Київський столичний університет імені Бориса Грінченка
Факультет інформаційних технологій та математики
Кафедра комп'ютерних наук

«Допущено до захисту»
Завідувач кафедри комп'ютерних наук
к.т.н. доцент Ірина МАШКІНА

_____
(підпис)
« ___ » _____ 2025 р.

КВАЛІФІКАЦІЙНА РОБОТА
на здобуття освітнього ступеня «бакалавр»
Спеціальність 122 «Комп'ютерні науки
Освітня програма 122.00.01 Інформатика

Тема роботи: **РОЗРОБКА МОБІЛЬНОГО ДОДАТКУ ДЛЯ ПЛАНУВАННЯ ОСОБИСТОГО БЮДЖЕТУ**

Виконав
студент групи __Інб-2-21-4.0д_____
(шифр академічної групи)
_____Шемет Євгеній Ігорович_____
(прізвище, ім'я, по батькові)

_____
(підпис)

Науковий керівник
_____к. т. н., доцент_____
(науковий ступінь, наукове звання)
_____Носенко Т. І._____
(прізвище, ініціали)

_____
(підпис)

Київ – 2025

Київський столичний університет імені Бориса Грінченка
Факультет інформаційних технологій та математики
Кафедра комп'ютерних наук

«Затверджую»
Завідувач кафедри комп'ютерних наук
к.т.н. доцент Ірина МАШКІНА

_____
(підпис)

ЗАВДАННЯ НА ВИКОНАННЯ КВАЛІФІКАЦІЙНОЇ РОБОТИ

студенту групи _Інб-2-21-4.0д___

Шемету Євгенію Ігоровичу

Тема роботи: Розробка мобільного додатку для планування особистого бюджету_

1. Вихідні дані: інформація про методи планування особистого бюджету, технологічні ресурси для розробки_застосунка

2. Основні завдання: аналіз існуючих методів та засобів планування особистого бюджету, аналіз існуючих_ програмних продуктів та онлайн сервісів для планування особистого бюджету, визначення їх переваг та недоліків; вибір технологій, моделей зберігання даних та розробка мобільного застосунку; тестування розробленого мобільного застосунку та оцінка інформативності, точності та зручності використання.

3. Пояснювальна записка: __64___

4. Графічні матеріали: _____11_

5. Додатки: _____3_____

6. Строк подання роботи на кафедру « ___ » _____ 2025 __ р.

Науковий керівник
Носенко Т. І., _к. т. н., доцент
_____
(підпис викладача)

Завдання прийняв до виконання
студент Шемет Є.І.
_____
(підпис студента)
20.10.2024

ІНДИВІДУАЛЬНИЙ ПЛАН
ВИКОНАННЯ КВАЛІФІКАЦІЙНОЇ РОБОТИ

студента групи __Інб-2-21-4.0д_Шемета Євгенія Ігорович

’Тема роботи: **Розробка мобільного додатку для планування особистого бюджету**

№ зп	Види роботи, завдань, заходів	Термін виконання	Відмітка про виконання
1	Постановка задачі. Предмет, мета, завдання	30.10.24	виконано
2	Робота з джерелами за темою роботи	30.11.24	виконано
3	Визначення змісту кваліфікаційної роботи	30.12.24	виконано
4	Аналітичний огляд: - сучасний стан напряму дослідження - визначення проблеми - обґрунтування актуальності дослідження - уточнення постановки завдання - ознайомлення з теоретичними основами	30.01.25	виконано
5	Розробка плану роботи над створенням додатку	30.02.25	
	- аналіз існуючих методів -аналіз існуючих сервісів для планування бюджету - аналіз технологій для розробки застосунку	30.03.25	виконано
6	Розробка веб-застосунку для планування особистого бюджету - розробка інтерфейсу веб-застосунку - опис реалізованого функціоналу застосунку - тестування розробленого застосунку	30.04.25	виконано
7	Висновки. Оформлення кваліфікаційної роботи	30.04.25	виконано
8	Подання роботи на відзив Рецензування Підготовка до захисту	30.05.25	виконано
12	Захист кваліфікаційної роботи	20.05.25	виконано

РЕФЕРАТ

Дипломна робота містить 64 сторінки, 11 ілюстрацій, 1 таблицю, 3 додатків, 13 джерел за переліком посилань.

В умовах зростаючої потреби у фінансовій грамотності та повсюдного використання мобільних технологій, розробка інтуїтивного додатку для планування особистого бюджету є надзвичайно актуальною. Такий інструмент надає користувачам можливість ефективно контролювати свої витрати, заощаджувати кошти та досягати фінансових цілей, що особливо важливо в нестабільній економічній ситуації

Об'єкт дослідження - є мобільний додаток для планування особистого бюджету.

Предмет дослідження є інструменти розробки програмного забезпечення для планування особистого бюджету, який має інтуїтивний інтерфейс.

Мета дослідження - розробка функціонального мобільного додатку для планування особистого бюджету для покращення фінансового положення користувача завдяки документуванню фінансових операцій та аналізу фінансової поведінки.

Завдання дослідження:

1. Аналіз сучасних мобільних додатків для контроля над власними коштами для визначення їх функціональних можливостей, структури, сильних та слабких сторін.
2. Розробка моделі додатку, структури даних та взаємодії користувача і системи.
3. Розробка інтерфейсу додатку.
4. Реалізація основних функціональних модулів: введення даних про доходи та витрати та їх категоризація, генерація звітів, налаштування сповіщень тощо.
5. Тестування розробленого додатку для виявлення вразливих місць та усунення помилок, оцінка його ефективності та зручності використання.

Висновки. Розробка мобільного додатку для планування особистого бюджету надасть користувачам зручний та ефективний інструмент для управління власними фінансами, сприяючи підвищенню фінансової дисципліни. Впровадження такого додатку допоможе людям краще

розуміти свої доходи та витрати, приймати обґрунтовані фінансові рішення та впевненіше досягати поставлених економічних цілей.

Рекомендації по використанню результатів роботи. Завдяки додатку користувач зможе відслідковувати використання коштів, та бачити негативний чи позитивний фактор від прийняття тих чи інших фінансових рішень.

ЗМІСТ

ВСТУП

Актуальність теми. Планування бюджету є невід'ємною частиною, так як завдяки цьому є можливість чітко розуміти своє фінансове положення, а також покращити фінансову грамотність [1]. Громадяни часто змушені економити на товарах та послугах першої необхідності, що є не доцільним і не раціональним підходом. Допомогти людині вирішити цю важливу проблему можуть мобільні додатки, призначені для ведення особистого бюджету та контролю витрат.

Метою роботи автори визначив розробку функціонального мобільного додатку для планування особистого бюджету для покращення фінансового положення користувача завдяки документуванню фінансових операцій та аналізу фінансової поведінки.

Предметом дослідження є інструменти розробки програмного забезпечення для планування особистого бюджету, який має інтуїтивний інтерфейс. **Об'єктом** дослідження є мобільний додаток для планування особистого бюджету.

Завдання дослідження:

1. Аналіз сучасних мобільних додатків для контроля над власними коштами для визначення їх функціональних можливостей, структури, сильних та слабких сторін.
2. Розробка моделі додатку, структури даних та взаємодії користувача і системи.
3. Розробка інтерфейсу додатку.
4. Реалізація основних функціональних модулів: введення даних про доходи та витрати та їх категоризація, генерація звітів, налаштування сповіщень тощо.

5. Тестування розробленого додатку для виявлення вразливих місць та усунення помилок, оцінка його ефективності та зручності використання [3].

Практична цінність. Завдяки додатку користувач зможе відслідковувати, що і в якій кількості він купив, чи потрібно це було, та таким чином зможе сама бачити негативний чи позитивний фактор від прийняття тих чи інших фінансових рішень.

Апробація роботи. Роботу було апробовано на двох науково-практичних конференціях молодих учених: «Студентський науковий пошук - 2024» , 07.11.24, м. Київ та «Інформаційні технології - 2025», 15.05.2025, м. Київ.

1. Огляд сучасних тенденцій розробки мобільних додатків для планування особистого бюджету

1.1. Аналіз існуючих рішень

Розробка мобільних додатків для контролю особистого бюджету має велике значення в сучасному світі з кількох причин:

Мобільні телефони стали невід'ємною частиною повсякденного життя. Мобільні додатки забезпечують зручний та швидкий доступ до інструментів управління фінансами в будь-який час і в будь-якому місці. Додатки допомагають користувачам краще розуміти свої доходи та витрати, виявляти неефективні витрати та приймати обґрунтовані фінансові рішення. Додатки автоматизують багато рутинних завдань, таких як облік транзакцій, розрахунок балансу та формування звітів, що заощаджує час та зусилля користувачів. Графіки та діаграми, які часто використовуються в додатках, полегшують сприйняття фінансової інформації та допомагають користувачам краще аналізувати свої фінанси. Додатки можуть бути налаштовані відповідно до індивідуальних потреб та

цілей користувачів, наприклад, створення бюджетів для певних категорій витрат або відстеження прогресу досягнення фінансових цілей. Регулярне використання додатків сприяє формуванню корисних фінансових звичок та підвищує відповідальність за управління грошима.

Сучасні додатки забезпечують захист фінансових даних користувачів за допомогою різних заходів безпеки, таких як шифрування даних та біометрична аутентифікація.

Отже, розробка мобільних додатків для контролю особистого бюджету є важливою для надання користувачам ефективних інструментів для управління фінансами, підвищення їхньої фінансової грамотності та покращення їхнього фінансового добробуту.

Як було зазначено вище, планування власного бюджету дозволяє дисципліновано вести своє фінансове положення [1]. Також, таким чином людина може дисциплінувати себе не лише в фінансовому, а й в повсякденному сенсі.

Мобільні додатки для обліку особистого бюджету відрізняються від веб-додатків. Мобільні додатки обліку особистого бюджету та інтернет-додатки (веб-додатки) мають спільну мету – допомогти користувачам керувати своїми фінансами, але вони відрізняються за кількома ключовими аспектами.

1. Доступність та зручність використання:

Мобільні додатки розроблені спеціально для використання на смартфонах і планшетах та забезпечують швидкий і легкий доступ до функцій обліку в будь-який час і в будь-якому місці. Вони часто оптимізовані для сенсорного керування та невеликих екранів та можуть використовувати функції пристрою, такі як камера (для сканування чеків), GPS (для відстеження місця витрат) і біометрична аутентифікація.

Інтернет-додатки доступні через веб-браузер на комп'ютерах, ноутбуках і мобільних пристроях, зазвичай мають ширший функціонал і можливості для аналізу даних, оскільки мають більше місця для відображення інформації. Додатки зручні для роботи з великими обсягами даних і створення детальних звітів.

2. Функціональність:

Мобільні додатки зосереджені на основних функціях, таких як облік доходів і витрат, категоризація та базові звіти та можуть мати обмежені можливості для бюджетування, планування фінансових цілей або інвестицій.

Інтернет-додатки можуть пропонувати розширені функції, такі як: детальний аналіз фінансових даних, планування фінансових цілей на довгострокову перспективу, управління інвестиціями та кредитами, інтеграція з іншими фінансовими інструментами.

3. Автономність та підключення до Інтернету:

Мобільні додатки:

- Можуть працювати в автономному режимі, дозволяючи користувачам вводити дані про витрати навіть без підключення до Інтернету.
- Синхронізація даних відбувається при підключенні до мережі.

Інтернет-додатки:

- Вимагають постійного підключення до Інтернету для доступу до функцій і даних.

4. Технології:

Мобільні додатки:

- Розробляються з використанням мов програмування, таких як Java, Kotlin (для Android), Swift, Objective-C (для iOS) або крос-платформних технологій, таких як React Native або Flutter.
- Можуть використовувати локальні бази даних (наприклад, SQLite) для зберігання даних на пристрої.

Інтернет-додатки:

- Розробляються з використанням веб-технологій, таких як HTML, CSS, JavaScript, а також серверних мов програмування та фреймворків (наприклад, Python/Django, PHP/Laravel, Node.js/Express).
- Зберігають дані на сервері в базах даних (наприклад, MySQL, PostgreSQL).

5. Оновлення та підтримка:

Мобільні додатки оновлюються через магазини додатків (App Store, Google Play). Користувачі повинні встановлювати оновлення вручну або налаштувати автоматичні оновлення.

Інтернет-додатки оновлюються на сервері, і зміни стають доступними користувачам одразу після оновлення.

Підсумовуючи, мобільні додатки забезпечують більшу мобільність і зручність для швидкого обліку витрат, тоді як інтернет-додатки часто пропонують ширший функціонал і можливості для глибокого фінансового аналізу.

1.2. Порівняння функціональності мобільних додатків для планування особистого бюджету

Розглянемо декілька популярних мобільних додатків для планування домашнього бюджету. Важливо зазначити, що точні дані про кількість завантажень постійно змінюються, тому автор наведуеприблизні цифри, актуальні на момент моєї останньої актуалізації.

1. Monefy

- Розробник: Monefy App
- Платформи: iOS, Android
- Основні функції: Облік витрат, категорії, візуалізація
- Особливості:
    - Простий та інтуїтивно зрозумілий інтерфейс.
    - Зручна інфографіка для аналізу витрат.
    - Можливість синхронізації між пристроями.
- Приблизна кількість завантажень: Понад 10 мільйонів (Google Play), точні дані для iOS можуть відрізнятися.
- Середня оцінка користувачів: 4.5 (залежно від магазину додатків та регіону)
- Недоліки:
    - Обмежені можливості аналітики.
    - Відсутність можливості підключення банківських карток у деяких версіях.

2. Money Manager

- Розробник: Realbyte Inc.
- Платформи: iOS, Android

- Основні функції: Облік доходів/витрат, категорії, статистика, прив'язка до рахунків, розпізнавання чеків
- Особливості:
    - Детальна статистика та звіти.
    - Можливість перегляду даних через комп'ютер.
    - Функція розпізнавання чеків для автоматичного введення даних.
- Приблизна кількість завантажень: Понад 10 мільйонів (Google Play), значна кількість завантажень в App Store.
- Середня оцінка користувачів: 4.6
- Недоліки: Велика кількість реклами в безкоштовній версії.

3. Spendee

- Розробник: Spendee a.s.
- Платформи: iOS, Android
- Основні функції: Об'єднання грошових потоків, категорії, бюджетування
- Особливості:
    - Зручний та інтуїтивно зрозумілий інтерфейс.
    - Вбудовані підказки для користувачів.
    - Можливість підключення банківських рахунків для автоматичної синхронізації (у платній версії).
- Приблизна кількість завантажень: Понад 5 мільйонів (Google Play), значна кількість в App Store.

- Середня оцінка користувачів: 4.4
- Недоліки: Обмежений функціонал у безкоштовній версії.

4. Wallet (by BudgetBakers)

- Розробник: BudgetBakers s.r.o.
- Платформи: iOS, Android
- Основні функції: Облік витрат, бюджетування, банківська синхронізація, планування регулярних платежів
- Особливості:
    - Сумісність з великою кількістю банків по всьому світу.
    - Автоматична синхронізація транзакцій.
    - Веб-версія для зручного доступу з комп'ютера.
- Приблизна кількість завантажень: Понад 10 мільйонів (Google Play), значна кількість в App Store.
- Середня оцінка користувачів: 4.6
- Недоліки: Можливі проблеми з конфіденційністю при синхронізації з банками.

5. Money Lover

- Розробник: Finsify
- Платформи: iOS, Android
- Основні функції: Облік витрат/доходів, бюджетування, борги, заощадження, звіти, конвертер валют
- Особливості:
    - Аналіз тенденцій доходів та витрат.

- Багато корисних сервісів для управління фінансами.
- Функції планування заощаджень та відстеження боргів.

- Приблизна кількість завантажень: Понад 10 мільйонів (Google Play), значна кількість в App Store.
- Середня оцінка користувачів: 4.7
- Недоліки: Деякі функції доступні лише в платній версії.

Кількість завантажень та оцінки можуть варіюватися залежно від регіону, магазину додатків та часу. Інформація про авторів та розробників часто доступна в описі додатку в магазині додатків. Більшість цих додатків мають як безкоштовні, так і платні версії з розширеним функціоналом.

Ці додатки допомагають користувачам ефективніше керувати своїми фінансами, відстежувати витрати, планувати бюджет та досягати фінансових цілей.

Основні функціональні можливості:

Усі розглянуті додатки пропонують базовий набір функцій для управління особистими фінансами:

- Облік доходів і витрат: Це ключова функція, що дозволяє користувачам реєструвати всі свої фінансові операції, вказуючи суму, дату, категорію та інші деталі.
- Категоризація: Доходи та витрати класифікуються за категоріями (наприклад, "Продукти", "Транспорт", "Розваги"), що допомагає зрозуміти структуру витрат.
- Звіти та статистика: Додатки генерують звіти та візуалізують дані у вигляді графіків і діаграм, щоб надати користувачам наочне уявлення про їхнє фінансове становище.

Спільне:

- Інтуїтивний інтерфейс: Розробники прагнуть зробити інтерфейс максимально простим і зручним для користувача, щоб полегшити процес введення даних і перегляду інформації.
- Мобільність: Усі ці програми є мобільними, що забезпечує доступ до них у будь-який час і в будь-якому місці.
- Підтримка різних платформ: Більшість популярних додатків доступні як для iOS, так і для Android.
- Безкоштовні та платні версії: Багато додатків пропонують як безкоштовні версії з обмеженим функціоналом, так і платні версії з розширеними можливостями.

Відмінності:

Хоча базові функції схожі, між додатками є певні відмінності:

- Автоматична синхронізація з банками: Деякі додатки (наприклад, Wallet, Spendee) дозволяють автоматично імпортувати транзакції з банківських рахунків, що значно спрощує процес обліку (часто доступно в платних версіях).
- Бюджетування: Функції бюджетування можуть відрізнятися за складністю та можливостями налаштування.
- Управління боргами: Деякі додатки (наприклад, Money Lover) мають спеціальні функції для відстеження боргів і кредитів.
- Розпізнавання чеків: Money Manager пропонує функцію розпізнавання чеків для автоматичного введення даних про витрати.
- Спеціалізовані функції: Деякі додатки можуть мати унікальні функції, орієнтовані на певну аудиторію (наприклад, інструменти для фрілансерів).

Технології:

Інформація про конкретні технології, які використовуються в кожному додатку, часто не є загальнодоступною. Однак, можна зробити деякі загальні висновки:

- Мобільна розробка:
  - Java та Kotlin (для Android)
  - Swift та Objective-C (для iOS)
  - Кросплатформні фреймворки (React Native, Flutter)

  для розробки додатків, що працюють на обох платформах
- Бази даних:
- SQLite (для локального зберігання даних на пристрої)
- Хмарні бази даних (Firebase, інші) для синхронізації та зберігання даних на сервері.
- API:
- RESTful API для обміну даними між додатком і сервером (якщо є).

Важливо зазначити, що розробники постійно оновлюють свої додатки, додаючи нові функції, покращуючи продуктивність і виправляючи помилки.

Порівняльний аналіз мобільних додатків для планування особистого бюджету представлені в Таблиці 1.

Таблиця 1. Порівняльний аналіз мобільних додатків для планування бюджету

Назва додатку	**Платформа**	**Ціна**	**Основні функції**	**Переваги**	**Недоліки**	**Рейтинг**
Monefy	iOS, Android	Безкоштовно, є платна версія	Облік витрат, категорії, візуалізація	Простий інтерфейс, інфографіка	Недостатня аналітика, неможливість підключити картки	4.5

Money Manager	iOS, Android	Безкоштовно, є платна версія	Облік доходів/витрат, категорії, статистика, прив'язка до рахунків, розпізнавання чеків	Можливість перегляду через комп'ютер, детальна статистика	Велика кількість реклами в безкоштовній версії	4.6
Spendee	iOS, Android	Безкоштовно, є платна версія	Об'єднання грошових потоків, категорії, інтуїтивний функціонал	Зручний інтерфейс, вбудовані підказки	Обмежений функціонал у безкоштовній версії	4.4
Bluecoins	Android	Платний	Облік витрат/доходів, бюджетування, звіти, для сім'ї та бізнесу	Зрозумілий інтерфейс, продумана система звітності	Відсутність безкоштовної версії	4.7
Monobudget	Android	Умовно-безкоштовний	Встановлення бюджету за категоріями, підключення рахунків Monobank, сімейний обмін	Інтеграція з Monobank, персональний фінансовий радник	Обмежена функціональність без платної підписки	4.3
Wallet	iOS, Android	Безкоштовно, є платна версія	Облік витрат, бюджетування, банківська синхронізація, планування регулярних платежів	Сумісність зі 156 банками, автоматична синхронізація, веб-версія	Можливі проблеми з конфіденційністю при синхронізації з банками	4.6
Money Lover	iOS, Android	Безкоштовно, є платна версія	Облік витрат/доходів, бюджетування, борги, заощадження, звіти, конвертер валют	Аналіз тенденцій доходів/витрат, багато корисних сервісів	Деякі функції доступні лише в платній версії	4.7

ВИСНОВКИ ДО РОЗДІЛУ 1

В результаті аналізу були виявлені основні функції, які вони пропонують, їхні переваги та недоліки.

Також було визначено, що планування особистого бюджету є важливим інструментом для контролю витрат, заощадження коштів, розуміння фінансового становища, впевненості у завтрашньому дні та досягнення фінансових цілей

У першому розділі було проаналізовано існуючі мобільні додатки для планування особистого бюджету.

2. Аналіз вимог до додатку для визначення потреб користувачів та характеристик системи.

2.1. Вимоги користувачів

Вимоги користувачів до додатку для планування особистого бюджету включають зручність використання, можливість додавання та відстеження доходів і витрат, створення звітів, налаштування бюджетів, а також забезпечення безпеки та конфіденційності даних [2]. Отже, система повинна відповідати наступним вимогам:

- полегшити ведення власного бюджету;
- надати можливість аналізу даних по доходам та витратам;
- автоматизувати процес підрахунків результатів;
- забезпечення безпеки та конфіденційності даних.

2.2. Функціональні вимоги та нефункціональні вимоги

Функціональні вимоги описують основні можливості, які повинен надавати мобільний додаток для планування особистого бюджету.

Облік доходів та витрат. Користувач повинен мати можливість додавати нові джерела доходу, вказуючи: суму доходу (обов'язково), дату отримання доходу (за замовчуванням - поточна дата, з можливістю вибору), категорію доходу (наприклад, зарплата, підробіток, інвестиції, подарунки) з можливістю вибору з попередньо визначеного списку або створення нової категорії, опис доходу (необов'язково), рахунок/гаманець, на який надійшов дохід (з можливістю вибору з попередньо створених).

Користувач повинен мати можливість додавати нові витрати, вказуючи, суму витрати (обов'язково), дату здійснення витрати (за замовчуванням - поточна дата, з можливістю вибору), категорію витрати

(наприклад, продукти, транспорт, комунальні послуги, розваги) з можливістю вибору з попередньо визначеного списку або створення нової категорії, опис витрати (необов'язково), рахунок/гаманець, з якого було здійснено витрату (з можливістю вибору з попередньо створених), місце здійснення витрати (необов'язково), можливість додавання фото чека або іншого підтвердження витрати (необов'язково).

Редагування та видалення записів. Користувач повинен мати можливість переглядати, редагувати та видаляти раніше додані записи про доходи та витрати. При редагуванні повинні зберігатися всі поля запису. При видаленні повинно бути підтвердження дії.

Керування категоріями

Користувач повинен мати можливість переглядати списки існуючих категорій доходів та витрат, а також мати можливість додавати нові категорії доходів та витрат з унікальними назвами тазмінювати назви існуючих категорій. Користувач повинен мати можливість видаляти порожні категорії. Якщо категорія містить записи, система повинна запропонувати користувачеві перенести ці записи до іншої існуючої категорії або скасувати видалення.

Керування рахунками/гаманцями

Користувач повинен мати можливість додавати нові рахунки (наприклад, банківські картки, готівка, електронні гаманці), вказуючи назву рахунку (обов'язково), початковий баланс (необов'язково), валюту рахунку (з можливістю вибору з попередньо визначеного списку).

Користувач повинен мати можливість редагувати назву та початковий баланс рахунку та можливість переглядати поточний баланс кожного створеного рахунку.

Користувач повинен мати можливість видаляти рахунки, які не містять транзакцій. Якщо рахунок містить транзакції, система повинна

запропонувати користувачеві перенести ці транзакції до іншого існуючого рахунку або скасувати видалення.

Бюджетування

Користувач повинен мати можливість створювати бюджети для певних категорій витрат на певний період (наприклад, місяць). При створенні бюджету користувач повинен вказати категорію витрат, суму бюджету, період дії бюджету (наприклад, поточний місяць, наступний місяць, інший період). Користувач повинен мати можливість переглядати поточний статус виконання кожного активного бюджету (залишок до ліміту або перевищення). Користувач повинен мати можливість редагувати суму та період дії існуючих бюджетів та мати можливість видаляти створені бюджети.

Звіти та аналітика

Користувач повинен мати можливість генерувати звіти про доходи та витрати за обраний період (наприклад, день, тиждень, місяць, рік, довільний діапазон дат). Звіти повинні включати загальну суму доходів, загальну суму витрат, різницю між доходами та витратами (сальдо), деталізацію доходів та витрат за категоріями з відображенням сум.

Візуалізація даних

Доцільно мати можливість представлення фінансових даних у вигляді графіків та діаграм (наприклад, кругові діаграми для розподілу витрат за категоріями, лінійні графіки для динаміки доходів та витрат) та можливість налаштування відображення даних (наприклад, вибір періоду, категорій).

Безпека та авторизація

Реєстрація та авторизація:

- Користувач повинен мати можливість зареєструватися в додатку (наприклад, за допомогою електронної пошти та пароля).
- Користувач повинен мати можливість авторизуватися в додатку для доступу до своїх даних.
- Можливість відновлення пароля.

Захист даних:

- Забезпечення безпечного зберігання фінансових даних користувача.
- Можливість використання біометричної автентифікації (відбиток пальця, розпізнавання обличчя) для швидкого доступу до додатку (за наявності відповідної функції на пристрої) **[3]:** .

Інші функціональні можливості

Користувач повинен мати можливість синхронізації даних користувача між різними пристроями (за умови використання одного облікового запису) та можливість розпізнавання чеків через інтеграцію з OCR (Optical Character Recognition) для автоматичного розпізнавання даних з фотографій чеків.

Користувач повинен мати можливість додавати заплановані доходи та витрати для прогнозування фінансового стану та автоматично імпортувати транзакції з банківських рахунків (за бажанням та за наявності відповідних API) [4]. Також користувач повинен мати можливість відстежувати свої борги та кредити, встановлювати графіки платежів та отримувати нагадування.

Нефункціональні вимоги

До нефункціональних вимог до додатку для планування особистого бюджету відносяться:

- продуктивність;
- надійність;
- безпека;
- зручність використання (Usability);
- супроводжуваність (Maintainability);
- переносимість (Portability) - кросплатформність та незалежність від конкретного обладнання.

ВИСНОВКИ ДО РОЗДІЛУ 2

У другому розділі було проведено аналіз вимог до мобільного додатку для планування особистого бюджету.

В результаті аналізу були визначені вимоги користувачів до додатку, які включають зручність використання, можливість додавання та відстеження доходів і витрат, створення звітів, налаштування бюджетів, а також забезпечення безпеки та конфіденційності даних.

Також були детально описані функціональні вимоги до додатку, такі як облік доходів та витрат, редагування та видалення записів, керування категоріями та рахунками/гаманцями, бюджетування, звіти та аналітика, візуалізація даних, безпека та авторизація, та інші функціональні можливості.

Крім того, були визначені нефункціональні вимоги до додатку, такі як продуктивність, надійність, безпека, зручність використання, супроводжуваність та переносимість.

2. Аналіз вимог до додатку для визначення потреб користувачів та характеристик системи.

2.1. Вимоги користувачів

Вимоги користувачів до додатку для планування особистого бюджету включають зручність використання, можливість додавання та відстеження доходів і витрат, створення звітів, налаштування бюджетів, а також забезпечення безпеки та конфіденційності даних [2]. Отже, система повинна відповідати наступним вимогам:

- полегшити ведення власного бюджету;
- надати можливість аналізу даних по доходам та витратам;
- автоматизувати процес підрахунків результатів;
- забезпечення безпеки та конфіденційності даних.

2.2. Функціональні вимоги та нефункціональні вимоги

Функціональні вимоги описують основні можливості, які повинен надавати мобільний додаток для планування особистого бюджету.

Облік доходів та витрат. Користувач повинен мати можливість додавати нові джерела доходу, вказуючи: суму доходу (обов'язково), дату отримання доходу (за замовчуванням - поточна дата, з можливістю вибору), категорію доходу (наприклад, зарплата, підробіток, інвестиції, подарунки) з можливістю вибору з попередньо визначеного списку або створення нової категорії, опис доходу (необов'язково), рахунок/гаманець, на який надійшов дохід (з можливістю вибору з попередньо створених).

Користувач повинен мати можливість додавати нові витрати, вказуючи, суму витрати (обов'язково), дату здійснення витрати (за замовчуванням - поточна дата, з можливістю вибору), категорію витрати

(наприклад, продукти, транспорт, комунальні послуги, розваги) з можливістю вибору з попередньо визначеного списку або створення нової категорії, опис витрати (необов'язково), рахунок/гаманець, з якого було здійснено витрату (з можливістю вибору з попередньо створених), місце здійснення витрати (необов'язково), можливість додавання фото чека або іншого підтвердження витрати (необов'язково).

Редагування та видалення записів. Користувач повинен мати можливість переглядати, редагувати та видаляти раніше додані записи про доходи та витрати. При редагуванні повинні зберігатися всі поля запису. При видаленні повинно бути підтвердження дії.

Керування категоріями

Користувач повинен мати можливість переглядати списки існуючих категорій доходів та витрат, а також мати можливість додавати нові категорії доходів та витрат з унікальними назвами тазмінювати назви існуючих категорій. Користувач повинен мати можливість видаляти порожні категорії. Якщо категорія містить записи, система повинна запропонувати користувачеві перенести ці записи до іншої існуючої категорії або скасувати видалення.

Керування рахунками/гаманцями

Користувач повинен мати можливість додавати нові рахунки (наприклад, банківські картки, готівка, електронні гаманці), вказуючи назву рахунку (обов'язково), початковий баланс (необов'язково), валюту рахунку (з можливістю вибору з попередньо визначеного списку).

Користувач повинен мати можливість редагувати назву та початковий баланс рахунку та можливість переглядати поточний баланс кожного створеного рахунку.

Користувач повинен мати можливість видаляти рахунки, які не містять транзакцій. Якщо рахунок містить транзакції, система повинна

запропонувати користувачеві перенести ці транзакції до іншого існуючого рахунку або скасувати видалення.

Бюджетування

Користувач повинен мати можливість створювати бюджети для певних категорій витрат на певний період (наприклад, місяць). При створенні бюджету користувач повинен вказати категорію витрат, суму бюджету, період дії бюджету (наприклад, поточний місяць, наступний місяць, інший період). Користувач повинен мати можливість переглядати поточний статус виконання кожного активного бюджету (залишок до ліміту або перевищення). Користувач повинен мати можливість редагувати суму та період дії існуючих бюджетів та мати можливість видаляти створені бюджети.

Звіти та аналітика

Користувач повинен мати можливість генерувати звіти про доходи та витрати за обраний період (наприклад, день, тиждень, місяць, рік, довільний діапазон дат). Звіти повинні включати загальну суму доходів, загальну суму витрат, різницю між доходами та витратами (сальдо), деталізацію доходів та витрат за категоріями з відображенням сум.

Візуалізація даних

Доцільно мати можливість представлення фінансових даних у вигляді графіків та діаграм (наприклад, кругові діаграми для розподілу витрат за категоріями, лінійні графіки для динаміки доходів та витрат) та можливість налаштування відображення даних (наприклад, вибір періоду, категорій).

Безпека та авторизація

Реєстрація та авторизація:

- Користувач повинен мати можливість зареєструватися в додатку (наприклад, за допомогою електронної пошти та пароля).
- Користувач повинен мати можливість авторизуватися в додатку для доступу до своїх даних.
- Можливість відновлення пароля.

Захист даних:

- Забезпечення безпечного зберігання фінансових даних користувача.
- Можливість використання біометричної автентифікації (відбиток пальця, розпізнавання обличчя) для швидкого доступу до додатку (за наявності відповідної функції на пристрої) **[3]:**

Інші функціональні можливості

Користувач повинен мати можливість синхронізації даних користувача між різними пристроями (за умови використання одного облікового запису) та можливість розпізнавання чеків через інтеграцію з OCR (Optical Character Recognition) для автоматичного розпізнавання даних з фотографій чеків.

Користувач повинен мати можливість додавати заплановані доходи та витрати для прогнозування фінансового стану та автоматично імпортувати транзакції з банківських рахунків (за бажанням та за наявності відповідних API) [4]. Також користувач повинен мати можливість відстежувати свої борги та кредити, встановлювати графіки платежів та отримувати нагадування.

Нефункціональні вимоги

До нефункціональних вимог до додатку для планування особистого бюджету відносяться:

- продуктивність;
- надійність;
- безпека;
- зручність використання (Usability);
- супроводжуваність (Maintainability);
- переносимість (Portability) - кросплатформність та незалежність від конкретного обладнання.

ВИСНОВКИ ДО РОЗДІЛУ 2

У другому розділі було проведено аналіз вимог до мобільного додатку для планування особистого бюджету. В результаті аналізу були визначені вимоги користувачів до додатку, які включають зручність використання, можливість додавання та відстеження доходів і витрат, створення звітів, налаштування бюджетів, а також забезпечення безпеки та конфіденційності даних. Також були детально описані функціональні вимоги до додатку, такі як облік доходів та витрат, редагування та видалення записів, керування категоріями та рахунками/гаманцями, бюджетування, звіти та аналітика, візуалізація даних, безпека та авторизація, та інші функціональні можливості. Крім того, були визначені нефункціональні вимоги до додатку, такі як продуктивність, надійність, безпека, зручність використання, супроводжуваність та переносимість.

3. Проектування мобільного додатку для планування особистого бюджету

3.1. Архітектура додатку

Автор запропонував модульну архітектуру мобільного додатку для планування особистого бюджету, яка є гнучкою та масштабованою.

Основними компонентами застосунку є клієнтська частина (Frontend), серверна частина (Backend), база даних та API (Application Programming Interface).

Клієнтська частина (Frontend)

Для розробки клієнтської частини (Frontend) були вибрані технології React, Vue.js, Angular (або нативний JavaScript/HTML/CSS для простішого варіанту).

Функціональні вимоги до додатку: повинен бути інтуїтивно зрозумілий інтерфейс для введення доходів та витра, відображення балансу, історії транзакцій та звітів у вигляді графіків та таблиць, можливість створення та редагування категорій доходів та витрат, функціонал для встановлення бюджетів на певні категорії, система сповіщень про перевищення бюджету або важливі події, аутентифікація та авторизація користувачів, підтримка різних валют та можливість експорту даних (наприклад, у CSV) [5].

Серверна частина (Backend)

Для розробки серверної частини буули вибрані технології: Python (Django/Flask), Node.js (Express), Java (Spring), Ruby on Rails. Функціональні вимоги до Backend**:** обробка запитів від клієнтської частини, зберігання та управління даними користувачів, транзакцій, категорій, бюджетів, реалізація бізнес-логіки (наприклад, розрахунок балансу, формування звітів), аутентифікація та авторизація користувачів (JWT, OAuth2), валідація даних, забезпечення безпеки даних, інтеграція з іншими сервісами [6].

База даних

Для озробки бази даних було вибрано ехнології MySQL, SQLite [7].

Структура даних:

Користувачі: user_id, username, password_hash, email, registration_date тощо.

Категорії доходів: category_id, user_id, name.

Категорії витрат: category_id, user_id, name.

Транзакції: transaction_id, user_id, type (дохід/витрата), category_id, amount, currency, date, description.

Бюджети: budget_id, user_id, category_id, period (місяць, рік), amount.

API (Application Programming Interface)

Тип: RESTful API.

Функціональність:

Набір ендпоінтів для взаємодії між клієнтською та серверною частинами. Приклади:

- /api/users/register (реєстрація користувача)
- /api/users/login (авторизація користувача)
- /api/transactions (отримання, додавання, редагування, видалення транзакцій)
- /api/categories/income (отримання, додавання, редагування, видалення категорій доходів)
- /api/categories/expense (отримання, додавання, редагування, видалення категорій витрат)

- /api/budgets (отримання, додавання, редагування, видалення бюджетів)
- /api/reports/balance (отримання балансу)
- /api/reports/history (отримання історії транзакцій)
- /api/reports/summary (отримання зведених звітів)

Додаткові можливі компоненти:

Модуль аналітики: Для більш глибокого аналізу фінансових даних, прогнозування та надання персоналізованих рекомендацій.

Модуль імпорту даних: Для автоматичного або ручного імпорту транзакцій з банківських виписок або інших джерел.

Система сповіщень: Для налаштування різних типів сповіщень (наприклад, про перевищення лімітів, майбутні платежі).

Мобільні додатки (Android/iOS): Для доступу до функціоналу застосунку на мобільних пристроях.

Принципи проектування:

Модульність: Кожен компонент є незалежним і виконує свою чітко визначену функцію. Це полегшує розробку, тестування та масштабування.

Розділення відповідальності: Кожен шар архітектури (клієнт, сервер, база даних) відповідає за свою частину функціоналу.

Масштабованість: Архітектура повинна дозволяти збільшувати потужність окремих компонентів за потреби (наприклад, збільшення кількості серверів або оптимізація бази даних).

Безпека: Забезпечення захисту даних користувачів на всіх рівнях (аутентифікація, авторизація, шифрування даних).

Зручність використання (UX): Клієнтська частина повинна бути інтуїтивно зрозумілою та приємною у використанні.

Приблизний флоу роботи:

1. Користувач взаємодіє з клієнтською частиною (наприклад, вводить нову витрату).
2. Клієнтська частина відправляє запит до серверної частини через API.
3. Серверна частина обробляє запит, валідує дані та взаємодіє з базою даних для збереження або отримання інформації.
4. Серверна частина повертає відповідь клієнтській частині.
5. Клієнтська частина відображає оновлені дані користувачеві.

Автор запропонував модульну архітектуру мобільного додатку для планування особистого бюджету, яка є гнучкою та масштабованою.

Основними компонентами застосунку є клієнтська частина (Frontend), серверна частина (Backend), база даних та API (Application Programming Interface).

Клієнтська частина (Frontend)

Для розробки клієнтської частини (Frontend) були вибрані технології React, Vue.js, Angular (або нативний JavaScript/HTML/CSS для простішого варіанту).

Функціональні вимоги до додатку: повинен бути інтуїтивно зрозумілий інтерфейс для введення доходів та витра, відображення балансу, історії транзакцій та звітів у вигляді графіків та таблиць, можливість створення та редагування категорій доходів та витрат, функціонал для встановлення бюджетів на певні категорії, система сповіщень про перевищення бюджету або важливі події, аутентифікація та

авторизація користувачів, підтримка різних валют та можливість експорту даних (наприклад, у CSV).

Серверна частина (Backend)

Для розробки серверної частини буули вибрані технології: Python (Django/Flask), Node.js (Express), Java (Spring), Ruby on Rails. Функціональні вимоги до Backend [8] **:** обробка запитів від клієнтської частини, зберігання та управління даними користувачів, транзакцій, категорій, бюджетів, реалізація бізнес-логіки (наприклад, розрахунок балансу, формування звітів), аутентифікація та авторизація користувачів (JWT, OAuth2), валідація даних, забезпечення безпеки даних, інтеграція з іншими сервісами.

База даних

Для озробки бази даних було вибрано ехнології MySQL, SQLite.

Структура даних:

Користувачі: user_id, username, password_hash, email, registration_date тощо.

Категорії доходів: category_id, user_id, name.

Категорії витрат: category_id, user_id, name.

Транзакції: transaction_id, user_id, type (дохід/витрата), category_id, amount, currency, date, description.

Бюджети: budget_id, user_id, category_id, period (місяць, рік), amount.

API (Application Programming Interface)

Тип: RESTful API.

Функціональність:

Набір ендпоінтів для взаємодії між клієнтською та серверною частинами. Приклади:

- /api/users/register (реєстрація користувача)

- /api/users/login (авторизація користувача)
- /api/transactions (отримання, додавання, редагування, видалення транзакцій)
- /api/categories/income (отримання, додавання, редагування, видалення категорій доходів)
- /api/categories/expense (отримання, додавання, редагування, видалення категорій витрат)
- /api/budgets (отримання, додавання, редагування, видалення бюджетів)
- /api/reports/balance (отримання балансу)
- /api/reports/history (отримання історії транзакцій)
- /api/reports/summary (отримання зведених звітів)

Додаткові можливі компоненти [9]:

Модуль аналітики: Для більш глибокого аналізу фінансових даних, прогнозування та надання персоналізованих рекомендацій.

Модуль імпорту даних: Для автоматичного або ручного імпорту транзакцій з банківських виписок або інших джерел.

Система сповіщень: Для налаштування різних типів сповіщень (наприклад, про перевищення лімітів, майбутні платежі).

Мобільні додатки (Android/iOS): Для доступу до функціоналу застосунку на мобільних пристроях.

Принципи проектування:

Модульність: Кожен компонент є незалежним і виконує свою чітко визначену функцію. Це полегшує розробку, тестування та масштабування.

Розділення відповідальності: Кожен шар архітектури (клієнт, сервер, база даних) відповідає за свою частину функціоналу.

Масштабованість: Архітектура повинна дозволяти збільшувати потужність окремих компонентів за потреби (наприклад, збільшення кількості серверів або оптимізація бази даних).

Безпека: Забезпечення захисту даних користувачів на всіх рівнях (аутентифікація, авторизація, шифрування даних).

Зручність використання (UX): Клієнтська частина повинна бути інтуїтивно зрозумілою та приємною у використанні.

Приблизний флоу роботи:

1. Користувач взаємодіє з клієнтською частиною (наприклад, вводить нову витрату).
2. Клієнтська частина відправляє запит до серверної частини через API.
3. Серверна частина обробляє запит, валідує дані та взаємодіє з базою даних для збереження або отримання інформації.
4. Серверна частина повертає відповідь клієнтській частині.
5. Клієнтська частина відображає оновлені дані користувачеві.

3.2. Опис модулів та їх взаємодії

1. Модуль аутентифікації та авторизації відповідає за реєстрацію нових користувачів, їхню автентифікацію (перевірку логіну та паролю) та авторизацію (визначення прав доступу до різних функцій застосунку). Виконує функції:

- Реєстрація користувачів (прийом даних, валідація, збереження в базі даних).

- Логін користувачів (перевірка облікових даних, генерація токенів сесії).
- Вихід з облікового запису (інвалідація токенів сесії).
- Керування сесіями користувачів.
- Захист від несанкціонованого доступу.

Клієнтська частина надсилає запити на реєстрацію, логін, вихід. Отримує токени сесії для подальших запитів. Інші модулі (серверна частина) перевіряє валідність токенів сесії для підтвердження автентифікації та авторизації користувача перед виконанням дій.

2. Модуль керування транзакціями дозволяє користувачам додавати, переглядати, редагувати та видаляти інформацію про свої доходи та витрати. Модуль виконує функції:

- Додавання нових транзакцій (прийом даних, валідація, збереження в базі даних).
- Перегляд списку транзакцій (фільтрація за датою, категорією, типом тощо).
- Редагування існуючих транзакцій.
- Видалення транзакцій.
- Відображення детальної інформації про окрему транзакцію.

Клієнтська частина надає інтерфейс для введення та перегляду транзакцій, надсилає відповідні запити до серверної частини [10]. Модуль

категорій використовує інформацію про категорії для прив'язки транзакцій до певних груп. Модуль звітів отримує дані про транзакції для формування фінансових звітів. База даних зберігає та отримує інформацію про транзакції.

3. Модуль керування категоріями:

Модулькерування категоріями дозволяє користувачам створювати, переглядати, редагувати та видаляти власні категорії доходів та витрат для кращої організації фінансів та виконує функції:

- Створення нових категорій (прийом назви, типу (дохід/витрата), збереження в базі даних).
- Перегляд списку категорій.
- Редагування назв існуючих категорій.
- Видалення категорій (з попередженням, якщо є пов'язані транзакції).

Клієнтська частина надає інтерфейс для керування категоріями, надсилає відповідні запити до серверної частини. Модуль керування транзакціями використовує список доступних категорій при додаванні або редагуванні транзакцій. База даних зберігає та отримує інформацію про категорії.

4. Модуль бюджетування дозволяє користувачам встановлювати фінансові ліміти на певні категорії витрат на певний період часу (наприклад, місяць). Модуль фиконує функції:

- Створення нових бюджетів (вибір категорії, встановлення суми ліміту, вибір періоду).

- Перегляд встановлених бюджетів.
- Редагування існуючих бюджетів.
- Видалення бюджетів.
- Порівняння фактичних витрат з встановленими бюджетами.

Клієнтська частина надає інтерфейс для керування бюджетами, відображає інформацію про стан виконання бюджетів. Модуль керування транзакціями отримує інформацію про витрати для порівняння з встановленими бюджетами. Модуль сповіщень може надсилати сповіщення про наближення або перевищення бюджету. База даних зберігає та отримує інформацію про бюджети.

5. Модуль звітів генерує різні фінансові звіти на основі даних про транзакції та бюджети, надаючи користувачам наочне уявлення про їхній фінансовий стан. Модуль звітів виконує наступні функції:

- Генерація звітів про доходи та витрати за певний період.
- Відображення балансу.
- Візуалізація даних у вигляді графіків та діаграм (наприклад, кругова діаграма витрат за категоріями).
- Порівняння доходів та витрат.
- Звіти про виконання бюджетів.

Клієнтська частина надсилає запити на генерацію звітів, відображає отримані дані. Модуль керування транзакціями отримує дані про

транзакції для формування звітів. Модуль бюджетування отримує інформацію про встановлені бюджети для звітів про їх виконання.

6. Модуль сповіщень відповідає за надсилання користувачам сповіщень про важливі події, такі як перевищення бюджету, майбутні платежі (якщо реалізовано) тощо. Функції модуля сповіщень:

- Налаштування правил для сповіщень.
- Генерація та надсилання сповіщень (наприклад, push-сповіщення, email).

Модуль бюджетування інформує про наближення або перевищення бюджету. Модуль керування транзакціями інформує про заплановані транзакції. Клієнтська частина відображає отримані сповіщення.

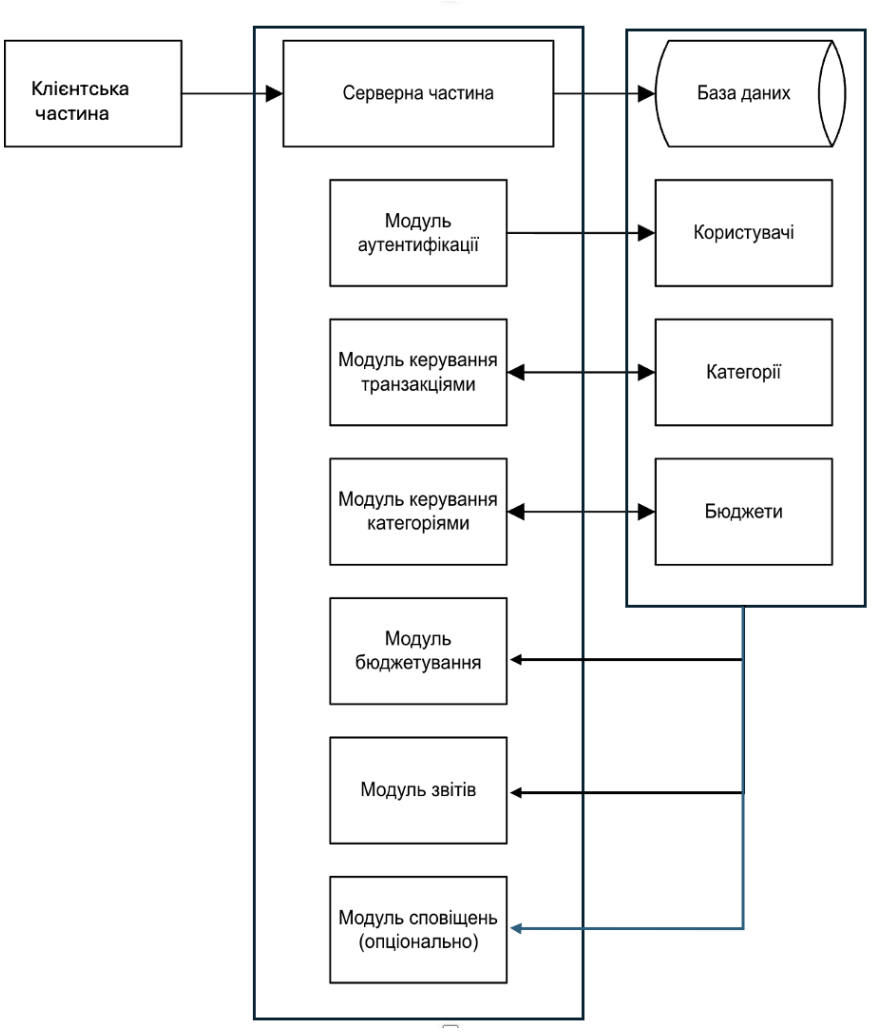

Рисунок 3.1. Схема взаємодії між модулями

Ця схема показує основні напрямки обміну даними між модулями. Наприклад, модуль керування транзакціями використовує модуль категорій для визначення типу транзакції та модуль аутентифікації для ідентифікації користувача, якому належить транзакція. Модуль звітів отримує дані з модуля керування транзакціями та модуля бюджетування для формування звітів. Такий модульний підхід робить мобільний додаток більш гнучким, легким у підтримці та розширенні. Кожен модуль може розроблятися та тестуватися незалежно, що спрощує процес розробки.

3.3. Вибір технологій та інструментів

Для розробки мобільних додатків, зокрема додатків для обліку домашнього бюджету, використовуються різноманітні сучасні інформаційні технології та інструменти. Ось деякі з них:

1. Мови програмування:

Java та Kotlin: Це основні мови для розробки на платформі Android. Kotlin стає все більш популярним завдяки своїй сучасності та безпеці.

Swift та Objective-C: Використовуються для розробки додатків на платформі iOS. Swift є новою та рекомендованою мовою Apple.

JavaScript та TypeScript: Застосовуються у кросплатформних фреймворках, таких як React Native, для розробки додатків, що працюють як на Android, так і на iOS.

Dart: Основна мова програмування для Flutter, ще одного популярного кросплатформного фреймворку.

C#: Може використовуватись у Xamarin для кросплатформної розробки, а також у .NET Framework.

2. Фреймворки та бібліотеки:

React Native: Розроблений Facebook, дозволяє будувати мобільні додатки за допомогою JavaScript та React.

Flutter: Створений Google, забезпечує швидку розробку кросплатформних додатків з гарним інтерфейсом.

Angular: Фреймворк від Google для створення веб- та мобільних додатків.

Vue.js: Прогресивний JavaScript-фреймворк для розробки інтерфейсів користувача.

.NET Framework та Xamarin: Платформи Microsoft для розробки додатків на різних платформах, включаючи мобільні.

3. Бази даних:

SQLite: Локальна база даних, часто використовується для зберігання даних безпосередньо на мобільному пристрої.

Realm: Мобільна база даних, яка пропонує альтернативу SQLite з покращеною продуктивністю та зручнішим API.

Firebase Realtime Database та Cloud Firestore: Хмарні бази даних від Google, що забезпечують синхронізацію даних між пристроями.

MySQL та PostgreSQL: Потужні реляційні бази даних, які можуть використовуватись для серверної частини додатку.

4. Інструменти розробки:

Android Studio: Офіційне інтегроване середовище розробки (IDE) для Android.

Xcode: IDE від Apple для розробки додатків під iOS.

Visual Studio Code: Популярний редактор коду з підтримкою багатьох мов та фреймворків.

Git: Система контролю версій, що дозволяє розробникам спільно працювати над кодом та відстежувати зміни.

5. Додаткові технології та інструменти:

API (Application Programming Interface): Набір протоколів та інструментів, що дозволяють різним програмам взаємодіяти між собою. У мобільних додатках часто використовуються RESTful API для обміну даними між клієнтською та серверною частинами.

OCR (Optical Character Recognition): Технологія оптичного розпізнавання символів, яка може використовуватись для автоматичного зчитування даних з чеків.

Cloud Services (AWS, Google Cloud, Azure): Хмарні платформи, що надають різноманітні сервіси для розробки, розгортання та підтримки мобільних додатків, такі як зберігання даних, обчислювальні ресурси та інструменти аналітики.

Інструменти для тестування та налагодження: Різноманітні емулятори, симулятори та інструменти для автоматизованого тестування, що допомагають забезпечити якість мобільних додатків.

Вибір автора:

Для розробки клієнтської частини (Frontend) були обрані технології React, Vue.js, Angular або нативний JavaScript/HTML/CSS.

Для розробки серверної частини (Backend) були обрані технології: Python (Django/Flask), Node.js (Express), Java (Spring), Ruby on Rails.

Для розробки бази даних були обрані технології MySQL, SQLite.

Для розробки API (Application Programming Interface) був обраний тип RESTful API.

Також, як середовище розробки було обрано .NET Framework, мову програмування C#, а системою керування базою даних – SQLite.

ВИСНОВКИ ДО РОЗДІЛУ 3

У третьому розділі було спроектовано мобільний додаток для планування особистого бюджету.

Розроблено модульну архітектуру додатку, що включає клієнтську частину (Frontend), серверна частина (Backend), базу даних та API (Application Programming Interface).

Для кожного компонента архітектури були обрані конкретні технології та описані їхні функціональні вимоги.

Також було детально описано модулі додатку та їх взаємодія, включаючи модуль аутентифікації та авторизації, модуль керування транзакціями, модуль керування категоріями, модуль бюджетування, модуль звітів та модуль сповіщень.

Наведено схему взаємодії між модулями, яка показує основні напрямки обміну даними між ними.

Крім того, було обґрунтовано вибір технологій та інструментів для розробки додатку, таких як .NET Framework, C# та SQLite.

4. Реалізація і тестування мобільного додатка

4.1. Інтерфейс користувача та опис роботи додатку

До основних компонентів додатку відносяться вікна «Головне вікно», «Вікно доходів», «Вікно витрат», «Майстер налаштувань».

Для забезпечення обробки різних типів помилок (інформаційних, попереджувальних, критичних) була спроектована ієрархія екранних форм. Графічний інтерфейс розроблено з використанням наступних елементів:

- Label – для надписів;
- ComboBox – для відображення випадаючого списку;
- Button – задля взаємодії з користувачем;
- TextField – для введення інформації користувачем;
- DataGridView – для відображення таблиць;
- DatePicker – для вибору дати;
- Chart – для відображення графіків.

Базу даних представлено на рис. 4.1 та 4.2.

Рисунок 4.1 — Таблиця «Targets»

	Имя	Тип данных	Первичный ключ	Внешний ключ	Уникальность	Проверка	Не NULL	Сравнение	Generated	
1	Id	INTEGER								NULL
2	Name	STRING								NULL
3	OperationType	STRING								NULL
4	FinanceOperationType	INT								0
5	Cost	INTEGER								NULL
6	Date	DATE								NULL

Рисунок 4.2 — Таблиця «FinanceOperations»

За необхідністю, можливе додавання нових полів та оновлення бази даних. Модель бази даних було наведено на рис. 2.3.

Додаток для планування власного бюджету – десктопний додаток, який запускається наступним чином: відкрити програму за допомогою відповідного exe-файлу. Після запуску, користувач обирає між доступними функціями, що були зазначені у пункті 2.3 Алгоритм роботи програмного продукту. Програмний код наведено у додатках А-В.

Після завантаження програмного додатку стають доступними наступні екранні форми:

- Якщо програмний продукт запущено вперше – відкривається вікно привітання майстра налаштувань (див. рис. 4.3).

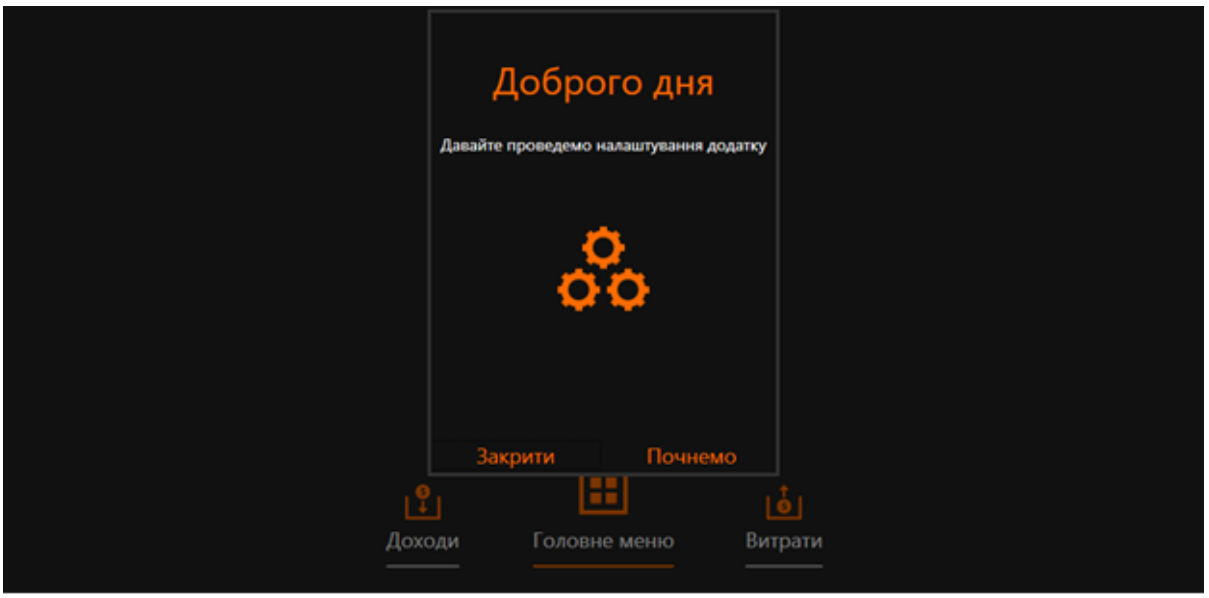

Рисунок 4.3 — Екранна форма «Привітання майстера налаштувань»

- Екранна форма «Визначення доходу» для утворення можливої основи (див. рис. 4.4).

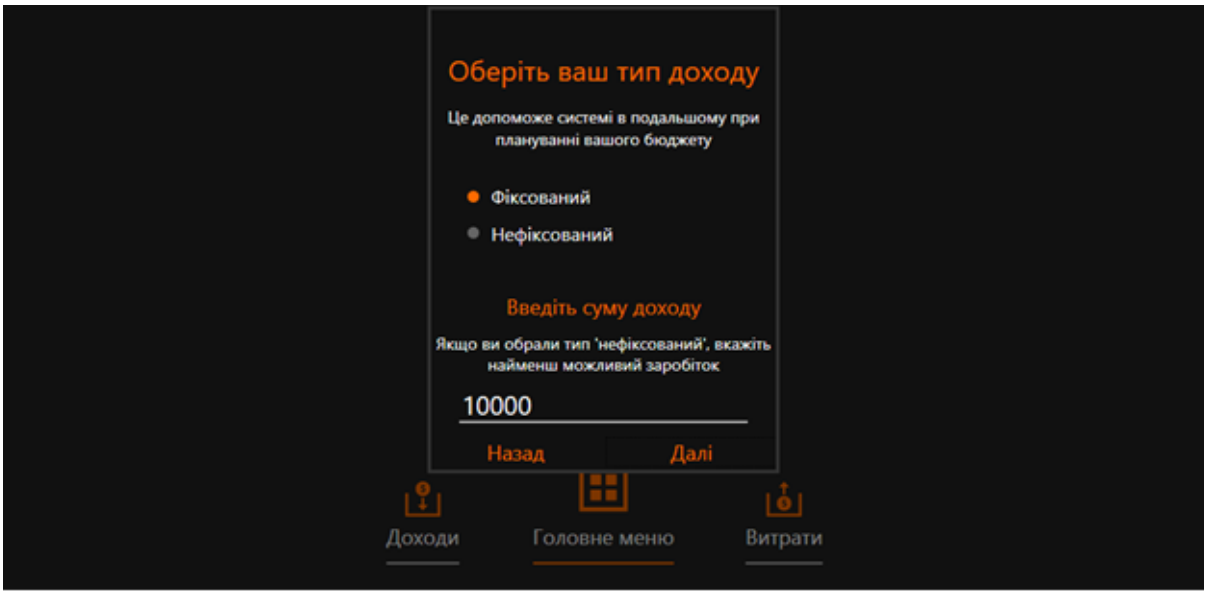

Рисунок 4.4 — Екранна форма «Визначення доходу»

- Екранна форма «Визначення витрат на день» відображає форму, завдяки якій встановлюється середні витрати за день (див. рис. 4.5).

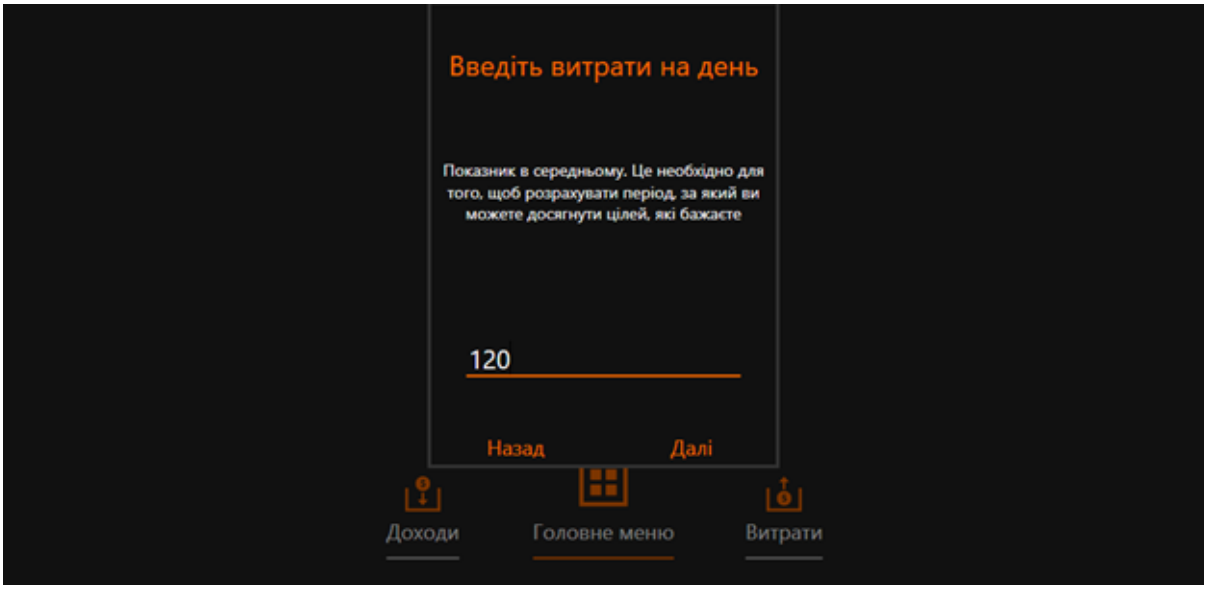

Рисунок 4.5 — Екранна форма «Визначення витрат на день»

- Екранна форма «Подушка фінансової безпеки» для визначення, чи необхідно людині створити фінансову подушку безпеки (див. рис. 4.6).

Рисунок 4.6 — Екранна форма «Подушка фінансової безпеки»

- Екранна форма «Створення нової цілі» дозволяє створити першу ціль (див. рис. 4.7).

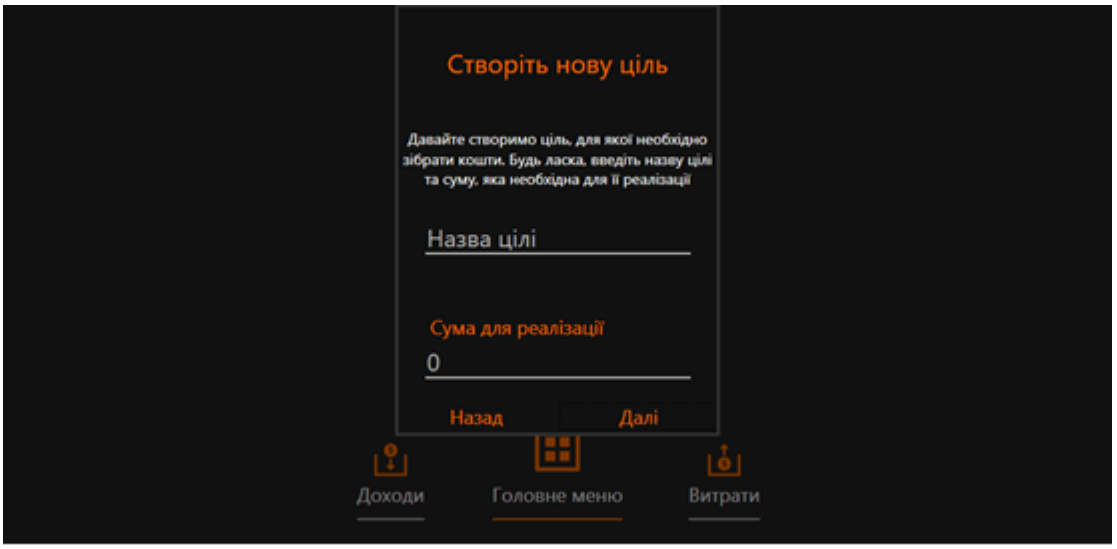

Рисунок 4.7 — Екранна форма «Прихід товару»

- Екранна форма «Завершення налаштування» (див. рис. 4.8).

Рисунок 4.8 — Екранна форма «Завершення налаштування»

- Екранна форма «Головне вікно» показує основну інформацію про доходи по місяцям, витратам по місяцям, досягненні цілі та подушки безпеки (якщо необхідна) (див. рис. 4.9).

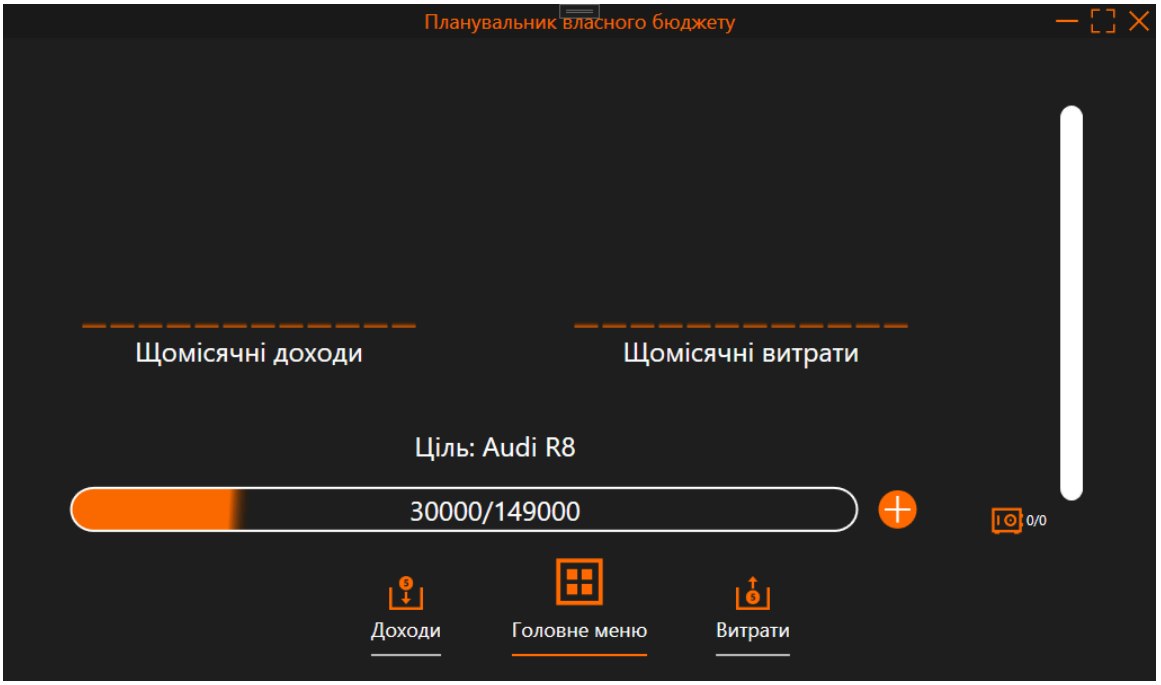

Рисунок 4.9 — Екранна форма «Головне вікно»

- Екранна форма «Доходи» відображає дані про доходи за день, а також має поля для створення нового доходу за сьогодні (див. рис. 4.10).

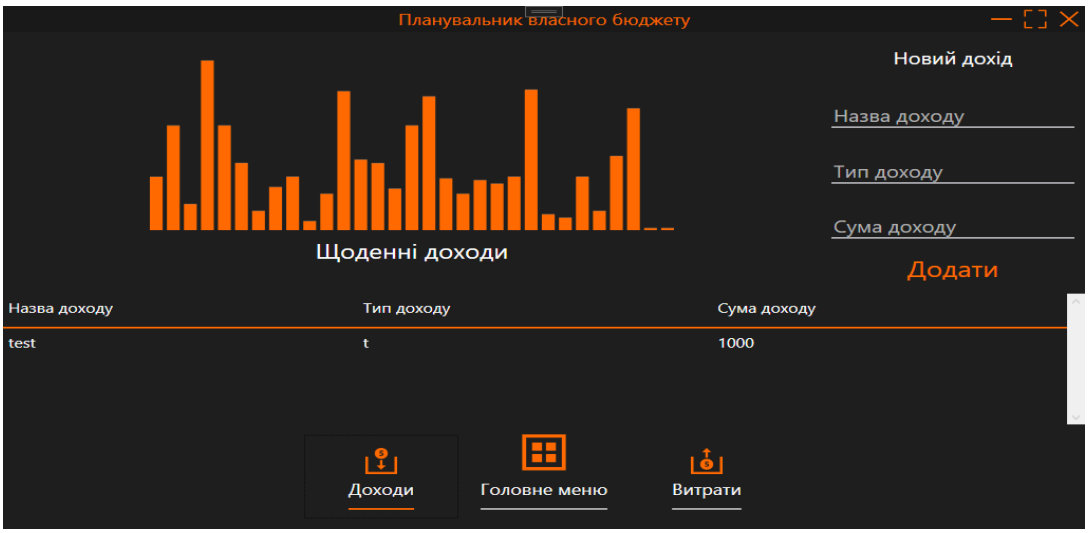

Рисунок 4.10 — Екранна форма «Доходи»

- Екранна форма «Витрати» відображає дані про доходи за день, а також має поля для створення нової витрати за сьогодні (див. рис. 4.11).

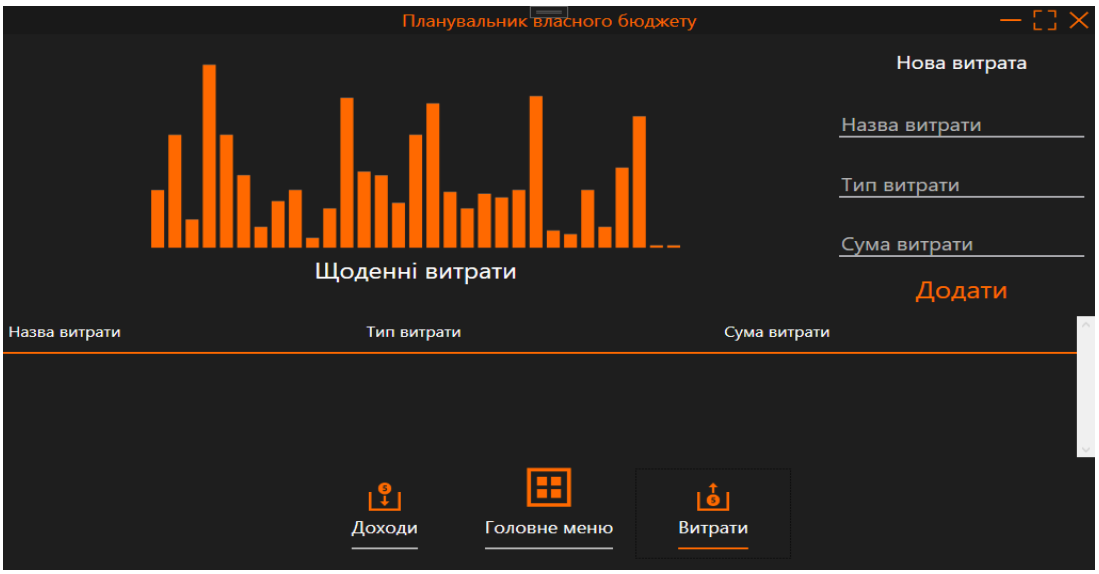

Рисунок 4.11 — Екранна форма «Витрати».

ВИСНОВКИ ДО РОЗДІЛУ 4

У четвертому розділі було представлено реалізацію та тестування мобільного додатку для планування особистого бюджету.

Було описано інтерфейс користувача та основні компоненти додатку, такі як вікна "Головне вікно", "Вікно доходів", "Вікно витрат" та "Майстер налаштувань".

Також було зазначено, які елементи інтерфейсу були використані для створення графічного інтерфейсу, та представлено структуру бази даних.

Крім цього, було надано опис роботи додатку та наведено приклади основних екранних форм, таких як "Привітання майстра налаштувань", "Визначення доходу", "Визначення витрат на день", "Подушка фінансової безпеки", "Створення нової цілі", "Головне вікно", "Доходи" та "Витрати"

ВИСНОВКИ

У ході виконання даної роботи було досліджено процес розробки мобільного додатку для планування особистого бюджету.

Проаналізовано існуючі мобільні додатки для планування особистого бюджету, їхні функції, переваги та недоліки.

Визначено вимоги користувачів та вимоги до системи, включаючи функціональні та нефункціональні вимоги.

Спроектовано архітектуру додатку, що складається з клієнтської частини (Frontend), серверної частини (Backend), бази даних та API.

Описано модулі додатку та їх взаємодія, а також обґрунтовано вибір технологій та інструментів для розробки.

Реалізовано та протестовано мобільний додаток для планування особистого бюджету, описано його інтерфейс користувача та основні компоненти.

У перспективі розвитку такого проекту, як мобільний додаток для планування особистого бюджету, можуть бути наступні напрямки.

По-перше, розширення функціональності. Це включає додавання можливостей управління інвестиціями, боргами та кредитами, реалізацію автоматичного імпорту транзакцій з банківських рахунків, інтеграцію з іншими фінансовими сервісами та платіжними системами.

По-друге, покращення аналітичних можливостей, а саме розширення можливостей формування звітів та їх деталізації, впровадження інструментів для аналізу фінансових даних та надання персоналізованих рекомендацій, а також візуалізацію даних за допомогою інтерактивних графіків та діаграм.

По-третє, підвищення зручності використання. Сюди входить оптимізація інтерфейсу користувача для різних пристроїв та розмірів екранів, впровадження голосового введення даних та інших сучасних

способів взаємодії, а також персоналізація налаштувань додатку відповідно до потреб користувача.

По-четверте, забезпечення безпеки та конфіденційності. Це передбачає посилення захисту даних користувачів за допомогою сучасних методів шифрування та аутентифікації, реалізацію можливості використання біометричної аутентифікації, а також забезпечення відповідності додатку вимогам законодавства щодо захисту персональних даних.

І нарешті, використання новітніх технологій включає застосування штучного інтелекту та машинного навчання для автоматизації фінансового аналізу та надання рекомендацій, впровадження технологій блокчейн для забезпечення безпеки та прозорості фінансових операцій, а також використання хмарних технологій для забезпечення масштабованості та надійності додатку.

Усі ці перспективи спрямовані на створення більш функціонального, зручного та безпечного інструменту для управління особистими фінансами, який допоможе користувачам досягти своїх фінансових цілей та покращити свій фінансовий добробут.

ДЖЕРЕЛА

1. П’ятикоп О., Гніденко В., Воротнікова З. Мобільний застосунок аналізу та прогнозування особистих витрат. Вісник Приазовського Державного Технічного Університету. Серія: Технічні науки, 1(49), 28–35. 2024. https://doi.org/10.31498/2225-6733.49.1.2024.321181
2. Технології створення програмних продуктів та інформаційних систем: навч. посібник / М. Ю. Карпенко, Н. О. Манакова, І. О. Гавриленко; Харків. нац. ун-т міськ. госп-ва ім. О. М. Бекетова. – Харків : ХНУМГ ім. О. М. Бекетова, 2017. – 93 с
3. Аналіз особливостей забезпечення кібербезпеки у банківських мобільних додатках. Єлизавета Логачова, Марина Єсіна, Всеволод Бобух. Комп’ютерні науки та кібербезпека Випуск 1(23), 2023. Міжнародний електронний науково-теоретичний журнал. Харків 2023
4. І.В Машкіна, ВО Абрамов, ТІ Носенко, ЄВ Іваніченко.Навчально-методичний посібник до виконання і захисту кваліфікаційної роботи бакалавра спеціальностей 122 Комп’ютерні науки для здобувачів вищої освіти денної форми навчання, Київський столичний університет імені Бориса Грінченка. Київ, 2024.- 43 с.
5. Теоретичні та практичні аспекти використання математичних методів та інформаційних технологій в освіті й науці:моногр. / за заг. ред. О. Литвин. — К.: Київ. ун-т ім. Б. Грінченка,2021. — 332 с.
6. Яскевич, Владислав Олександрович (2023) JavaScriptбібліотека React Навчальний посібник для студентів спеціальності Комп’ютерні науки Київський університет імені Бориса Грінченка, Україна, Київ. https://elibrary.kubg.edu.ua/id/eprint/48587/

7. В. Яцишин, Н. Шаблій, Д. Денисов призначення і доцільність використання api gateway у комп’ютерних системах. https://elartu.tntu.edu.ua/bitstream/lib/40314/2/XNTK_2022_Yatsyshyn_V-Purpose_and_feasibility_of_80.pdf
8. Інформаційні технології: наука, техніка, технологія, освіта, здоров’я. MicroCAD-2024 Мікросервісна архітектура на прикладі мобільного додатку remnemo Заволодько Г.Е., Заволодько В.В., Скляров О.О., Глєбов Є.В. https://repository.kpi.kharkov.ua/server/api/core/bitstreams/8550d45a-4a9c-4237-874f-0b952206ce12/content
9. Learn Android Tutorial | Android Studio Tutorial - Javatpoint. www.javatpoint.com. URL: https://www.javatpoint.com/androidtutorial (date of access: 08.04.2024).
10. Трофименко О. Г. Організація баз даних : навч. посібник / О. Г. Трофименко, Ю. В. Прокоп, Н. І. Логінова, І. М. Копитчук. 2-ге вид. виправ. і доповн. – Одеса : Фенікс, 2019. – 246 с.]
11. Smyth N. Android Studio Electric Eel Essentials - Java Edition: Developing Android Apps Using Android Studio 2022.1.1 and Java. Payload Media, 2023
12. Розробка веб-додатків з використанням Laravel. Step2Dev. URL: https://step2.dev/uk/blog/laravel (дата звернення: 25.05.2024).]
13. Мікросервісна архітектура для початківців. GlobalLogic. URL: https://www.globallogic.com/ua/insights/blogs/microservices-architecture-for-beginners-part-one/ (дата звернення: 04.02.2024

ДОДАТОК А. КОД ВІЗУАЛЬНОЇ ЧАСТИНИ ГОЛОВНОГО ВІКНА

```
Window x:Class="Financial_expense_control_system.MainWindow"

xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
        xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
        xmlns:d="http://schemas.microsoft.com/expression/blend/2008"

xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
xmlns:wizardelements="clr-namespace:Financial_expense_control_system.MV
C.Views.WizardElements"
        mc:Ignorable="d" WindowStyle="None" ResizeMode="NoResize"
        Height="600" Width="1024">
    <Grid Name="MainPageGrid">
        <Grid.RowDefinitions>
            <RowDefinition Height="30"/>
            <RowDefinition Height="*"/>
        </Grid.RowDefinitions>
        <Grid Background="#1D1D1D"
            MouseLeftButtonDown="MoveForm">

            <StackPanel Orientation="Horizontal"
                    HorizontalAlignment="Center"
                    VerticalAlignment="Center">

                <Label Content="Планувальник власного бюджету"
                    Foreground="#FF6B00"
                    FontSize="18" Padding="0"
                    MouseLeftButtonDown="MoveForm"/>
            </StackPanel>

            <StackPanel Orientation="Horizontal"
                    HorizontalAlignment="Right">

                <Button Width="30"
                    BorderThickness="0"
                    Foreground="#FFF"
                    Margin="0 0 2 0"
                    Click="HideApp"
                    Cursor="Hand">

                    <Button.Background>
```

```
                    <SolidColorBrush Color="Transparent"/>
                </Button.Background>

                <Button.Triggers>
                    <EventTrigger RoutedEvent="MouseEnter">
                        <BeginStoryboard>
                            <Storyboard>
                                                    <ColorAnimation
Storyboard.TargetProperty="(Button.Background).(SolidColorBrush.Color)"
                                        To="#888888" FillBehavior="HoldEnd"
                                        Duration="0:0:0.25"/>
                            </Storyboard>
                        </BeginStoryboard>
                    </EventTrigger>

                    <EventTrigger RoutedEvent="MouseLeave">
                        <BeginStoryboard>
                            <Storyboard>
                                                    <ColorAnimation
Storyboard.TargetProperty="(Button.Background).(SolidColorBrush.Color)"
                                                    To="Transparent"
FillBehavior="HoldEnd"
                                        Duration="0:0:0.25"/>
                            </Storyboard>
                        </BeginStoryboard>
                    </EventTrigger>
                </Button.Triggers>

                <Button.Content>
                    <Image Source="../../icons/HideApp.png"/>
                </Button.Content>
            </Button>

            <Button Width="30"
                    BorderThickness="0"
                    Foreground="#FFF"
                    Margin="0 0 2 0"
                    Click="ChangeSize"
                    Cursor="Hand">

                <Button.Background>
                    <SolidColorBrush Color="Transparent"/>
```

```
                </Button.Background>

                <Button.Triggers>
                    <EventTrigger RoutedEvent="MouseEnter">
                        <BeginStoryboard>
                            <Storyboard>
                                                                <ColorAnimation
Storyboard.TargetProperty="(Button.Background).(SolidColorBrush.Color)"
                                                To="#888888" FillBehavior="HoldEnd"
                                                Duration="0:0:0.25"/>
                            </Storyboard>
                        </BeginStoryboard>
                    </EventTrigger>

                    <EventTrigger RoutedEvent="MouseLeave">
                        <BeginStoryboard>
                            <Storyboard>
                                                                <ColorAnimation
Storyboard.TargetProperty="(Button.Background).(SolidColorBrush.Color)"
                                                                To="Transparent"
FillBehavior="HoldEnd"
                                                Duration="0:0:0.25"/>
                            </Storyboard>
                        </BeginStoryboard>
                    </EventTrigger>
                </Button.Triggers>

                <Button.Content>
                    <Image Source="../../icons/FullScreen.png"/>
                </Button.Content>
            </Button>

            <Button Width="30"
                    BorderThickness="0"
                    Foreground="#FFF"
                    Click="CloseApp"
                    Cursor="Hand">

                <Button.Background>
                    <SolidColorBrush Color="Transparent"/>
                </Button.Background>
```

```
                    <Button.Triggers>
                      <EventTrigger RoutedEvent="MouseEnter">
                        <BeginStoryboard>
                          <Storyboard>
                                                              <ColorAnimation
Storyboard.TargetProperty="(Button.Background).(SolidColorBrush.Color)"
                                          To="#888888" FillBehavior="HoldEnd"
                                          Duration="0:0:0.25"/>
                          </Storyboard>
                        </BeginStoryboard>
                      </EventTrigger>

                      <EventTrigger RoutedEvent="MouseLeave">
                        <BeginStoryboard>
                          <Storyboard>
                                                              <ColorAnimation
Storyboard.TargetProperty="(Button.Background).(SolidColorBrush.Color)"
                                                              To="Transparent"
FillBehavior="HoldEnd"
                                          Duration="0:0:0.25"/>
                          </Storyboard>
                        </BeginStoryboard>
                      </EventTrigger>
                    </Button.Triggers>

                    <Button.Content>
                      <Image Source="../../icons/CloseApp.png"/>
                    </Button.Content>
                  </Button>
                </StackPanel>
              </Grid>

              <Grid Grid.Row="1"
                  Background="#222222">
                <Grid.RowDefinitions>
                  <RowDefinition Height="*"/>
                  <RowDefinition Height="120"/>
                </Grid.RowDefinitions>

                <Grid Name="ContentBox"/>

                <StackPanel Grid.Row="1"
```

```
                    Orientation="Horizontal"
                    HorizontalAlignment="Center"
                    VerticalAlignment="Center"
                    Height="100">

                  <Button Background="Transparent"
                        BorderThickness="0"
                        Cursor="Hand"
                        Width="150"
                        Margin="2 0 2 0"
                        Click="OpenIncomes">

                    <Button.Triggers>
                      <EventTrigger RoutedEvent="MouseEnter">
                        <BeginStoryboard>
                          <Storyboard>
                                                                <ColorAnimation
Storyboard.TargetProperty="(Button.Background).(SolidColorBrush.Color)"
                                            To="#888888" FillBehavior="HoldEnd"
                                            Duration="0:0:0.5"/>
                          </Storyboard>
                        </BeginStoryboard>
                      </EventTrigger>

                      <EventTrigger RoutedEvent="MouseLeave">
                        <BeginStoryboard>
                          <Storyboard>
                                                                <ColorAnimation
Storyboard.TargetProperty="(Button.Background).(SolidColorBrush.Color)"
                                                                To="Transparent"
FillBehavior="HoldEnd"
                                            Duration="0:0:0.5"/>
                          </Storyboard>
                        </BeginStoryboard>
                      </EventTrigger>
                    </Button.Triggers>

                    <Button.Content>
                      <StackPanel VerticalAlignment="Bottom">

                        <Image Source="../../icons/Incomes.png"
                            Width="40" Height="40"
```

```
                    Margin="0"/>

                <Label Padding="0 0 0 10"
                    FontSize="18"
                    Content="Доходи"
                    Foreground="#FFF"
                    HorizontalAlignment="Center"/>

                <StackPanel Name="IncomesActivatedPanel"
                        Height="2" Background="#B4B4B4"/>
            </StackPanel>
        </Button.Content>
    </Button>

    <Button Background="Transparent"
        BorderThickness="0"
        Cursor="Hand"
        Width="150"
        Margin="2 0 2 0"
        Padding="0 0 0 12"
        Click="OpenDashboard">

        <Button.Triggers>
            <EventTrigger RoutedEvent="MouseEnter">
                <BeginStoryboard>
                    <Storyboard>
                                                    <ColorAnimation
Storyboard.TargetProperty="(Button.Background).(SolidColorBrush.Color)"
                                        To="#888888" FillBehavior="HoldEnd"
                                        Duration="0:0:0.5"/>
                    </Storyboard>
                </BeginStoryboard>
            </EventTrigger>

            <EventTrigger RoutedEvent="MouseLeave">
                <BeginStoryboard>
                    <Storyboard>
                                                    <ColorAnimation
Storyboard.TargetProperty="(Button.Background).(SolidColorBrush.Color)"
                                                    To="Transparent"
FillBehavior="HoldEnd"
                                        Duration="0:0:0.5"/>
```

```
                    </Storyboard>
                </BeginStoryboard>
            </EventTrigger>
        </Button.Triggers>

        <Button.Content>
            <StackPanel VerticalAlignment="Bottom">

                <Image Source="../../icons/Dashboard.png"
                    Width="60" Height="60"
                    Margin="0"/>

                <Label Padding="0 0 0 10"
                    FontSize="18"
                    Content="Головне меню"
                    Foreground="#FFF"
                    HorizontalAlignment="Center"/>

                <StackPanel Name="DashboardActivatedPanel"
                        Height="2" Background="#FF6B00"/>
            </StackPanel>
        </Button.Content>
    </Button>

    <Button Background="Transparent"
        BorderThickness="0"
        Cursor="Hand"
        Width="150"
        Margin="2 0 2 0"
        Click="OpenExpenses">

        <Button.Triggers>
            <EventTrigger RoutedEvent="MouseEnter">
                <BeginStoryboard>
                    <Storyboard>
                                                    <ColorAnimation
Storyboard.TargetProperty="(Button.Background).(SolidColorBrush.Color)"
                                    To="#888888" FillBehavior="HoldEnd"
                                    Duration="0:0:0.5"/>
                    </Storyboard>
                </BeginStoryboard>
            </EventTrigger>
```

```
                    <EventTrigger RoutedEvent="MouseLeave">
                      <BeginStoryboard>
                        <Storyboard>
                                                                <ColorAnimation
Storyboard.TargetProperty="(Button.Background).(SolidColorBrush.Color)"
                                                                To="Transparent"
FillBehavior="HoldEnd"
                                                Duration="0:0:0.5"/>
                        </Storyboard>
                      </BeginStoryboard>
                    </EventTrigger>
                  </Button.Triggers>

                  <Button.Content>
                    <StackPanel VerticalAlignment="Bottom">

                      <Image Source="../../icons/Payments.png"
                          Width="40" Height="40"
                          Margin="0"/>

                      <Label Padding="0 0 0 10"
                          FontSize="18"
                          Content="Витрати"
                          Foreground="#FFF"
                          HorizontalAlignment="Center"/>

                      <StackPanel Name="ExpensesActivatedPanel"
                                Height="2" Background="#B4B4B4"/>
                    </StackPanel>
                  </Button.Content>
                </Button>
              </StackPanel>
              <ProgressBar Name="Loader"
                        Grid.Row="1"
                        Height="3"
                        VerticalAlignment="Bottom"
                        Background="Transparent"
                        BorderBrush="Transparent"
                        BorderThickness="0"
                        IsIndeterminate="True"
                        Foreground="#FF6B00"
```

```
                    Visibility="Hidden"/>
        </Grid>
        <Grid Name="Blocker"
            Grid.Row="0"
            Grid.RowSpan="2"
            Opacity="0.7"
            Background="#BB000000">
        </Grid>
        <Grid Name="WizardGrid"
            Grid.Row="0"
            Grid.RowSpan="2"
            Background="Transparent"
            VerticalAlignment="Center"
            HorizontalAlignment="Center">
        </Grid>
    </Grid>
</Window>
```

ДОДАТОК Б. КОД МОДЕЛЕЙ

```
public enum FinanceOperationType
{
    Income,
    Expense
}

public abstract class FinanceOperation
{
    private string _name;
    private string _operationType;
    private readonly FinanceOperationType _financeOperationType;
    private uint _cost;
    private readonly DateTime _date;

    public string Name { get => _name; }

    public string OperationType { get => _operationType; }

        public FinanceOperationType FinanceOperationType { get =>
_financeOperationType; }

    public uint Cost { get => _cost; }

    public DateTime Date { get => _date; }


        public FinanceOperation(string name, string operationType,
FinanceOperationType financeType, uint cost)
    {
        _name = name;
        _operationType = operationType;
        _financeOperationType = financeType;
        _cost = cost;
        _date = DateTime.Now.Date;
    }
}
internal class ExpenseOperation : FinanceOperation
{
     public ExpenseOperation(string name, string operationType, uint cost)
: base(name, operationType, FinanceOperationType.Expense, cost) { }

            public static async Task<List<FinanceOperation>>
GetOperations(SqliteController controller)
    {
        List<FinanceOperation> opers = new();

         SqliteCommand command = new("Select Name, OperationType, Cost,
Date FROM FinanceOperations WHERE FinanceOperationType = 0",
controller.connection);
        var result = await command.ExecuteReaderAsync();

        while (result.Read())
        {
            if (!uint.TryParse(result[2].ToString(), out uint cost))
                throw new("Cannot get datas");

                    opers.Add(new ExpenseOperation(result[0].ToString(),
result[1].ToString(), cost));
        }
```

```
            return opers;
        }

            public static async Task<FinanceOperation> AddNewOperation(string
name, string operationType, uint cost, SqliteController controller)
        {
            ExpenseOperation operation = new(name, operationType, cost);

             SqliteCommand command = new($"INSERT INTO FinanceOperations (Name,
OperationType, FinanceOperationType, Cost, Date) VALUES ('{operation.Name}',
'{operation.OperationType}', 0, {operation.Cost}, '{operation.Date}')",
controller.connection);
            await command.ExecuteNonQueryAsync();
            return operation;
        }
    }

    public class IncomeOperation : FinanceOperation
    {
        public IncomeOperation(string name, string operationType, uint cost) :
base(name, operationType, FinanceOperationType.Income, cost) { }

                    public static async Task<List<FinanceOperation>>
GetOperations(SqliteController controller)
        {
            List<FinanceOperation> opers = new();

              SqliteCommand command = new("Select Name, OperationType, Cost,
Date FROM FinanceOperations WHERE FinanceOperationType = 1",
controller.connection);
            var result = await command.ExecuteReaderAsync();

            while (result.Read())
            {
                if (!uint.TryParse(result[2].ToString(), out uint cost))
                    throw new("Cannot get datas");

                        opers.Add(new IncomeOperation(result[0].ToString(),
result[1].ToString(), cost));
            }

            return opers;
        }

            public static async Task<FinanceOperation> AddNewOperation(string
name, string operationType, uint cost, SqliteController controller)
        {
            IncomeOperation operation = new(name, operationType, cost);

             SqliteCommand command = new($"INSERT INTO FinanceOperations (Name,
OperationType, FinanceOperationType, Cost, Date) VALUES ('{operation.Name}',
'{operation.OperationType}', 1, {operation.Cost}, '{operation.Date}')",
controller.connection);
            await command.ExecuteNonQueryAsync();
            return operation;
        }
    }

    public class Target
    {
        private int _id;
        private string _name;
```

```
        private double _neededCost;
        private double _currentCost;

        public int Id { get => _id;}

        public string Name { get => _name; }

        public double NeededCost { get => _neededCost; }

         public double CurrentCost { get => _currentCost; set => _currentCost =
value; }


            public Target(int id, string name, double neededCost, double
currentCost)
        {
            if (string.IsNullOrWhiteSpace(name))
                    throw new ArgumentNullException("Cannot create new Target
object. Name is null or empty string");

            if (neededCost < 0)
                  throw new ArgumentException("Cannot create new Target object.
NeededCost cannot be less zero");

            if (currentCost < 0)
                  throw new ArgumentException("Cannot create new Target object.
Current cost cannot be less zero");

            _id = id;
            _name = name;
            _neededCost = neededCost;
            _currentCost = currentCost;
        }


            public static async Task<Target> GetLastTarget(SqliteController
controller)
        {
            Target target = null;
              SqliteCommand command = new("SELECT * FROM Targets ORDER BY Id
DESC LIMIT 1", controller.connection);

            var result = await command.ExecuteReaderAsync();

            while (result.Read())
            {
                int.TryParse(result[0].ToString(), out int id);
                string name = result[1].ToString();
                double.TryParse(result[2].ToString(), out double needed);
                double.TryParse(result[3].ToString(), out double current);

                target = new(id, name, needed, current);
            }

            if (target is null)
                  throw new Exception("Cannot execute SQL operation - table is
empty");

            return target;
        }
```

```
                public static async Task<Target> UpdateCurrentTarget(double
currentCost, SqliteController controller)
            {
                if (currentCost < 0)
                    throw new ArgumentException("Cannot update datas. Current cost
cannot be less zero");

                Target target = await GetLastTarget(controller);
                target.CurrentCost += currentCost;

                   SqliteCommand command = new($"UPDATE Targets SET CurrentCost =
{target.CurrentCost} WHERE Id = {target.Id}", controller.connection);
                await command.ExecuteNonQueryAsync();

                return target;
            }


              public static async Task<Target> AddNewTarget(string name, double
neededCost, SqliteController controller)
            {
                if (string.IsNullOrWhiteSpace(name))
                    throw new ArgumentException("Cannot create new target. Name is
null");

                if (neededCost <= 0)
                     throw new ArgumentException("Cannot create new target. Needed
cost cannot be less zero");

                    SqliteCommand command = new($"INSERT INTO Targets (Name,
NeededCost,     CurrentCost)     VALUES     ('{name}',     {neededCost},     0)",
controller.connection);
                await command.ExecuteNonQueryAsync();

                return await GetLastTarget(controller);
            }
        }
```

ДОДАТОК В. КОД КОНТРОЛЕРІВ

```
public class FinanceOperationsController
    {
        private SqliteController _controller;

        public FinanceOperationsController()
        {
            _controller = SqliteController.Initialize();
        }

              public async Task<(string message, List<FinanceOperation>
operations)> GetOperations(bool isIncomes)
        {
            try
            {
                                        return (string.Empty, (isIncomes) ?
IncomeOperation.GetOperations(_controller).Result                          :
ExpenseOperation.GetOperations(_controller).Result);
            }
            catch (Exception e)
            {
                return (e.Message, null);
            }
        }

          public async Task<(string message, FinanceOperation operation)>
AddNewOperation(string name, string type, uint cost, bool isIncome)
        {
            try
            {
                                          return (string.Empty, (isIncome) ?
IncomeOperation.AddNewOperation(name,   type,   cost,   _controller).Result   :
ExpenseOperation.AddNewOperation(name, type, cost, _controller).Result);
            }
            catch (Exception e)
            {
                return (e.Message, null);
            }
        }
    }
public class SqliteController
{
    private static SqliteController controller;
    public SqliteConnection connection;


    private SqliteController()
    {
        connection = new("DataSource=FinancesDB.db;");
        connection.Open();
    }


    public static SqliteController Initialize()
    {
        controller ??= new SqliteController();
        return controller;
    }
}
```

```
public class TargetController
{
    private SqliteController _controller;

    public TargetController()
    {
        _controller = SqliteController.Initialize();
    }


    public async Task<(string message, Target target)> GetLastTarget()
    {
        try
        {
                                                return    (string.Empty,    await
Target.GetLastTarget(_controller));
        }

        catch (Exception e)
        {
            return (e.Message, null);
        }
    }

                 public    async    Task<(string    message,    Target    target)>
UpdateCurrentTarget(double currentCost)
    {
        try
        {
                                                return    (string.Empty,    await
Target.UpdateCurrentTarget(currentCost, _controller));
        }

        catch (Exception e)
        {
            return (e.Message, null);
        }
    }

     public async Task<(string message, Target target)> AddNewTarget(string
name, double neededCost)
    {
        try
        {
                          return  (string.Empty,  await  Target.AddNewTarget(name,
neededCost, _controller));
        }

        catch (Exception e)
        {
            return (e.Message, null);
        }
    }
}
```