веб-додатку. Було проведено функціональні та нефункціональні тести, що дозволило виявити та виправити помилки, а також забезпечити стабільну роботу системи.

Було оцінено ефективність розробленого веб-додатку на основі зворотного зв'язку від користувачів та результатів тестування. Додаток показав високу продуктивність, зручність використання та безпеку, що підтвердило доцільність обраних рішень та технологій. Загалом, проведене тестування та оцінка ефективності веб-додатку дозволили переконатися у відповідності

розробленого продукту вимогам користувачів та його готовності до впровадження в реальних умовах.

У процесі розробки веб-додатку для управління фінансами, також, було досягнуто значних результатів, що підтверджують ефективність обраного підходу та технологій. Використання Flask у якості серверного фреймворку дозволило створити легкий та гнучкий додаток, що відповідає сучасним вимогам до веб-розробки. Застосування SQLite для зберігання даних забезпечило простоту налаштування та високу продуктивність для невеликих обсягів даних, що робить систему придатною для широкого кола користувачів.

Важливим результатом роботи стало також створення зручного та інтуїтивно зрозумілого інтерфейсу користувача. Використання сучасних технологій, таких як HTML, CSS та JavaScript, дозволило розробити інтерфейс, який не тільки забезпечує зручність у використанні, але й сприяє підвищенню продуктивності користувачів. Інтерактивні графіки та візуалізація даних за допомогою Chart.js дозволяють легко аналізувати фінансову інформацію та приймати обґрунтовані рішення щодо управління фінансами.

Загалом, третій розділ демонструє, що розробка веб-додатку була успішно виконана, а обрані архітектурні рішення, технології та методи забезпечили створення надійного, безпечного та зручного інструменту для управління фінансами. Результати роботи підтверджують можливість подальшого розвитку та впровадження цього додатку у практичну діяльність, що буде корисним для широкого кола користувачів.

ВИСНОВКИ

Фінансова грамотність та ефективне управління особистими фінансами є вкрай важливими аспектами життя сучасної людини. З кожним роком все більше людей усвідомлюють необхідність контролювати свої доходи та витрати, планувати бюджет, здійснювати інвестиції та досягати фінансових цілей. Ефективне управління фінансами допомагає розвинути навички планування, контролю боргів та прийняття обґрунтованих рішень щодо управління коштами. Крім того, зростає популярність активного відстеження доходів та витрат, оскільки це дозволяє краще розуміти фінансові звички та приймати обґрунтовані рішення.

У рамках дослідження було розроблено функціональний і зручний у використанні веб-додаток для ефективного управління особистими фінансами на базі мови програмування Python та веб-фреймворку Flask. Застосування новітніх технологій, включаючи SQLite для реалізації бази даних, HTML, CSS та JavaScript для створення інтерактивного користувацького інтерфейсу, а також бібліотеки Chart.js для візуалізації фінансових даних за допомогою динамічних графіків, дозволило створити масштабований, високопродуктивний і безпечний програмний продукт, який повністю задовольняє потреби користувачів.

Ретельний аналіз існуючих рішень у сфері управління фінансами, як локальних програм, так і хмарних сервісів та мобільних додатків, уможливив визначення ключових функціональних можливостей, які були успішно інтегровані у розроблений додаток. До них належать зручний облік доходів і витрат, гнучке планування бюджету, докладна категоризація фінансових транзакцій та наочна візуалізація фінансової статистики у вигляді діаграм і графіків. Врахування передових принципів дизайну користувацького інтерфейсу та досвіду (UI/UX) забезпечило створення інтуїтивно зрозумілого й адаптивного інтерфейсу для використання на різних пристроях.

Процес розробки програмної архітектури був ретельно продуманий з акцентом на високу продуктивність, масштабованість та безпеку додатку. Використання Flask як основи серверної частини гарантувало ефективну обробку запитів, тоді як SQLite забезпечила надійне та швидке зберігання даних. Крім того, ретельна реалізація валідації вхідних даних як на клієнтській, так і на серверній стороні, суттєво підвищила безпеку додатку, що є критично важливим для фінансових програм.

Комплексне тестування функціональності додатку на завершальному етапі дослідження продемонструвало високі показники ефективності, надійності та зручності розробленого рішення як інструменту для контролю особистих фінансів. Отримані результати повністю підтвердили доцільність використання обраного стеку технологій для реалізації подібних веб-додатків та успішне досягнення поставлених цілей дослідження.

Перспективи подальшого вдосконалення створеного додатку охоплюють інтеграцію багатокористувацької підтримки, забезпечення можливості прямого імпорту фінансових транзакцій з банківських сервісів, розширення аналітичних можливостей та імплементацію модулів для прогнозування фінансових показників за допомогою передових алгоритмів машинного навчання. Загалом, результати дослідницької роботи демонструють успішну реалізацію поставлених завдань, високий потенціал удосконалення розробленого програмного забезпечення та перспективи його промислового впровадження в якості ефективного інструменту для зручного й безпечного управління особистими фінансами.

У ході виконання кваліфікаційної роботи були виконані усі завдання, визначені на початку дослідження. Кожне завдання було виконано наступним чином:

• проведено огляд сучасних методів та програмних засобів для фінансового управління. Проаналізовано популярні веб-додатки такі як Quicken, Microsoft Money, GnuCash, Mint, Personal Capital та You Need a Budget

(YNAB). Виконано порівняльний аналіз цих додатків за функціональністю, зручністю використання, безпекою даних та вартістю;

• обрано архітектурний підхід MVC для розробки додатку. Проектування архітектури включало визначення компонентів системи: серверна частина на Flask, база даних SQLite та клієнтська частина з використанням HTML, CSS та JavaScript. Розроблено дизайн інтерфейсу користувача з урахуванням принципів UI/UX;

• реалізовано серверну частину додатку на Flask, забезпечуючи обробку HTTP-запитів, маршрутизацію та шаблонізацію. Використано SQLite для зберігання даних. Реалізовано базову функціональність додатку, включаючи реєстрацію користувачів, введення та категоризацію фінансових транзакцій, планування бюджету;

• проведено тестування додатку на відповідність функціональним вимогам. Виконано юніт-тестування компонентів системи за допомогою pytest. Проведено інтеграційне тестування для перевірки взаємодії між компонентами системи та забезпечення безперебійної роботи;

• проведено оцінку ефективності додатку з точки зору продуктивності та зручності використання. Проведено опитування користувачів для збору зворотного зв’язку щодо функціональності та інтерфейсу додатку. Результати показали, що додаток забезпечує високий рівень зручності використання та задовольняє потреби користувачів у управлінні фінансами.

Таким чином, усі поставлені завдання були успішно виконані, що дозволило створити зручний, функціональний та безпечний веб-додаток для управління особистими фінансами. Отримані результати мають практичне значення, оскільки сприяють автоматизації процесів управління фінансами, підвищуючи ефективність і зручність користування для користувачів.

СПИСОК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ

1. Empower (formerly Personal Capital) vs. Mint vs. Quicken Review – Which Financial App is Best. URL: https://www.roboadvisorpros.com/personal-capital-vs-mint-vs-quicken-review-best-financial-app/

2. Mint vs. Empower vs. YNAB. URL: https://moneywise.com/managing-money/budgeting/mint-vs-empower-vs-ynab

3. How to Choose the Best Personal Finance Software and Apps. URL: https://www.nerdwallet.com/article/finance/best-personal-finance-software

4. Домашня бухгалтерія – 7 найкращих програм. URL: https://financer.com/ua/blog/best-apps/

5. Melton, L. (2019). Financial Planning & Analysis and Performance Management. Wiley. 656 p.

6. Envelope Budgeting.URL: https://www.financestrategists.com/financial-advisor/personal-finance/envelope-budgeting/

7. How to use the envelope budget system. URL: https://www.capitalone.com/learn-grow/money-management/envelope-budget-system/

8. Pay Yourself First Budget: Definition & How It Works. URL: https://budgetmethod.com/pay-yourself-first-budget-definition-how-it-works/

9. Pay Yourself First: What It Means & How To Do It. URL: https://www.supermoney.com/pay-yourself-first

10. Personal Finance Apps: Harnessing Technology for Financial Management. URL: https://www.graygroupintl.com/blog/personal-finance-apps

11. 14 Financial Management Tools that Every Business Must Have. URL: https://www.cflowapps.com/top-financial-management-tools/

12. 10 зручних додатків для контролю особистого бюджету. URL: https://minfin.com.ua/ua/2019/07/24/38514719/

13. The 15 best financial management software tools for businesses in 2024. URL: https://www.cubesoftware.com/blog/best-financial-management-software-tools

14. Поради щодо управління особистими фінансами. URL: https://geer.com.ua/upravlinnia-osobystymy-finansamy/

15. Методи фінансового аналізу. URL: https://finances.in.ua/metody-finansovoho-analizu/

16. The transition from traditional banking to mobile internet finance. URL: https://jfin-swufe.springeropen.com/articles/10.1186/s40854-017-0062-0

17. A Not-So-Brief History of Everything Spreadsheets. URL: https://www.spreadsheet.com/spreadsheets

18. Freshbooks vs Quickbooks vs Xero - Which Is Best? URL: https://www.reconcile.ly/blog/freshbooks-vs-quickbooks-vs-xero

19. International Payment Methods: Simple Guide. URL: https://wise.com/us/blog/international-payment-methods

20. Blockchain in Financial Services: Enhancing Security, Transparency, and Efficiency. URL: https://www.technology-innovators.com/blockchain-in-financial-services-enhancing-security-transparency-and-efficiency/

21. E. Hendrickson. Explore It! Reduce Risk and Increase Confidence with Exploratory Testing. Pragmatic Bookshelf. 2013. 175 p.

22. Sam Newman. Building Microservices: Designing Fine-Grained Systems. O'Reilly Media. 2021. 500 p.

23. UI/UX ДИЗАЙН: ПРИНЦИПИ ТА МЕТОДИ СТВОРЕННЯ ЗРУЧНОГО ІНТЕРФЕЙСУ КОРИСТУВАЧА. URL: https://whileweb.com/uk/blog/uiux-dizajn-principi-ta-metodi-stvorennya-zruchnogo-interfejsu-koristuvacha/

24. Adham Dannaway. Practical UI. 2022. 282p.

25. Steps of Software Development Process We Used to Build 200+ Products. URL: https://relevant.software/blog/software-development-process/

26. Flask’s documentation. URL: https://flask.palletsprojects.com/en/3.0.x/

27. Suyufan bin Uzayar. Mastering Visual Studio Code: A Beginner's Guide. Taylor & Francis. 2022. 246 p.

28. HTML: HyperText Markup Language. URL: https://devdocs.io/html/

29. CSS (Cascading Style Sheets). URL: https://w3schoolsua.github.io/index.html#gsc.tab=0

30. Comparing Database Management Systems: MySQL, PostgreSQL, MSSQL Server, MongoDB, Elasticsearch, and others. URL: https://www.altexsoft.com/blog/comparing-database-management-systems-mysql-postgresql-mssql-server-mongodb-elasticsearch-and-others/

31. Theurer, K. (2021). Practical Web Development with Flask and Python. Packt Publishing. 400 p.

32. Basics of MVC, Microservices, and Spring Boot. URL: https://www.educative.io/courses/developing-microservices-with-spring-boot/basics-of-mvc-microservices-and-spring-boot

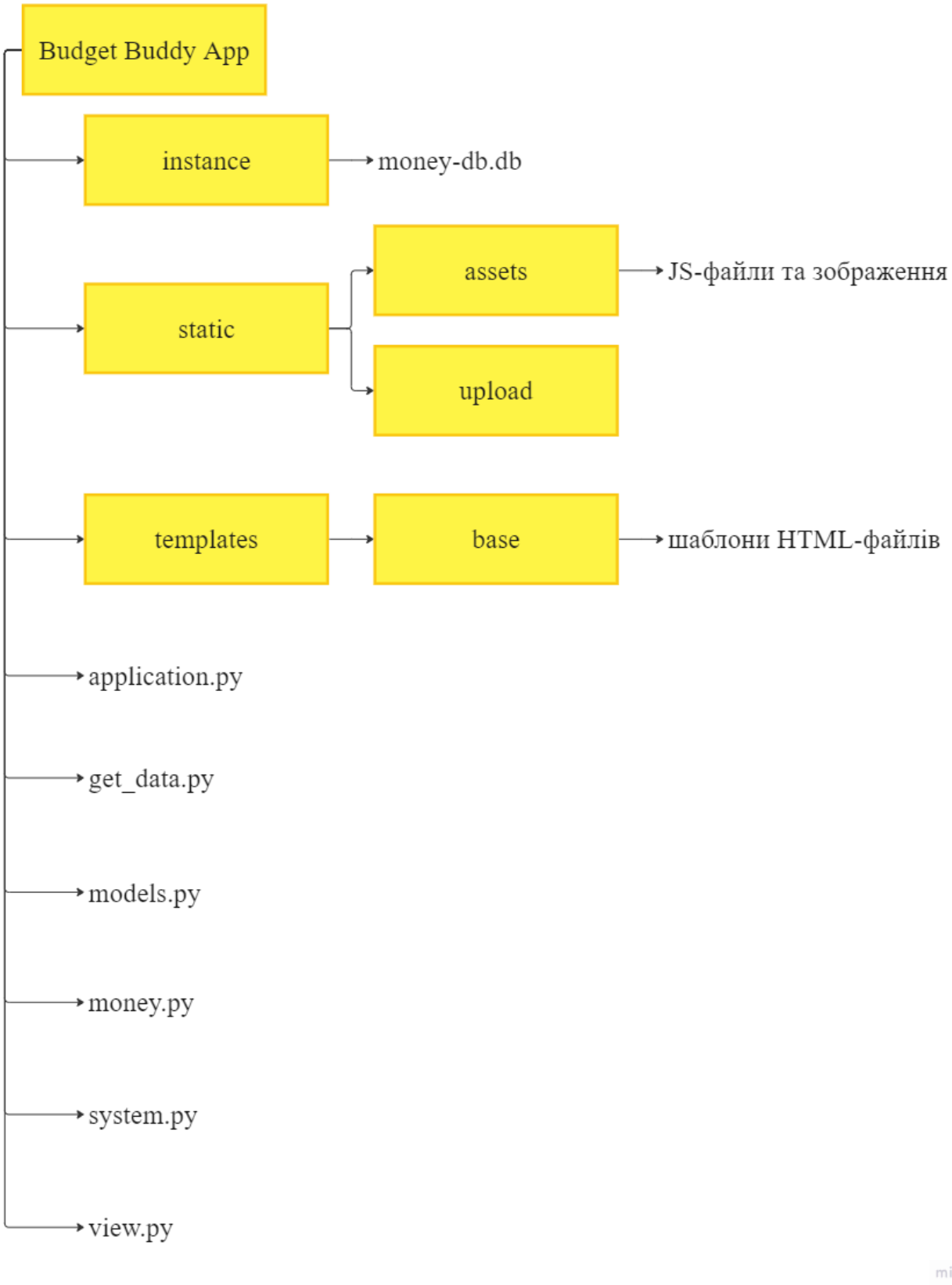
33. MVC vs. Microservices: Understanding their Architecture. URL: https://wisdomplexus.com/blogs/mvc-vs-microservices/

34. Chart.js. URL: https://www.chartjs.org/docs/latest/

35. Comparison of Popular JavaScript Charting Libraries. URL: https://www.cybrosys.com/blog/comparison-of-popular-javascript-charting-libraries

36. S. Krug. Don't Make Me Think. – New Riders, 2014. – 216 p.

ДОДАТОК А


Budget Buddy App
instance
money-db.db
static
assets
JS-файли та зображення
upload
templates
base
шаблони HTML-файлів
application.py
get_data.py
models.py
money.py
system.py
view.py
miro

ДОДАТОК Б

ЛІСТИНГИ ФАЙЛІВ, ЩО ВІДНОСЯТЬСЯ ДО СЕРВЕРНОЇ ЧАСТИНИ ДОДАТКУ

Б. 1 Лістинг файлу «application.py»

```
from flask import Flask
from flask_sqlalchemy import SQLAlchemy
import os

app = Flask(__name__)
app.config["SQLALCHEMY_DATABASE_URI"] = "sqlite:///money-db.db"
app.config['SQLALCHEMY_TRACK_MODIFICATIONS'] = False
#app.config.from_object(Configuration)


    # Шлях для зберігання завантажених аватарів
    UPLOAD_FOLDER = 'static/upload'
    ALLOWED_EXTENSIONS = {'png', 'jpg', 'jpeg', 'gif'}

    app.config['UPLOAD_FOLDER'] = UPLOAD_FOLDER

    app.secret_key = 'fdgfh78@#5?>gfhf89dxv06k'

    db = SQLAlchemy(app)

    if not os.path.exists(UPLOAD_FOLDER):
        os.makedirs(UPLOAD_FOLDER)
```

Б. 2 Лістинг файлу «get_data.py»

```
from models import *
from sqlalchemy import func
from datetime import datetime, timedelta
from flask import jsonify

class AccountInfo:
    #Клас для операції з аккаунтом користувача
```

```
8.    def __init__(self, user_id):
9.        # Ініціалізує об'єкт AccountInfo з ідентифікатором користувача
10.            self.user = users.query.get(user_id)
11.
12.        def get_user_info(self):
13.            # Повертає інформацію про користувача, включаючи ID, ім'я,
   аватар та баланс
14.            return {
15.                'id': self.user.id,
16.                'user_name': self.user.user_name,
17.                'user_avatar': self.user.user_avatar,
18.                'balance': int(self.user.balance)
19.            }
20.
21.        def add_to_balance(self, add_sum):
22.            # Додає вказану суму до балансу користувача
23.            self.user.balance += float(add_sum)
24.            db.session.commit()
25.
26.        def diff_balance(self, add_sum):
27.            # Віднімає вказану суму з балансу користувача
28.            self.user.balance -= float(add_sum)
29.            db.session.commit()
30.
31.
32.    class TransactionInfo:
33.        #Клас для роботи з транзакціями та отримання інформації про них
34.        def __init__(self, user_id, transaction_type):
35.            # Ініціалізує об'єкт TransactionInfo з ідентифікатором
   користувача та типом транзакції
36.            self.user_id = user_id
37.            self.transaction_type = transaction_type
38.
39.        def get_transaction_list(self):
40.            # Отримує список транзакцій на основі ідентифікатора
   користувача та типу транзакції
41.            transactions =
   transactions_list.query.filter_by(user_id=self.user_id,
   pay_type=self.transaction_type).all()
42.            return [{
```

```
43.                 'id': transaction.id,
44.                 'created_at': transaction.created_at.strftime('%Y-%m-
    %d'),
45.                 'pay_type': transaction.pay_type,
46.                 'pay_name': transaction.pay_name,
47.                 'pay_category': transaction.pay_category,
48.                 'pay_sum': int(transaction.pay_sum),
49.                 'pay_comment': transaction.pay_comment,
50.                 'pay_icon': transaction.pay_icon
51.             } for transaction in transactions]
52.
53.         def get_total(self):
54.             # Отримує загальну суму транзакцій для вказаного користувача
    та типу транзакції
55.             total_sum =
    transactions_list.query.with_entities(func.sum(transactions_list.pay_sum)
    ).filter_by(
56.                 user_id=self.user_id,
    pay_type=self.transaction_type).scalar()
57.             return int(total_sum) if total_sum is not None else 0
58.
59.         def add_transaction(self, user_id, created_at, pay_type,
    pay_category, pay_name, pay_comment, pay_sum, pay_icon):
60.             # Додає нову транзакцію для користувача з вказаними даними
61.             transaction = transactions_list(user_id=user_id,
    created_at=created_at, pay_type=pay_type, pay_category=pay_category,
62.                                             pay_name=pay_name,
    pay_comment=pay_comment, pay_sum=pay_sum, pay_icon=pay_icon)
63.             db.session.add(transaction)
64.             db.session.commit()
65.
66.             account_info = AccountInfo(user_id)
67.             #Якщо тим транзакції - дохід - додаємо до балансу, у всіх
    протилежних випадках віднімаємо
68.             if pay_type == 'income':
69.                 account_info.add_to_balance(pay_sum)
70.             else:
71.                 account_info.diff_balance(pay_sum)
72.             return transaction.id
73.
```

```
74.
75.         def get_dashboard_info(self):
76.             # Отримує інформацію для панелі приладів: баланс, доходи та
    витрати користувача
77.             user_balance = users.query.get(self.user_id).balance
78.             """Цей код використовує функцію next() для пошуку суми
    доходів і витрат у списку кортежів sums.
79.                 Кожний кортеж містить тип транзакції та відповідну суму.
    Вираз у круглих дужках створює генератор,
80.                 який перевіряє кожен кортеж у списку. Для кожного
    кортежу виконується перевірка типу
81.                 транзакції (if pay_type == 'income' або if pay_type ==
    'expenses').
82.                 Якщо знайдено кортеж з типом 'income', то функція next()
    повертає відповідну суму доходів (sum_value).
83.                 Якщо такого кортежу немає, повертається значення за
    замовчуванням, у даному випадку - 0.
84.                 Такий підхід дозволяє швидко знайти суму доходів і
    витрат без перегляду всього списку.
85.             """
86.             sums =
    transactions_list.query.with_entities(transactions_list.pay_type,
    func.sum(transactions_list.pay_sum)).filter_by(user_id=self.user_id).grou
    p_by(transactions_list.pay_type).all()
87.             user_income = next((sum_value for pay_type, sum_value in
    sums if pay_type == 'income'), 0)
88.             user_expenses = next((sum_value for pay_type, sum_value in
    sums if pay_type == 'expenses'), 0)
89.             return {
90.                 'balance': int(user_balance),
91.                 'user_income': int(user_income),
92.                 'user_expenses': int(user_expenses)
93.             }
94.
95.         def get_last_transactions_category(self):
96.             # Отримує інформацію про останні транзакції за категорією
97.             get_distinct_category =
    transactions_list.query.with_entities(
98.                 transactions_list.pay_category,
99.                 transactions_list.pay_type,
```

```
100.                 func.count(transactions_list.pay_category).label('count'
    ),
101.                 func.sum(transactions_list.pay_sum).label('sum_payment')
    ,
102.                 category_list.category_id,
103.                 category_list.category_name,
104.                 category_list.category_avatar
105.             ).join(
106.                 category_list,
107.                 category_list.category_name ==
    transactions_list.pay_category
108.             ).filter(
109.                 transactions_list.user_id==self.user_id
110.             ).group_by(
111.                 transactions_list.pay_category,
    transactions_list.pay_type
112.             ).all()
113.
114.             categories = []
115.             for category in get_distinct_category:
116.                 categories.append({
117.                     'category_name': category.category_name,
118.                     'category_type': category.pay_type,
119.                     'category_sum': int(category.sum_payment)
120.                 })
121.             return categories
122.
123.         def get_weekly_data(self):
124.             # Отримує щотижневі дані про доходи та витрати користувача
125.             current_date = datetime.now().date()
126.             dates = [(current_date - timedelta(days=i)).strftime('%Y-%m-
    %d') for i in range(7, -1, -1)]
127.
128.             incomes = []
129.             expenses = []
130.             for date in dates:
131.                 created_at = datetime.strptime(date, '%Y-%m-%d').date()
132.                 income =
    transactions_list.query.with_entities(func.sum(transactions_list.pay_sum)
    ).filter_by(
```

```
133.                     user_id=self.user_id, pay_type='income',
    created_at=created_at).scalar() or 0
134.                 expense =
    transactions_list.query.with_entities(func.sum(transactions_list.pay_sum)
    ).filter_by(
135.                     user_id=self.user_id, pay_type='expenses',
    created_at=created_at).scalar() or 0
136.                 print(date, income, expense)
137.                 incomes.append(income)
138.                 expenses.append(expense)
139.             return dates, incomes, expenses
140.
141.         def get_category_data(self):
142.             # Отримати дані категорій з бази даних
143.             get_distinct_category =
    transactions_list.query.with_entities(
144.                 transactions_list.pay_category,
145.                 transactions_list.pay_type,
146.                 func.count(transactions_list.pay_category).label('count'
    ),
147.                 func.sum(transactions_list.pay_sum).label('sum_payment')
148.             ).filter_by(user_id=self.user_id).group_by(transactions_list
    .pay_category, transactions_list.pay_type).all()
149.
150.             # Створити списки для категорій доходів та витрат
151.             income_categories = []
152.             expenses_categories = []
153.             for category in get_distinct_category:
154.                 category_name = category.pay_category
155.                 category_count = category.count
156.                 category_sum = category.sum_payment
157.                 if category.pay_type == 'income':
158.                     income_categories.append({'category': category_name,
    'count': category_count, 'sum': category_sum})
159.                 else:
160.                     expenses_categories.append({'category':
    category_name, 'count': category_count, 'sum': category_sum})
161.             print(income_categories, expenses_categories)
162.             return income_categories, expenses_categories
```

Б. 3 Лістинг файлу «models.py»

```
from application import app, db
from datetime import datetime
from sqlalchemy.orm import relationship

class users(db.Model):
    id = db.Column(db.Integer, primary_key=True)
    user_email = db.Column(db.String(255), nullable=False)
    user_pass = db.Column(db.String(255), nullable=False)
    user_name = db.Column(db.String(255), nullable=False)
        user_avatar = db.Column(db.String(255), nullable=False)
        balance = db.Column(db.Float, nullable=False)
        api_key = db.Column(db.String(255), nullable=False)
        # Додайте зв'язок між користувачем і його транзакціями
        transactions = db.relationship('transactions_list',
  backref='user', lazy=True)

        def __init__(self, *args, **kwargs):
            super(users, self).__init__(*args, **kwargs)

        def is_active(self):
            return True

        def get_id(self):
            return self.id

        def is_authenticated(self):
            return self.authenticated

        def is_anonymous(self):
            return False

        def __repr__(self):
            return "user id: {}".format(self.id)


    class transactions_list(db.Model):
        id = db.Column(db.Integer, primary_key=True)
```

```
37.         user_id = db.Column(db.Integer, db.ForeignKey('users.id'),
    nullable=False)
38.         created_at = db.Column(db.Date)
39.         pay_type = db.Column(db.String(255))
40.         pay_category = db.Column(db.String(255))
41.         pay_sum = db.Column(db.Float)
42.         pay_name = db.Column(db.String(255))
43.         pay_comment = db.Column(db.String(255))
44.         pay_icon = db.Column(db.String(255))
45.
46.         def __init__(self, *args, **kwargs):
47.             super(transactions_list, self).__init__(*args, **kwargs)
48.
49.         def __repr__(self):
50.             return f"id: {self.id}\n"
51.
52.     class category_list(db.Model):
53.         id = db.Column(db.Integer, primary_key=True)
54.         category_id = db.Column(db.Integer, db.ForeignKey('users.id'),
    nullable=False)
55.         category_type = db.Column(db.String(255))
56.         category_name = db.Column(db.String(255))
57.         category_avatar = db.Column(db.String(255))
58.
59.         def __init__(self, *args, **kwargs):
60.             super(category_list, self).__init__(*args, **kwargs)
61.
62.         def __repr__(self):
63.             return f"id: {self.id}\n"
```

Б. 4 Лістинг файлу «money.py»

```
1.from application import app, db
2.import view
3.
4.
5.if __name__ == "__main__":
6.     app.run(host='0.0.0.0', debug=False, threaded=True)
```

Б. 5 Лістинг файлу «system.py»

```
import secrets
import string

def generate_api_key(length=16):
    # Генеруємо випадковий рядок з великих літер, малих літер і цифр
    characters = string.ascii_letters + string.digits
    api_key = ''.join(secrets.choice(characters) for _ in range(length))
    return api_key
```

Б. 6 Лістинг файлу «view.py»

```
from flask import Flask, request
import get_data
from application import app, db, ALLOWED_EXTENSIONS, UPLOAD_FOLDER
import json
from datetime import timedelta, datetime
from flask import Flask, abort, session, request, app, redirect, url_for, jsonify
from models import *
import secrets
import requests
    import system
    from werkzeug.security import generate_password_hash, check_password_hash
    from flask import render_template, request, flash, session, url_for, redirect, jsonify, Response, g, make_response
    from flask_login import LoginManager, login_user, current_user, login_required, logout_user
    from werkzeug.utils import secure_filename
    import os

    login_manager = LoginManager()
    # Ініціалізація LoginManager з Flask app
    login_manager.init_app(app)


    @login_manager.user_loader
```

```
23.     def load_user(user_id):
24.         try:
25.             return users.query.get(user_id)
26.         except Exception as e:
27.             db.session.rollback()
28.             return users.query.get(user_id)
29.
30.
31.     @app.route('/')
32.     def start_page():
33.         return redirect(url_for('login'))
34.
35.
36.     # Опис функції для визначення активного пункту меню
37.     def is_active(page):
38.         return request.path == page
39.
40.
41.     @app.route('/logout')
42.     @login_required
43.     def logout():
44.         logout_user()
45.         return redirect('/login')
46.
47.
48.     @app.route('/login', methods=['POST', 'GET'])
49.     def login():
50.         if request.method == 'POST':
51.             email = request.form.get('email')
52.             password = request.form.get('password')
53.
54.             user = users.query.filter_by(user_email=email).first()
55.             print(email, password)
56.             if user and check_password_hash(user.user_pass, password):
57.                 login_user(user, remember=True)
58.                 return redirect(url_for('home_page'))
59.
60.             flash("Користувач з такими даними не знайдений", 'success')
61.             return redirect(url_for('login'))
62.
```

```
63.         return render_template('login.html')
64.
65.
66.     @app.route('/registration', methods=['POST', 'GET'])
67.     def registration():
68.         if request.method == 'POST':
69.             if len(request.form['email']) > 4 and
    len(request.form['password']) > 6:
70.                 user_email = request.form['email']
71.                 user_name = request.form['name']
72.                 user_password = request.form['password']
73.                 avatar_file = request.files['avatar']
74.                 hash_pwd = generate_password_hash(user_password)
75.                 old_user =
    users.query.filter_by(user_email=user_email).first()
76.                 if not old_user:
77.                     filename = secure_filename(avatar_file.filename)
78.                     avatar_file.save(os.path.join(app.config['UPLOAD_FOL
    DER'], filename))
79.                     avatar_path = os.path.join(UPLOAD_FOLDER, filename)
80.                     create_account = users(user_email=user_email,
    user_pass=hash_pwd, user_name=user_name,
81.                                            user_avatar=avatar_path, balance=0,
    api_key=system.generate_api_key())
82.                     db.session.add(create_account)
83.                     db.session.commit()
84.                     # Отримання айді нового користувача
85.
86.                     return redirect(url_for('login'))
87.                 else:
88.                     flash('Такий Email вже зареєстрован', 'error')
89.                 db.session.close()
90.             else:
91.                 flash('Пароль має бути більш ніж 6 символів', 'error')
92.         return render_template('register.html')
93.
94.
95.     @app.route('/home')
96.     @login_required
97.     def home_page():
```

```
98.         user = current_user
99.         user_id = user.id
100.        api_key = user.api_key
101.
102.        user_info = get_data.AccountInfo(user_id).get_user_info()
103.
104.        transactions_cls = get_data.TransactionInfo(user_id, 'all')
105.        dashboard_info = transactions_cls.get_dashboard_info()
106.        charts = transactions_cls.get_weekly_data()
107.        transactions_category =
    transactions_cls.get_last_transactions_category()
108.        return render_template('main.html', user_info=user_info,
    dashboard_info=dashboard_info,
109.                               charts=charts, is_active=is_active,
    api_key=api_key, transactions_category=transactions_category)
110.
111.
112.    @app.route('/statistic')
113.    @login_required
114.    def statistic_page():
115.        user = current_user
116.        user_id = user.id
117.        api_key = user.api_key
118.
119.        user_info = get_data.AccountInfo(user_id).get_user_info()
120.
121.        transactions_cls = get_data.TransactionInfo(user_id, 'all')
122.        dashboard_info = transactions_cls.get_dashboard_info()
123.        charts = transactions_cls.get_category_data()
124.        print(charts)
125.        return render_template('statistic.html', user_info=user_info,
    dashboard_info=dashboard_info,
126.                               is_active=is_active,
    income_categories=charts[0], expenses_categories=charts[1])
127.
128.
129.    @app.route('/income')
130.    @login_required
131.    def income_page():
132.        user = current_user
```

```
133.         user_id = user.id
134.         api_key = user.api_key
135.
136.         transaction_cls = get_data.TransactionInfo(user_id, 'income')
137.
138.         user_info = get_data.AccountInfo(user_id).get_user_info()
139.         transaction_list = transaction_cls.get_transaction_list()
140.         total_income = transaction_cls.get_total()
141.         return render_template('incomes.html', user_info=user_info,
    api_key=api_key,
142.                                transaction_list=transaction_list,
    len_transactions=len(transaction_list),
143.                                total_income=total_income,
    is_active=is_active)
144.
145.
146.     @app.route('/expenses')
147.     @login_required
148.     def expenses_page():
149.         user = current_user
150.         user_id = user.id
151.         api_key = user.api_key
152.
153.         transaction_cls = get_data.TransactionInfo(user_id, 'expenses')
154.
155.         user_info = get_data.AccountInfo(user_id).get_user_info()
156.         transaction_list = transaction_cls.get_transaction_list()
157.         total_income = transaction_cls.get_total()
158.         return render_template('expenses.html', user_info=user_info,
    api_key=api_key,
159.                                transaction_list=transaction_list,
    len_transactions=len(transaction_list),
160.                                total_income=total_income,
    is_active=is_active)
161.
162.
163.     @app.route('/api/add_transaction', methods=['POST'])
164.     def add_transaction():
165.         data = request.json
166.         print(data)
```

```
167.         # Перевірка наявності обов'язкових полів
168.         if 'name' not in data or 'number' not in data or 'date' not in
     data:
169.             return jsonify({'error': 'Missing required fields'}), 400
170.
171.         get_user_info =
     users.query.filter_by(api_key=data['api_key']).first()
172.
173.         category_info =
     category_list.query.filter_by(category_id=data.get('category', ''),
     category_type=data['pay_type']).first()
174.
175.
176.         # Конвертуємо рядок у об'єкт datetime
177.         dateObject = datetime.strptime(data['date'], '%Y-%m-%d')
178.
179.         create_transaction = get_data.TransactionInfo(get_user_info.id,
     data['pay_type']).add_transaction(
180.                                             user_id=get_user_info.id,
     created_at=dateObject, pay_type=data['pay_type'],
181.                                             pay_category=category_info.categ
     ory_name, pay_name=data['name'],
182.                                             pay_comment=data.get('textDescri
     ption', ''), pay_sum=data['number'],
183.                                             pay_icon=category_info.category_
     avatar)
184.
185.
186.         return jsonify({'message': 'Transaction added successfully',
     'avatar': category_info.category_avatar,
187.                         'id': create_transaction }), 201
188.
189.
190.     from flask import jsonify
191.
192.     @app.route('/api/delete/transaction/<int:transaction_id>',
     methods=['DELETE'])
193.     def delete_transaction(transaction_id):
194.         api_key = request.headers.get('Authorization')
195.         if api_key:
```

```
196.             api_key = api_key.replace('Bearer ', '')
197.         print(api_key)
198.         get_user_info = users.query.filter_by(api_key=api_key).first()
199.         if get_user_info:
200.             # Далі ви можете видалити транзакцію з бази даних або
    зробити інші операції
201.             transaction_info =
    transactions_list.query.filter_by(id=transaction_id,
    user_id=get_user_info.id)
202.
203.             if transaction_info:
204.                 transaction_sum = transaction_info.first().pay_sum  #
    Отримуємо суму транзакції
205.                 get_user_info.balance += transaction_sum
206.                 transaction_info.delete()
207.                 db.session.commit()
208.
209.                 return jsonify({'message': 'Транзакція успішно
    видалена', 'sum': transaction_sum}), 200
210.             else:
211.                 return jsonify({'error': 'Транзакція не знайдена або ви
    не маєте доступу до цієї транзакції'}), 404
212.         else:
213.             return jsonify({'error': 'Невірний API ключ'}), 403
```