МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

ЧЕРКАСЬКИЙ ДЕРЖАВНИЙ БІЗНЕС-КОЛЕДЖ

кафедра комп’ютерної інженерії та інформаційних технологій

КВАЛІФІКАЦІЙНА РОБОТА

на тему

Особливості розробки сучасних персональних фінансових додатків

Виконала: студентка групи 2П-20
Спеціальності
121 – «Інженерія програмного забезпечення»

Єлизавета ДВОРЯКІВСЬКА

Керівник:
Станіслав МАРЧЕНКО

Черкаси 2024

ЧЕРКАСЬКИЙ ДЕРЖАВНИЙ БІЗНЕС-КОЛЕДЖ

Кафедра комп'ютерної інженерії та інформаційних технологій
(повна назва випускової кафедри)

Спеціальність 121 "Інженерія програмного забезпечення"
(шифр і назва спеціальності)

Освітня програма Інженерія програмного забезпечення
(назва освітньої програми)

ЗАТВЕРДЖУЮ

Завідувач кафедри
комп'ютерної інженерії та
інформаційних технологій
(назва кафедри)
_____ Хотунов В.І.
(підпис) (ПІБ)
«_____»_____2023 р.

З А В Д А Н Н Я

НА ВИПУСКНУ РОБОТУ СТУДЕНТУ

Дворяківській Єлизаветі Георгіївні
(прізвище, ім'я, по батькові студента)

1. Тема випускної роботи Особливості розробки сучасних персональних фінансових додатків

Керівник роботи Марченко Станіслав Віталійович, спеціаліст І категорії
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)
затверджені наказом закладу вищої освіти від " 13 " жовтня 2023 року № 65У.

2. Строк подання студентом випускної роботи 03.06.2024

3. Вихідні дані до випускної роботи інструмент для створення діаграм та схем Draw.io , прототипування засобами онлайн-сервіс розробки інтерфейсів Figma.

4. Зміст випускної роботи (перелік питань, які потрібно розробити) Огляд поточного стану предметної області. Проєктування персональних фінансових додатків на основі функціональних вимог. Прототипування персонального фінансового додатка з урахуванням атрибутів якості.

5. Дата видачі завдання 15.09.2023р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів кваліфікаційної роботи	Терміни виконання етапів	Примітка про виконання з підписами наукового керівника і студента
1	Вступ	20.10.2023	
2	Розділ 1. Огляд поточного стану предметної області	22.12.2023	
3	Розділ 2. Проєктування персональних фінансових додатків на основі функціональних вимог	15.03.2024	
4	Розділ 3. Прототипування персонального фінансового додатка з урахуванням атрибутів якості	15.05.2024	
5	Висновки	17.05.2024	
6	Оформлення випускної роботи (чистовий варіант)	27.05.2024	
7	Здача випускної роботи на кафедру для рецензування (за 14 днів до захисту)	31.05.2024	
8	Перевірка випускної роботи на наявність ознак плагіату (за 10 днів до захисту)	03.06.2024	
9	Подання випускної роботи на затвердження завідувачу кафедри (за 7 днів до захисту)	06.06.2024	

Студент _____ Дворяківська Є.Г.
(підпис) (прізвище та ініціали)

Керівник роботи _____ Марченко С.В.
(підпис) (прізвище та ініціали)

АНОТАЦІЯ

Персональний фінансовий додаток – це спеціалізований програмний продукт, призначений для управління особистими фінансовими ресурсами та відстеження фінансових операцій користувача. Такий додаток дозволяє відстежувати доходи та витрати, планувати бюджет, контролювати заощадження, здійснювати платежі, аналізувати фінансові звички та тенденції. Персональні фінансові додатки можуть інтегруватися з банківськими рахунками, кредитними картками, інвестиційними портфелями та іншими фінансовими інструментами, забезпечуючи централізований доступ до всієї фінансової інформації користувача в одному місці.

Вони також можуть пропонувати функції бюджетування, встановлення фінансових цілей, відстеження позик, створення звітів та візуалізацію даних. Персональні фінансові додатки покликані полегшити управління особистими фінансами, підвищити фінансову грамотність та допомогти досягти фінансової стабільності та добробуту.

У кваліфікаційній роботі проведено ґрунтовний аналіз ринку аналогічних додатків, визначено їхні переваги та недоліки. Досліджено специфічні вимоги до персональних фінансових додатків, включно з питаннями безпеки, конфіденційності даних та зручності використання. Розглянуто різні архітектурні підходи до розробки програмного забезпечення та обрано найоптимальніший варіант. Спроєктовано базу даних для зберігання фінансової інформації користувачів. Створено прототип користувацького інтерфейсу додатку в Figma з урахуванням принципів юзабіліті та візуалізації даних. Результати дослідження можуть бути використані для подальшої розробки високоякісних персональних фінансових додатків.

ABSTRACT

A personal finance application is a specialized software product designed to manage personal financial resources and track a user’s financial transactions. Such an application allows you to track income and expenses, plan a budget, monitor savings, make payments, and analyze financial habits and trends. Personal finance applications can integrate with bank accounts, credit cards, investment portfolios, and other financial instruments, providing centralized access to all of the user’s financial information in one place.

They can also offer budgeting features, setting financial goals, tracking loans, generating reports, and data visualization. Personal finance applications are intended to facilitate personal finance management, increase financial literacy, and help achieve financial stability and well-being.

In the qualification work a comprehensive analysis of the market for similar applications was conducted, identifying their advantages and disadvantages. The specific requirements for personal finance applications were investigated, including security, data privacy, and usability concerns. Various software architectural approaches were considered, and the optimal solution was selected. A database was designed to store users’ financial information. A prototype of the application’s user interface was created in Figma, taking into account usability principles and data visualization. The research results can be utilized for further development of high-quality personal finance applications.

ПЕРЕЛІК СКОРОЧЕНЬ, УМОВНИХ ПОЗНАЧЕНЬ

CSV (від анг. Comma-Separated Values) – файловий формат

BLL (від анг. Business Logic Layer) – бізнес модель

DAL (від анг. Data Access Layer) – прошарок архітектури програмного забезпечення

ЗМІСТ

ВСТУП 3

РОЗДІЛ 1. ОГЛЯД ПОТОЧНОГО СТАНУ ПРЕДМЕТНОЇ ОБЛАСТІ 5

1.1. Порівняння методів ведення фінансових обчислень 5

1.2. Огляд існуючих аналогів на ринку 7

1.3. Постановка задачі 16

РОЗДІЛ 2. ПРОЄКТУВАННЯ ПЕРСОНАЛЬНИХ ФІНАНСОВИХ ДОДАТКІВ НА ОСНОВІ ФУНКЦІОНАЛЬНИХ ВИМОГ 17

2.1. Визначення основних функціональних вимог 17

2.2 Вибір архітектурного стилю додатка 22

2.3. Огляд обраної архітектури додатку 35

РОЗДІЛ 3. ПРОТОТИПУВАННЯ ПЕРСОНАЛЬНОГО ФІНАНСОВОГО ДОДАТКА З УРАХУВАННЯМ АТРИБУТІВ ЯКОСТІ 42

3.1. Важливість прототипування під час розробки програмного забезпечення . 42

3.2. Розробка прототипу фінансового застосунку 43

3.2.1. Огляд обраного інструменту для прототипування 44

3.2.2. Розробка бази даних для фінансового додатку 47

3.3. Проєктування та прототипування користувацького інтерфейсу фінансового додатку у Figma 52

ВИСНОВКИ 58

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ 59

ДОДАТОК А

ВСТУП

Актуальність обраної теми. У сучасному світі, де фінансові питання відіграють ключову роль у повсякденному житті, керування власними грошовими ресурсами стає невід'ємною складовою успіху та благополуччя. Справа не лише в тому, скільки ви заробляєте, але й у тому, наскільки раціонально ви розпоряджаєтесь своїми доходами. Фінансовий менеджмент – це комплексний процес, що охоплює управління грошовими потоками, оптимізацію витрат та належний розподіл коштів відповідно до пріоритетів та цілей.

Ведення особистої бухгалтерії, ретельне планування надходжень і видатків не є обов'язковою вимогою для кожної людини. Проте такий підхід допомагає більш ефективно використовувати наявні ресурси, уникати непотрібних витрат і досягати фінансової стабільності. На перший погляд, управління фінансами здається складним завданням, яке вимагає спеціальних знань та навичок. І частково це дійсно так – керувати фінансами підприємства, навіть невеликого, без відповідної кваліфікації та експертизи буде дуже складно.

Однак, коли йдеться про особисті фінанси, ситуація дещо інша. Контролювати та оптимізувати свої доходи й витрати цілком можливо, маючи лише базові знання у цій сфері. Головне – усвідомити важливість фінансової грамотності та прикладати зусилля для вдосконалення своїх навичок керування грошима. Адже від цього значною мірою залежить якість вашого життя та можливість досягати поставлених цілей.

Об'єкт дослідження. Об'єктом дослідження є процеси розробки персональних фінансових додатків.

Предмет дослідження. Предметом дослідження виступають моделі, методи та технології розробки сучасних персональних фінансових додатків.

Мета дослідження. Метою дослідження є надання комплексного розуміння процесу розробки персональних фінансових застосунків, висвітлення ключових викликів та надання рекомендації щодо створення успішних та ефективних рішень, які допоможуть користувачам краще керувати своїми фінансами в епоху цифрових технологій

Завдання дослідження. Для досягнення цієї мети поставленні наступні задачі:

1) Вивчення тенденцій ринку фінансових додатків та ідентифікація потреб користувачів.

2) Архітектурний аналіз сучасних технологій розробки персональних фінансових додатків.

3) Проєктування та прототипування персонального фінансового додатку.

РОЗДІЛ 1
ОГЛЯД ПОТОЧНОГО СТАНУ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1. Порівняння методів ведення фінансових обчислень

Управління фінансами є невід'ємною складовою повсякденного життя як для окремих людей, так і для компаній та організацій. Раніше всі розрахунки проводилися вручну за допомогою паперу, олівця та калькулятора. Проте, з розвитком технологій та зростанням популярності смартфонів і персональних комп'ютерів, з'явилася можливість використовувати спеціалізовані фінансові програми для здійснення фінансових обчислень [1].

Ручні розрахунки мають як переваги, так і недоліки. Вони дозволяють зберігати більший контроль над процесом та краще розуміти сутність кожної операції. Крім того, паперові записи є фізичним підтвердженням проведених обчислень, що може знадобитися для аудиту чи перевірки. Однак ручні обчислення є більш трудомісткими, займають більше часу та підвищують ризик виникнення помилок, особливо при роботі з великими обсягами даних або складними формулами.

З іншого боку, використання фінансових додатків має низку переваг, які роблять процес фінансових обчислень більш ефективним та зручним. Насамперед, ці програми автоматизують обчислення, мінімізуючи ризик помилок та економлячи час. Вони забезпечують швидкі та точні результати, що є дуже важливим у бізнес-середовищі, де час та ефективність мають вирішальне значення.

Більшість фінансових додатків також пропонують широкий спектр функцій, таких як створення бюджетів, відстеження витрат, управління інвестиціями та генерування звітів. Ці можливості дозволяють користувачам отримувати повну картину своїх фінансів, приймати обґрунтовані рішення та планувати майбутні витрати та доходи. Фінансові додатки часто забезпечують безпечне хмарне сховище даних, що дає змогу користувачам мати доступ до своїх фінансових записів з будь-якого пристрою та в будь-якому місці. Це

особливо зручно для людей, які ведуть активний спосіб життя або працюють віддалено.

Важливо зазначити, що сучасні фінансові додатки не лише забезпечують зручність та ефективність, а й підтримують високі стандарти безпеки та конфіденційності. Вони використовують шифрування даних та інші заходи безпеки для захисту конфіденційної фінансової інформації користувачів.

Багато фінансових додатків пропонують інтеграцію з банками та фінансовими установами, що полегшує відстеження транзакцій та автоматичне оновлення даних. Ця функція допомагає заощадити час та забезпечує точнішу картину фінансового стану користувача.

Хоча ведення фінансових обчислень вручну може бути доречним для певних людей або ситуацій, використання фінансових додатків стало стандартом у сучасному світі. Завдяки своїй ефективності, зручності та широкому спектру функцій, ці програми допомагають користувачам краще керувати своїми фінансами, економити час та приймати більш обґрунтовані фінансові рішення.

За результатами проведеного дослідження від Sensor Tower, за перший квартал 2021 року кількість завантажень фінансових додатків зросла на 34% порівняно з цим самим періодом в Європі та США, статистику представлено на рис. 1.1 [2].

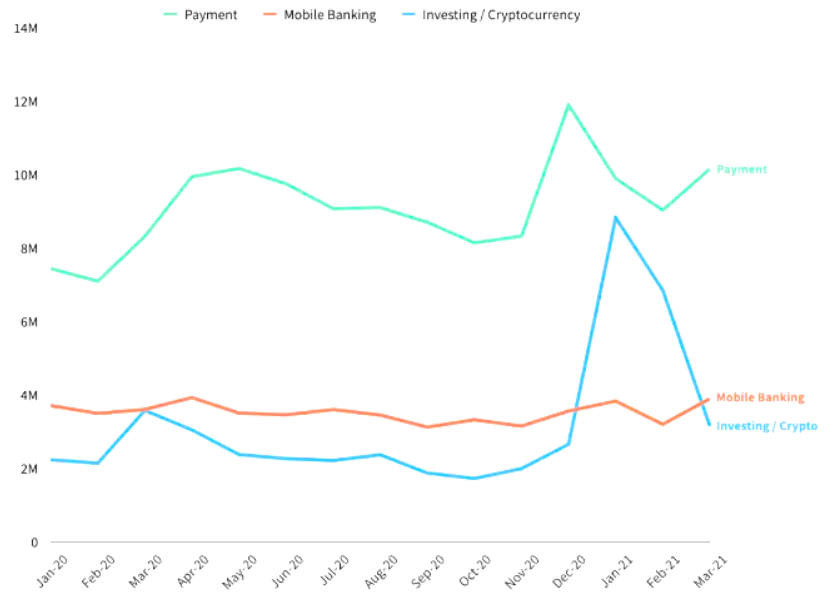

Рисунок 1.1 – Результати дослідження Sensor Tower

1.2. Огляд існуючих аналогів на ринку

Нині існує широкий вибір програмних рішень, призначених для полегшення процесу обліку та управління персональними фінансами. У цьому розділі буде розглянуто найпопулярніші та найбільш затребувані програми даної категорії [3, 4].

Money Lover – це популярний мобільний додаток для ведення персонального фінансового обліку та контролю над витратами.

Money Lover забезпечує інструменти для обліку боргових зобов'язань та регулярних платежів, а також своєчасно нагадує користувачам про необхідність здійснити чергові виплати. У налаштуваннях додатку можна встановити бажані фінансові цілі, до яких користувач буде прагнути. Це допомагає ефективно контролювати власний бюджет, відстежувати стан ощадних рахунків та уникати потенційних фінансових проблем [5].

Преміум-версія Money Lover пропонує додаткову функцію експорту даних у формат Excel, що полегшує подальшу роботу з фінансовою інформацією. Крім того, у платній підписці відсутня рекламна складова, що забезпечує більш зручний та незаважаючий користувацький досвід. Завдяки комплексному набору функцій та можливості встановлення фінансових цілей, Money Lover стає потужним інструментом для ефективного управління особистими фінансами, контролю бюджету, відстеження боргів та заощаджень, що в кінцевому рахунку допомагає уникнути фінансових труднощів.

Переваги Money Lover:

– Додаток дозволяє легко вносити дані про всі свої фінансові операції, категоризувати їх та відстежувати залишки.

– Користувачі можуть встановлювати бюджетні ліміти для різних категорій витрат і відстежувати, наскільки вони дотримуються запланованого бюджету.

– Money Lover генерує наочні графіки та діаграми, що допомагає візуально аналізувати фінансові тенденції.

– Дані можна синхронізувати між декількома пристроями за

допомогою хмарного сервісу.

– Додаток підтримує численні валюти та автоматично конвертує їх за поточним курсом.

– Є можливість відстежувати свої борги та кредити.

– Money Lover пропонує гнучкі інструменти для фільтрації та генерації детальних звітів.

Завдяки інтерфейсу та широкому набору функцій Money Lover є одним з найпопулярніших додатків для персонального фінансового менеджменту на ринку.

Недоліки застосунку:

– Відсутність функції автоматичного імпорту фінансових даних з банківських рахунків користувача. Для відстеження своїх доходів та витрат необхідно вручну вносити інформацію про кожну транзакцію в додатку.

– Обмежені можливості для довгострокового фінансового планування. Хоча додаток дозволяє встановлювати певні фінансові цілі, функціоналу для деталізованого планування бюджету на тривалі періоди та побудови складних фінансових стратегій бракує.

– Невелика кількість вбудованих категорій для класифікації різних типів доходів та витрат. Існуючих опцій може бути недостатньо, а можливості створення власних категорій є доволі обмеженими.

– Відсутність інтеграції Money Lover з іншими фінансовими додатками чи онлайн-сервісами. Це обмежує можливості для комплексного обліку та управління фінансами з єдиного центру.

– Користувачам також бракує належних інструментів для спільного доступу до фінансових даних з іншими особами, наприклад, членами сім’ї.

– Для отримання доступу до розширених функцій, таких як експорт даних чи відключення реклами, потрібно оформити платну підписку на преміум-версію додатку.

Завдяки інтерфейсу та широкому набору функцій, які представлені на рис. 1.2. Money Lover є одним з найпопулярніших додатків для персонального фінансового менеджменту на ринку.

Wallet – це фінансовий трекер та особистий електронний гаманець. З його допомогою ви можете створити план майбутніх заощаджень, планувати бюджет на кілька наступних років, а також контролювати та управляти своїми фінансами в реальному часі. Інтерфейс та функціїї додатку представлено на рис.1.3.

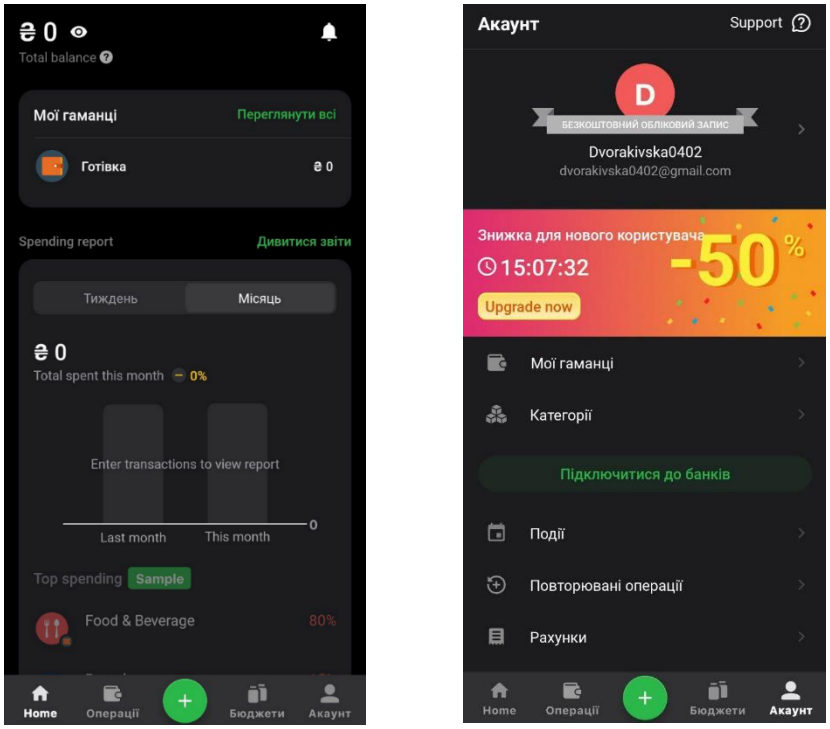

Рисунок. 1.2 – Інтерфейс додатку Money Lover

Переваги застосунку Wallet:

– розробка та впровадження автоматизованого оновлення банківської системи для ефективного управління фінансовими операціями та покращення сервісу для клієнтів;

– мультифункціональність, яка допомагає досягти цілей, включаючи підвищення продуктивності, оптимізацію ресурсів і покращення взаємодії з клієнтами. Ця мультифункціональність дозволить також швидко відреагувати у разі погіршення фінансового стану, забезпечуючи вчасне інформування

керівництва і надання рекомендацій щодо протидії таким ситуаціям;

– детальні звіти, які містять всю необхідну інформацію про ваші фінансові операції, та зрозумілі діаграми, що наглядно демонструють стан ваших фінансів на кредитних рахунках. Ці інструменти полегшують контроль за борговою ситуацією і допомагають вчасно виявляти проблеми;

– можливість розгляду спільного використання вибраних облікових записів з іншими користувачами, що дозволить ефективніше та більш продуктивно використовувати наявні ресурси;

– підтримка різноманітних форматів документів для зручності користувача, зокрема такі популярні формати, як CSV, XLS та PDF;

– для забезпечення додаткового рівня захисту, програму можна заблокувати пін-кодом. Це дозволить обмежити доступ до програми лише для тих, хто знає цей пін-код, забезпечуючи таким чином її безпеку та конфіденційність.

Завдяки зручним інструментам контролю бюджету, візуалізації та аналізу Wallet допомагає тримати фінанси під контролем та раціонально розпоряджатися коштами.

Недоліки застосунку:

– проблема відсутності підключення до банкінгу монобанку, який є одним з найбільш зручних та сучасних банківських інструментів;

– відсутність вбудованого калькулятора є обмеженням під час додавання записів. Якщо вам потрібно додати кілька витрат або розрахувати податок, це необхідно зробити окремо, що може бути не зручно і займати додатковий час;

– встановити точний час здійснення платежу неможливо. Це може призвести до того, що порядок всіх платежів, які мають бути здійснені, може бути порушено;

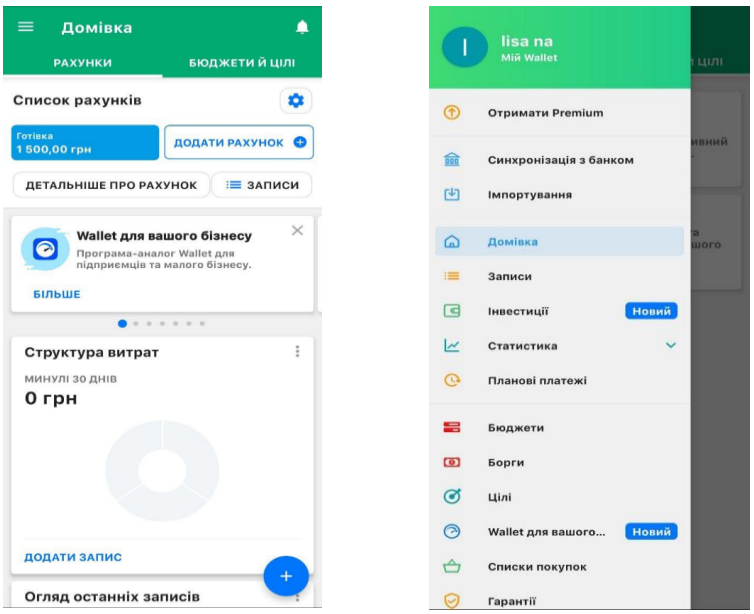

Рисунок 1.3 – Інтерфейс додатку Wallet

Spendee – це зручний та функціональний мобільний застосунок, призначений для всебічного управління особистими фінансами. Він доступний для пристроїв на базі операційних систем iOS та Android, що робить його легкодоступним для більшості смартфонів та планшетів. Інтерфейс і функціональні можливості додатку Spendee зображені на рис.1.4.

Spendee пропонує користувачам потужні інструменти для детального аналізу та контролю над фінансовими операціями. Кожну окрему транзакцію можна ретельно відфільтрувати за численними параметрами, що дозволяє отримувати максимально деталізовану картину руху коштів. Крім того, є можливість імпортувати дані про транзакції з інших фінансових додатків або з електронних таблиць Excel, заощаджуючи час на ручному введенні інформації. А для створення резервних копій чи переносу даних в інші програми, користувачі можуть експортувати свої фінансові записи у популярних форматах CSV та XLS [6].

Переваги застосунку Spendee:

– наявна автоматична синхронізація даних у режимі реального часу, завдяки якій усі зміни одразу оновлюються на всіх пристроях користувача. Це забезпечує актуальність інформації та зручність доступу до фінансових даних з

різних гаджетів;

– є можливість створення окремих «гаманців» для різних цілей та намірів, наприклад, заощадження на відпустку, придбання омріяної речі чи організацію святкування дня народження. Користувачі можуть розподіляти свої доходи та витрати за відповідними гаманцями, спрощуючи процес фінансового планування;

– застосунок дає змогу обмінюватися гаманцями з іншими користувачами, надаючи можливість налаштувати рівні доступу для спільного перегляду або редагування даних;

– Spendee пропонує функцію геомітки, яка автоматично фіксує координати місця покупки або оплати, це дозволяє зручно відстежувати місцезнаходження транзакцій;

– забезпечує цілодобову професійну технічну підтримку користувачів, це дозволяє оперативно вирішувати проблеми, які виникають у користувачів.

При такій кількості переваг застосунок все ж має й недоліки:

– відсутність інтеграції з популярним українським банком Monobank доволі незручно для користувачів. Можливість підключення до банкінгу Monobank-у значно спростила б процес імпорту фінансових даних та синхронізацію транзакцій;

– відсутність вбудованого калькулятора у Spendee. Коли постає потреба додати кілька витрат одночасно або розрахувати суму з урахуванням податків чи зборів, доводиться виконувати ці операції окремо в іншому додатку;

– неможливість встановити точний час здійснення платежу чи транзакції. Додаток автоматично присвоює час внесення запису як час операції. Це може призводити до того, що хронологічний порядок всіх фінансових операцій виявиться порушеним, що ускладнить аналіз даних та спотворить загальну картину руху коштів;

– відсутня функція автоматичного імпорту даних безпосередньо з банківських рахунків користувача. Це значно спростило б процес обліку фінансових операцій та зробило б його більш комфортним.

Загалом, незважаючи на певні недоліки, Spendee залишається одним з лідерів серед мобільних додатків для управління фінансами завдяки своїй функціональності, зручності та якісній візуалізації даних.

Monobudget – це фінансовий застосунок для смартфонів, розроблений для допомоги користувачам у відстеженні їхніх доходів та витрат. На рис.1.5 зображені інтерфейс та основні функції застосунку Monobudget.

Переваги застосунку Monobudget:

– Monobudget має простий та зрозумілий інтерфейс. Користувачам не потрібно проходити навчання чи читати громіздкі інструкції, щоб розібратися з функціональністю застосунку;

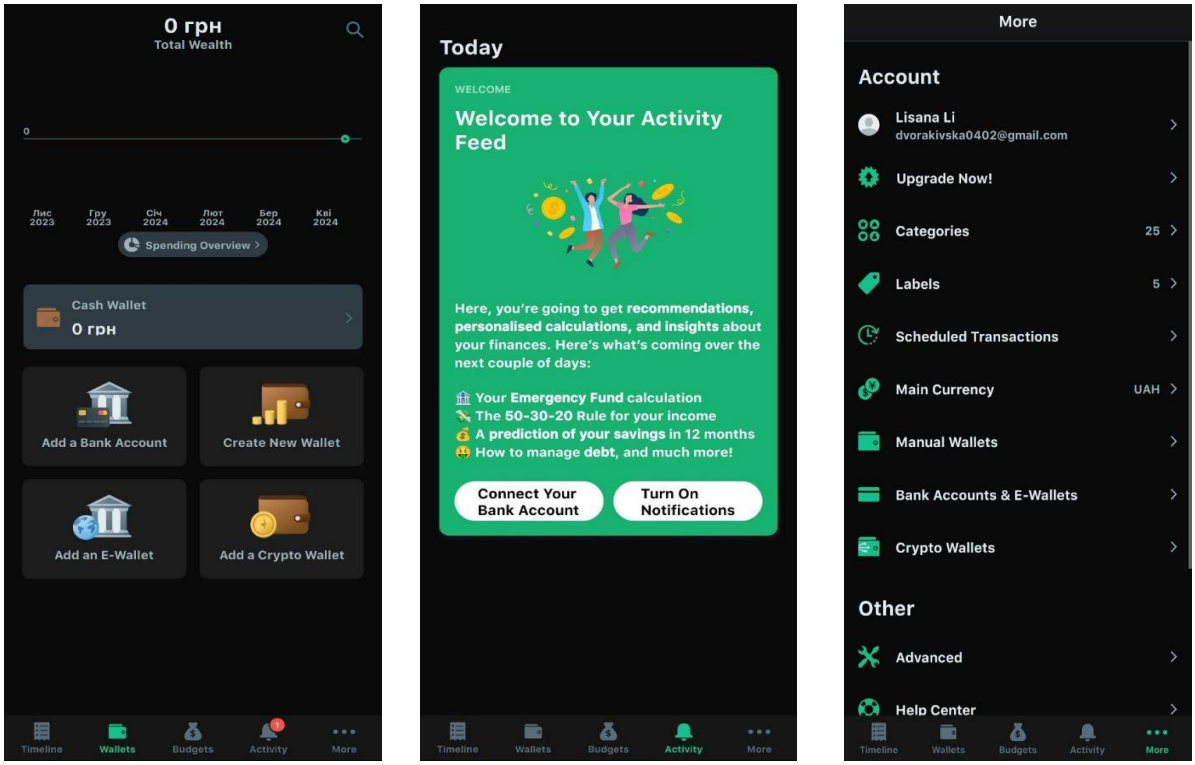

Рисунок 1.4 – Інтерфейс додатку Spendee

– може автоматично розпізнавати та категоріювати транзакції на основі назви або місцезнаходження. Це значно спрощує процес відстеження витрат і дозволяє користувачам економити час;

– можливість встановлювати бюджети для різних категорій витрат. Користувачі можуть визначати ліміти для своїх витрат на їжу, розваги, транспорт та інші категорії, що допомагає їм краще контролювати свої фінансові потоки;

– Monobudget пропонує наочні звіти та графіки, які дають змогу користувачам візуально відстежувати свої фінанси. Ці функції забезпечують швидкий та зрозумілий огляд витрат, доходів і загального фінансового стану;

– синхронізація даних між кількома пристроями. Користувачі можуть легко переносити свої фінансові дані з одного пристрою на інший, забезпечуючи доступ до своїх даних у будь-якому місці та в будь-який час;

– додаток абсолютно безкоштовний, що значно збільшує кількість користувачів.

Недоліки застосунку:

– не має можливості безпосередньо підключатися до банківських рахунків, тому всі транзакції потрібно вводити вручну;

– у безкоштовній версії функціональність є обмеженою, і багато корисних функцій доступні лише в преміум-підписці. Деякі користувачі також скаржаться на відсутність таких функцій, як відстеження витрат на подорожі чи інвестиції;

– проблематичність спільного використання бюджету між членами сім’ї. Це може ускладнити координацію фінансів для сімейних пар або сімей з кількома особами, які мають спільні витрати;

Monobudget є привабливим вибором для тих, хто прагне тримати свої фінанси під контролем завдяки своєму простому та інтуїтивному дизайну, автоматизованим функціям відстеження витрат і наочним звітам та графікам.

Згадані мобільні додатки призначені для допомоги користувачам у відстеженні фінансових надходжень і витрат, створенні бюджетів та аналізі звичок споживання. Їхній інтуїтивний і зручний інтерфейс полегшує управління коштами. Ключові переваги цих застосунків включають синхронізацію між пристроями, автоматичне розпізнавання банківських транзакцій, нагадування про майбутні платежі та візуалізацію даних у графічному вигляді.

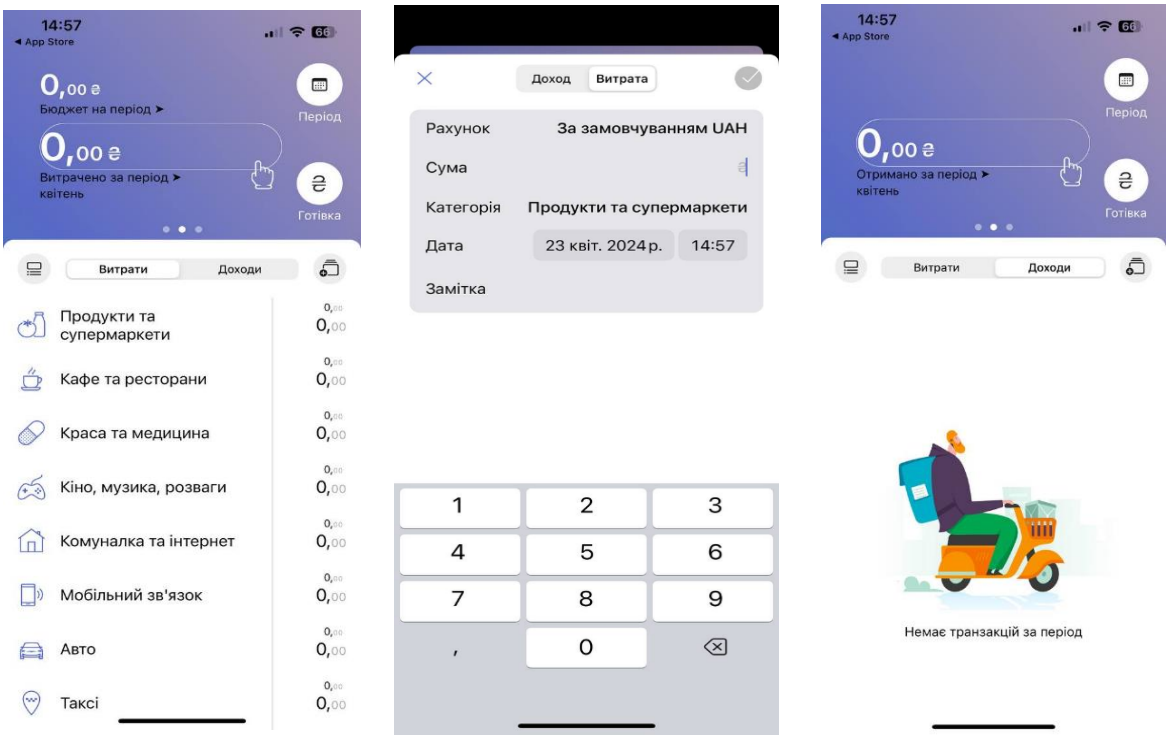

Рисунок 1.5 – Інтерфейс додатку Monobudget

Однак кожен додаток має свої недоліки. Деякі мають обмежену кількість категорій витрат, інші не підтримують різні валюти чи не інтегруються з певними банками. Крім того, розширені функції, як правило, потребують додаткової оплати.

Загалом, усі ці програми є корисними інструментами для управління особистими фінансами. Вибір найбільш доречного варіанту залежить від індивідуальних потреб користувача, таких як необхідність підтримки конкретної валюти, кількість банківських рахунків чи бажання отримати покращені аналітичні можливості.

Далі в табл.1.1 представлено порівняння ключових особливостей і відмінностей між різними мобільними застосунками для відстеження фінансів, які були розглянуті раніше. Наведена таблиця демонструє ключові параметри, функціональні можливості та відмінні риси кожного з розглянутих фінансових застосунків.

Таблиця 1.1 – Порівняльна характеристика аналогів на ринку

Характеристика	Monobudget	Spendee	Wallet	Money Lover
Безкоштовна версія	Так	Так	Так	Так
Преміум – версія	Так	Так	Так	Так
Синхронізація між пристроями	Так	Так	Так	Так
Автоматичне розпізнавання транзакцій	Так	Так	Частково	Так
Кількість доступних валют	Понад 200	Понад 170	Обмежена	Декілька Основних
Нагадування про платежі	Так	Так	Так	Так
Створення бюджетів	Так	Так	Так	Так
Візуалізація даних	Графіки, діаграми	Графіки, діаграми	Діаграми	Графіки, діаграми
Інтеграція з банківськими рахунками	Часткова	Повна	Обмежена	Часткова
Підтримка різних валют	Так	Так	Ні	Частково

1.3. Постановка задачі

Відповідно до мети та завдань кваліфікаційної роботи деталізуємо обсяг виконання робіт. Це дослідження передбачає:

– проведення огляду ринку персональних фінансових додатків, аналіз їх основних функцій, особливостей та популярності серед користувачів.

– визначення ключових вимог до персональних фінансових додатків, таких як безпека, конфіденційність, зручність використання, інтеграція з банківськими рахунками та фінансовими інструментами.

– аналіз проблем та викликів, з якими стикаються розробники під час створення персональних фінансових додатків, зокрема, питань захисту даних, відповідності нормативним вимогам, масштабованості та продуктивності.

– аналіз технологічних рішень та архітектурних підходів, що використовуються для розробки персональних фінансових додатків.

РОЗДІЛ 2
ПРОЄКТУВАННЯ ПЕРСОНАЛЬНИХ ФІНАНСОВИХ ДОДАТКІВ НА ОСНОВІ ФУНКЦІОНАЛЬНИХ ВИМОГ

2.1. Визначення основних функціональних вимог

Управління доходами та витратами. Одна з найважливіших функцій персонального фінансового додатку – це можливість ефективно управляти доходами та витратами користувача. Ця функція дозволяє користувачам вводити або імпортувати детальні дані про свої фінансові надходження та витрати. Додаток повинен забезпечувати зручний спосіб категоризації транзакцій за типами, такими як зарплата, їжа, транспорт, розваги тощо. Це допоможе користувачам краще відстежувати та аналізувати свої витрати в різних сферах життя.

Додаток має надавати наочну візуалізацію доходів та витрат у формі графіків, діаграм або звітів. Візуальне представлення фінансових даних допомагає користувачам швидко виявляти тенденції та закономірності, а також легко відстежувати свої фінансові потоки. Функція пошуку та фільтрації транзакцій за датою, категорією або сумою також є вкрай важливою, оскільки вона дозволяє користувачам знаходити та аналізувати конкретні операції, що становлять інтерес [7].

Бюджетування та планування. Ефективне управління доходами та витратами є фундаментальною частиною персонального фінансового планування. Надаючи користувачам зручні інструменти для відстеження їхніх фінансових операцій, категоризації витрат та візуалізації даних, додаток допомагає їм краще розуміти свої фінансові звички та приймати більш обґрунтовані рішення щодо управління грошима.

Персональний фінансовий додаток повинен надавати користувачам потужні інструменти для створення та відстеження бюджетів для різних категорій витрат, таких як житло, їжа, транспорт, розваги тощо. Це дозволить їм чітко розуміти, скільки коштів вони можуть витрачати на певні потреби, не виходячи за рамки своїх фінансових можливостей.

Крім того, додаток має забезпечувати можливість встановлення конкретних цілей та лімітів для кожного бюджету. Наприклад, користувач може визначити максимальну суму, яку він може витрачати на їжу щомісяця, або поставити ціль заощадити певну суму на великі покупки чи відпустку. Функція відстеження фактичних витрат порівняно з бюджетом дозволить користувачам стежити за своїми витратами в режимі реального часу та вносити необхідні коригування.

Одним з ключових аспектів бюджетування є можливість отримувати сповіщення, коли витрати наближаються або перевищують встановлені ліміти. Це допоможе користувачам уникати невиправданих витрат та дотримуватися бюджету. Персональний фінансовий додаток має надавати зручні інструменти для планування майбутніх витрат та доходів, дозволяючи користувачам прогнозувати свої фінансові потоки та приймати обґрунтовані рішення щодо управління коштами.

Ретельне бюджетування та планування є основою фінансової стабільності та досягнення довгострокових цілей. Забезпечуючи користувачів необхідними інструментами для створення, відстеження та коригування бюджетів, а також планування майбутніх фінансових операцій, персональний фінансовий додаток стає потужним помічником у раціональному управлінні грошовими ресурсами.

Інтеграція з банківськими рахунками та фінансовими установами. Інтеграція з банківськими рахунками та фінансовими установами є однією з найбільш важливих і водночас складних функцій персонального фінансового додатку. Ця функція дозволяє користувачам безпосередньо підключати свої банківські рахунки, кредитні та дебетові картки до додатку, забезпечуючи безперервну синхронізацію даних.

Після успішного підключення рахунків додаток автоматично імпортує всі транзакції, здійснені користувачем, звільняючи його від потреби вручну вводити ці дані. Це значно спрощує процес відстеження доходів та витрат, забезпечуючи точність і актуальність інформації. Крім того, додаток постійно відстежує

баланси та залишки на підключених рахунках, надаючи користувачам актуальну картину їхнього фінансового стану.

Деякі просунуті персональні фінансові додатки можуть також інтегруватися з платіжними системами банків, дозволяючи користувачам здійснювати платежі, перекази коштів або навіть інвестиції безпосередньо з додатку. Ця функція забезпечує зручність та економію часу, оскільки користувачам більше не потрібно переходити на різні веб-сайти або мобільні додатки для здійснення фінансових операцій.

Однак інтеграція з банківськими рахунками та фінансовими установами вимагає найвищого рівня безпеки та конфіденційності. Персональний фінансовий додаток повинен використовувати шифрування даних, багаторівневу автентифікацію та інші заходи захисту для запобігання несанкціонованого доступу або витоку конфіденційної інформації. Додаток також має дотримуватися всіх відповідних нормативних вимог та стандартів безпеки.

Інтегрована функція роботи з банківськими рахунками та фінансовими установами значно підвищує зручність та ефективність персонального фінансового додатку, даючи користувачам повну картину їхнього фінансового стану та можливість керувати своїми коштами з одного центрального місця. Однак розробники додатків повинні приділяти належну увагу питанням безпеки та конфіденційності, щоб гарантувати захист конфіденційних даних користувачів.

Аналітика та звітність. Аналітика та звітність дозволяє користувачам отримувати глибоке розуміння свого фінансового стану та тенденцій, що лежать в основі їхньої фінансової поведінки.

Ключовим аспектом є можливість генерувати детальні звіти, які охоплюють різні аспекти фінансового життя користувача, такі як доходи, витрати, бюджети та прогрес у досягненні фінансових цілей. Ці звіти можуть бути представлені у зручному для сприйняття форматі, наприклад, у вигляді таблиць, графіків і діаграм, що полегшує візуальний аналіз даних.

Додаток також повинен надавати користувачам можливість аналізувати тенденції витрат та доходів за певний період часу, наприклад, за місяць, квартал або рік. Це дозволяє виявляти закономірності та виявляти області, де можна оптимізувати фінансову поведінку. Порівняння поточних фінансових показників з минулими періодами також є корисною функцією, яка допомагає користувачам відстежувати свій прогрес і робити необхідні коригування.

Одним з найбільш потужних інструментів аналітики є можливість отримувати персоналізовані пропозиції щодо оптимізації фінансової поведінки на основі аналізу даних користувача. Наприклад, додаток може виявити, що користувач витрачає непропорційно велику суму на розваги, і запропонувати стратегії для скорочення цих витрат або перерозподілу коштів на інші пріоритетні цілі.

Аналітика та звітність забезпечують користувачів цінними інсайтами та рекомендаціями, які допомагають їм приймати більш обґрунтовані фінансові рішення. Ця функція перетворює персональний фінансовий додаток з простого інструменту відстеження фінансів на потужну аналітичну платформу, яка допомагає користувачам досягати своїх фінансових цілей та покращувати загальне управління фінансами.

Безпека та конфіденційність. Безпека та конфіденційність є критично важливими аспектами для будь-якого персонального фінансового додатку. Оскільки ці додатки оперують конфіденційними фінансовими даними користувачів, такими як номери банківських рахунків, дані про доходи та витрати, необхідно вжити суворих заходів для захисту цієї інформації від несанкціонованого доступу або витоку.

По-перше, додаток повинен використовувати найсучасніші методи шифрування даних під час передачі та зберігання. Це гарантує, що навіть у випадку перехоплення даних, вони будуть захищені та нечитабельні для зловмисників. Крім того, необхідно впровадити багаторівневу систему аутентифікації користувачів, яка може включати складні паролі, біометричні дані або токени безпеки.

Важливо також забезпечити дотримання відповідних нормативних вимог та стандартів безпеки, таких як Загальний регламент захисту даних та Стандарт безпеки даних індустрії платіжних карт . Ці вимоги встановлюють чіткі правила щодо обробки, зберігання та захисту конфіденційних даних, включаючи фінансову інформацію.

Для підвищення рівня безпеки персональний фінансовий додаток може надавати користувачам можливість встановлювати додаткові заходи захисту, такі як двофакторна аутентифікація або обмеження доступу до певних функцій або даних на основі ролей або пристроїв. Ці додаткові заходи безпеки дозволяють користувачам адаптувати рівень захисту відповідно до їхніх індивідуальних потреб та вподобань.

Необхідно пам’ятати, що витік або зловмисне використання конфіденційних фінансових даних може мати серйозні наслідки для користувачів, включаючи фінансові втрати, шахрайство та порушення конфіденційності. Тому розробники персональних фінансових додатків повинні приділяти найвищу увагу питанням безпеки та конфіденційності на всіх етапах розробки та експлуатації додатку.

Багатоплатформна підтримка та синхронізація. Багатоплатформна підтримка та безперебійна синхронізація даних є ключовими вимогами для сучасних персональних фінансових додатків. Користувачі очікують мати можливість доступу до своїх фінансових даних та керування ними з різних пристроїв та платформ, незалежно від їхнього місцезнаходження.

Для задоволення цих вимог персональний фінансовий додаток повинен бути доступним як для мобільних платформ iOS та Android, так і у вигляді веб-версії. Наявність мобільних додатків забезпечує користувачам зручність управління своїми фінансами в будь-якому місці та в будь-який час, використовуючи свої смартфони або планшети. Водночас веб-версія додатку дозволяє їм отримувати доступ до своїх даних з будь-якого пристрою, підключеного до Інтернету, включаючи настільні комп’ютери чи ноутбуки.

Крім того, персональний фінансовий додаток має забезпечувати безперебійну синхронізацію даних між усіма платформами в режимі реального часу. Це означає, що будь-які зміни, внесені користувачем у свої фінансові дані на одному пристрої, автоматично відображатимуться на всіх інших підключених пристроях. Така функціональність гарантує, що користувачі завжди матимуть доступ до актуальної та узгодженої інформації про свій фінансовий стан, незалежно від того, з якого пристрою вони працюють.

Синхронізація даних також полегшує перехід між різними платформами, дозволяючи користувачам розпочинати роботу на одному пристрої та продовжувати її на іншому без втрати даних або необхідності повторного введення інформації.

Багатоплатформна підтримка та безперервна синхронізація даних є важливими факторами, що сприяють задоволеності користувачів та зручності використання персонального фінансового додатку. Вони забезпечують гнучкість та мобільність, дозволяючи користувачам управляти своїми фінансами в будь-якому місці та з будь-якого пристрою, зберігаючи при цьому цілісність та актуальність своїх даних.

2.2 Вибір архітектурного стилю додатка

Архітектура розробки програмного забезпечення є ключовим компонентом у процесі створення додатків. Вона визначає загальну структуру системи, розподіляє функціональність між різними компонентами та описує принципи їх взаємодії. Іншими словами, архітектура встановлює чіткі правила та підходи, за якими різні частини програми виконують свої завдання та обмінюються даними.

Ретельно спроєктована архітектура гарантує, що програмне забезпечення буде виконувати свої передбачені функції та відповідати початковим вимогам, визначеним на етапі планування розробки. Вона допомагає зменшити складність розуміння системи шляхом чіткого розмежування обов’язків та створення зрозумілої ієрархічної структури [8 –12].

Архітектура та дизайн програмного забезпечення забезпечують організовану основу, яка полегшує роботу програмістів. Якість архітектури безпосередньо впливає на те, наскільки зручним буде подальше обслуговування, модифікація, розширення та підтримка програмного забезпечення протягом його життєвого циклу.

Монолітна архітектура – це підхід до розробки програмного забезпечення, за яким вся система створюється як єдине ціле, без чіткого розділення на окремі компоненти чи служби. Система являє собою єдиний, цілісний, нерозривний блок коду, схема архітектури представлена на рис. 2.1.

Переваги монолітної архітектури:

– Простота розробки та розгортання. Оскільки вся система розробляється як єдине ціле, без необхідності встановлювати та налаштовувати взаємодію між окремими компонентами, процес розробки та розгортання стає більш прямолінійним і зрозумілим. Розробники можуть зосередитися на створенні функціональності, не витрачаючи зусиль на інтеграцію різних частин системи.

– Швидкий розвиток на ранніх стадіях. Відсутність потреби в інтеграції окремих компонентів дозволяє прискорити розвиток на ранніх стадіях проекту. Команда розробників може швидко реалізувати основні функції додатку, не витрачаючи час на налагодження взаємодії між різними модулями.

– Легкість тестування. Оскільки всі компоненти знаходяться в одному місці та тісно пов’язані один з одним, процес тестування стає більш прямолінійним і ефективним. Розробники можуть легко відтворювати різні сценарії використання та перевіряти взаємодію між частинами додатку.

– Відсутність віддалених викликів. Відсутність необхідності у віддалених викликах між компонентами може покращити загальну продуктивність системи. Оскільки всі частини додатку знаходяться в одному місці, немає потреби в додаткових мережевих з’єднаннях чи віддалених викликах, що може зменшити накладні витрати та прискорити обробку даних.

Недоліки монолітної архітектури:

– Обмежена масштабованість. Оскільки вся система розгортається як єдине ціле, збільшення її потужності або розширення функціональності може бути ускладненим або навіть неможливим без повного перезапуску та переробки всього додатку.

– Складність модифікації та підтримки. Через тісну зв’язність компонентів, внесення змін або виправлення помилок у одній частині системи може вплинути на інші частини, ускладнюючи підтримку та модифікацію.

– Монолітна збійка. У разі, якщо один з компонентів системи виходить з ладу, це може призвести до збою всього додатку, оскільки всі частини тісно пов’язані між собою.

– Відсутність модульності та ізоляції. Неможливість чітко відокремити різні функціональні блоки ускладнює їх повторне використання в інших проектах та розподілену розробку між різними командами.

– Складність горизонтального масштабування. Оскільки вся функціональність зосереджена в одному місці, масштабування може бути дуже складним або навіть неможливим без значних архітектурних змін.

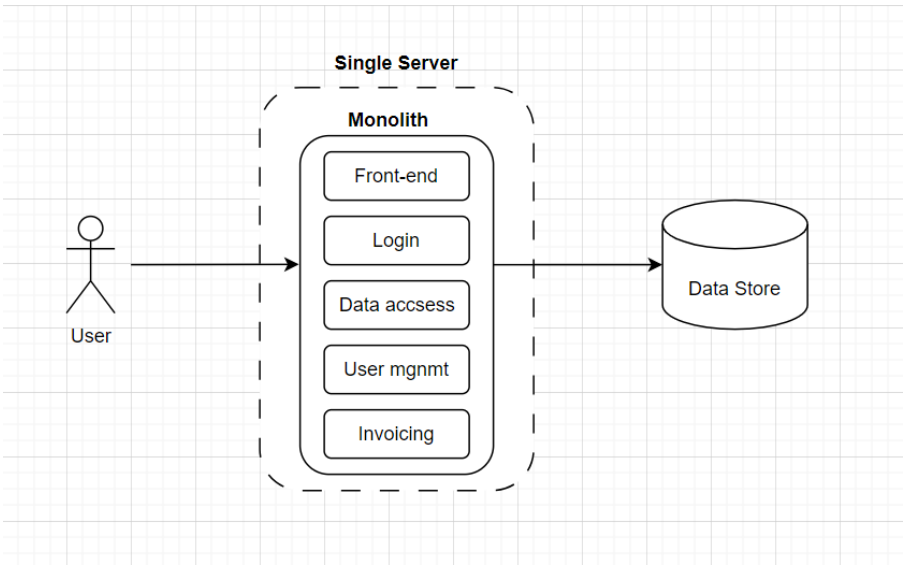

Рисунок 2.1 – Демонстрація монолітної архітектури

Клієнт-серверна архітектура – це архітектурний стиль, при якому програмне забезпечення розділяється на дві окремі частини: клієнтську та серверну. Клієнт і сервер взаємодіють один з одним через мережу,

використовуючи певний протокол зв’язку, як представлено на рис. 2.2. Переваги клієнт-серверної архітектури:

– клієнт відповідає за відображення даних та взаємодію з користувачем, тоді як сервер обробляє основну бізнес-логіку та управляє даними. Такий поділ забезпечує зрозумілу структуру системи, полегшуючи її розробку, підтримку та масштабування.

– зберігання та обробка даних централізовано на сервері забезпечує кращу безпеку, цілісність та керованість даними. Це дозволяє уникнути дублювання даних та спрощує їх резервне копіювання та відновлення.

– якщо потрібно збільшити потужність обробки даних, можна додати більше серверів, а якщо необхідно підтримати більше користувачів, можна додати більше клієнтських додатків. Така гнучкість дозволяє легко адаптувати систему до змінних вимог та навантаження.

– клієнтська частина може бути реалізована для веббраузерів, мобільних пристроїв, настільних комп’ютерів та інших платформ, тоді як серверна частина залишається незмінною. Це забезпечує кросплатформність та уніфікований доступ до функціоналу системи.

Недоліки клієнт-серверної архітектури:

– налагодження комунікації між клієнтом та сервером через мережу може бути складним завданням, особливо якщо мережа нестабільна або має високу затримку.

– система повністю залежить від наявності та якості мережевого з’єднання, що може вплинути на продуктивність та доступність.

– передача даних між клієнтом та сервером через мережу створює потенційні ризики для безпеки та конфіденційності даних.

– клієнтська частина часто має обмежену функціональність і покладається на сервер для виконання складних операцій.

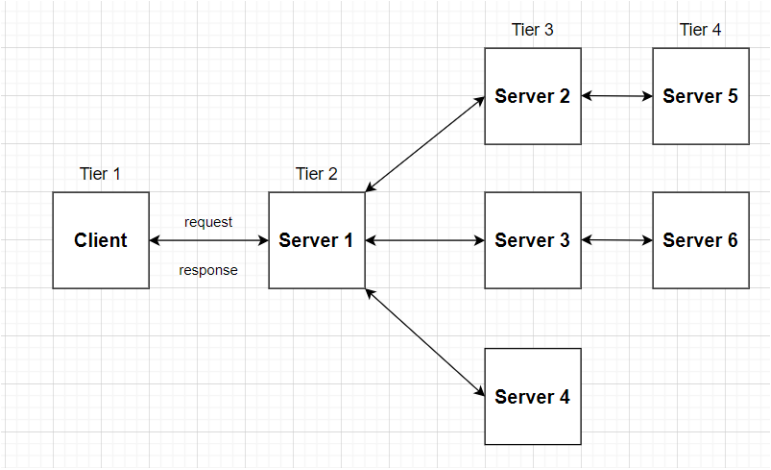

Рисунок 2.2 – Демонстрація клієнт-серверної архітектури

Багаторівнева архітектура (multi-layer architecture), також відома як шарувата архітектура, є підходом до структурування програмного забезпечення, в якому функціональність системи розподіляється між кількома окремими рівнями або шарами. Найбільш поширена реалізація багаторівневої архітектури передбачає розподіл системи на три ключові шари: презентаційний рівень, прикладний рівень та рівень доступу до даних. Діаграмну ілюстрацію представлену на рис. 2.3.

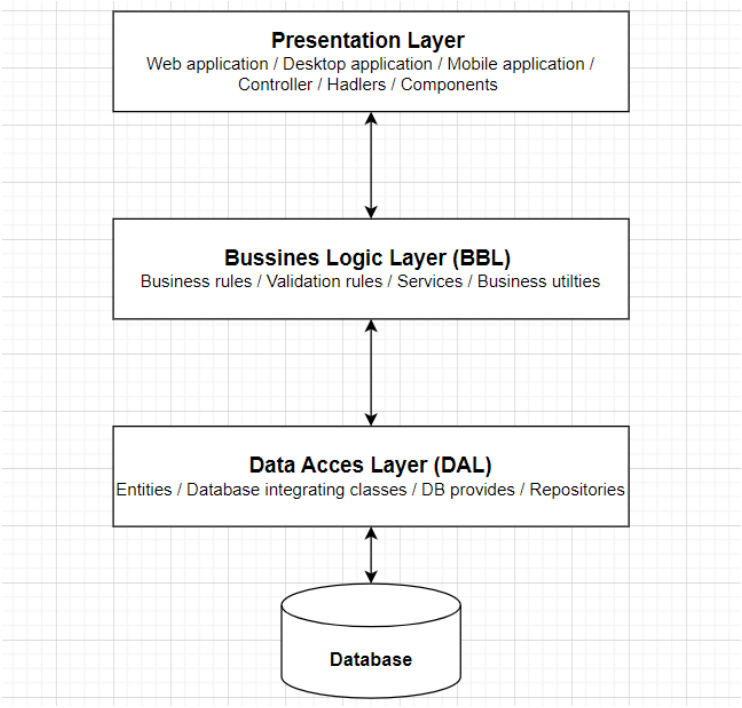

Рисунок 2.3 – Діаграмна ілюстрація трирівневої архітектури програмного забезпечення

Рівень представлення (Presentation Layer). Презентаційний рівень відповідає за забезпечення взаємодії з користувачем через графічний інтерфейс. Його основна функція - візуалізувати програму та дозволити користувачеві взаємодіяти з нею.

Цей рівень складається з елементів інтерфейсу, таких як вікна, панелі, кнопки та інші віджети для відображення даних та обробки введених користувачем даних. Він також містить контролери, які керують взаємодією користувачів з інтерфейсом.

Презентаційний рівень відділений від бізнес-логіки системи. Це дозволяє створювати різні графічні інтерфейси для різних платформ (мобільні, веб, десктопні додатки), використовуючи спільний рівень бізнес-логіки. Такий підхід робить кодову базу більш модульною та масштабованою. Наприклад, мобільний, веб- та настільний додатки можуть мати різні інтерфейси, але спільний функціональний рівень.

*Рівень бізнес - логіки (**Business Logic Layer**).* Рівень бізнес - логіки, також відомий як логічний рівень або середній рівень, є ключовим компонентом системи. Це базовий функціональний модуль, де відбувається обробка даних та реалізується основна бізнес-логіка.

На цьому рівні дані, зібрані з презентаційного рівня (інтерфейсу користувача), обробляються відповідно до набору бізнес-правил та вимог. Бізнес-логіка визначає, як система повинна інтерпретувати та маніпулювати даними. Рівень додатків також може взаємодіяти з рівнем даних, запитуючи додаткову інформацію або здійснюючи операції додавання, видалення чи модифікації даних.

Рівень додатків зазвичай реалізується з використанням мов програмування високого рівня, таких як Python, Java, Perl, PHP або Ruby. Для взаємодії з рівнем даних він використовує спеціальні виклики API, які абстрагують деталі доступу до даних.

Цей рівень дуже важливий, оскільки він інкапсулює основну бізнес-логіку додатку. Він діє як проміжна ланка між презентаційним рівнем та рівнем даних,

забезпечуючи коректну обробку даних та виконання необхідних операцій згідно з вимогами бізнесу.

Рівень доступу до даних (Data Access Layer). Рівень доступу до даних (Data Access Layer, DAL) – це архітектурний компонент, який виступає проміжною ланкою між бізнес-логікою додатку та сховищем даних. Він відповідає за управління збереженням та отриманням даних, необхідних для функціонування програми. DAL забезпечує рівень абстракції, який дозволяє бізнес-логіці взаємодіяти з системою зберігання даних, не знаючи деталей її внутрішньої реалізації. Цей рівень приховує від вищих шарів специфіку роботи з конкретними базами даних, файловими сховищами чи іншими джерелами даних, надаючи уніфікований інтерфейс для доступу та маніпуляцій з даними.

Переваги багаторівневої архітектури:

– модульність значно полегшує процеси розробки, тестування, підтримки та модифікації різних компонентів додатку. Розробники можуть зосередитися на конкретному рівні, не турбуючись про деталі реалізації інших частин системи;

– розподіл обов’язків сприяє кращому розподілу робочого навантаження між командами розробників і полегшує управління проектом загалом. Команди можуть спеціалізуватися на певних рівнях, що підвищує продуктивність і ефективність роботи. Приклад такого розмежування наведено на рис. 2.4;

– повторне використання коду економить час і зусилля розробників, оскільки їм не потрібно писати код для однакових функцій з нуля;

– гнучкість та розширюваність визначають, що зміни в одному рівні не впливають на інші рівні, додавання нової функціональності або модифікація існуючої стає простішим завданням. Це забезпечує кращу адаптивність системи до змінних вимог та потреб бізнесу;

– полегшене тестування, оскільки спрощується процес виявлення та усунення помилок у зв’язку з можливостями локалізації проблеми на конкретному рівні, не впливаючи на решту системи.

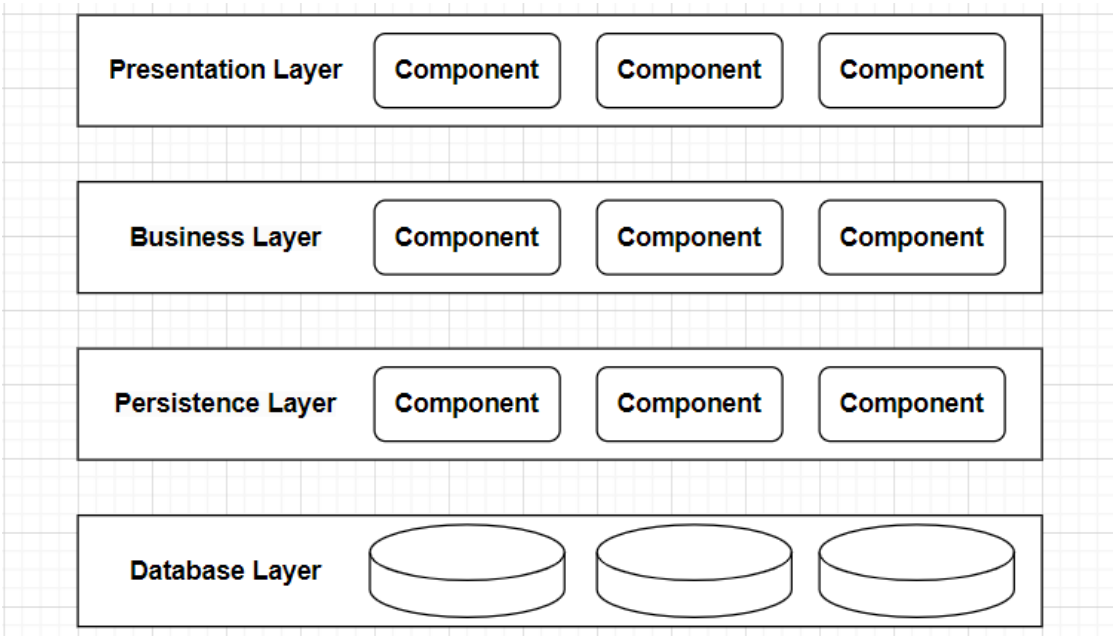

Рисунок 2.4 – Демонстрація багаторівневої архітектури

Недоліки багаторівневої архітектури:

– збільшення часу відгуку: кожен додатковий рівень архітектури вимагає додаткових обчислень та передачі даних між рівнями. Це призводить до затримок у часі відгуку системи, оскільки запит має пройти крізь кілька рівнів, перш ніж буде оброблений і повернутий результат. Чим більше рівнів, тим більше затримок і, відповідно, довший час відгуку;

– навантаження на мережу: у багаторівневій архітектурі різні рівні часто розміщуються на різних серверах або вузлах, що вимагає передачі даних через мережу. Ця передача даних може створювати додаткове навантаження на мережеві ресурси та стати вузьким місцем, особливо якщо мережа має обмежену пропускну здатність або високу затримку;

– збільшення складності: кожен додатковий рівень архітектури вносить додаткову складність у систему. Це може ускладнити розробку, тестування, налагодження та обслуговування застосунку. Крім того, збільшується ймовірність виникнення помилок або несумісностей між рівнями, що може негативно вплинути на продуктивність;

– додаткові накладні витрати: для забезпечення комунікації між рівнями потрібні додаткові механізми, такі як міжпроцесна взаємодія, віддалені

виклики процедур (RPC) або обмін повідомленнями. Ці механізми вносять додаткові накладні витрати, що можуть зменшити загальну продуктивність системи;

– складність масштабування: в багаторівневій архітектурі різні рівні можуть мати різні вимоги до ресурсів та масштабування. Це може ускладнити процес масштабування окремих рівнів відповідно до потреб застосунку, особливо якщо рівні сильно пов’язані між собою.

Мікросервісна архітектура – це підхід до розробки програмного забезпечення, в якому додаток розбивається на невеликі, незалежні та слабко зв’язані мікросервіси, які предсталено на рис. 2.5. Кожен мікросервіс виконує єдину бізнес-функцію, має власний цикл розробки та може бути розгорнутий та масштабований незалежно.

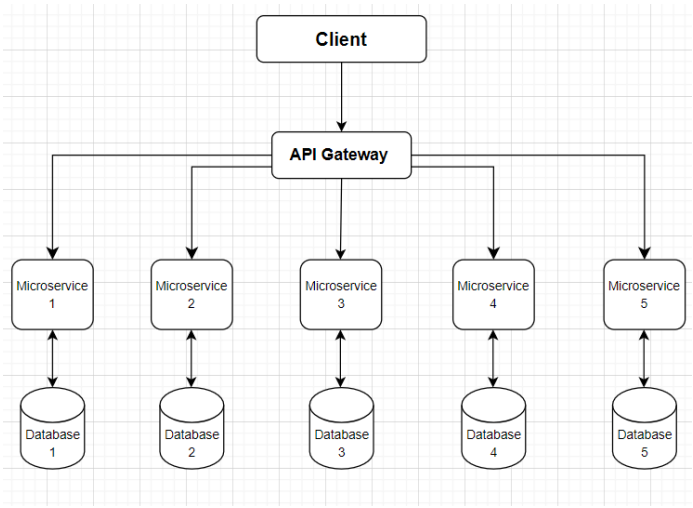

Рисунок 2.5 – Демонстрація мікросервісної архітектури

Переваги мікросервісної архітектури:

– кожен мікросервіс розробляється, тестується та розгортається абсолютно незалежно від інших, що дозволяє різним командам розробників працювати паралельно над різними частинами системи. Це значно прискорює процес розробки та забезпечує більшу гнучкість у впровадженні змін та нових функцій.

– окремі мікросервіси можна легко масштабувати відповідно до поточних потреб, розгортаючи додаткові екземпляри або збільшуючи

потужність ресурсів для тих мікросервісів, які мають високе навантаження. Це відбувається без жодного впливу на решту системи, що забезпечує оптимальне використання ресурсів та високу ефективність.

– якщо один з мікросервісів виходить з ладу через помилку або перевантаження, це не призводить до збою всієї системи. Інші мікросервіси продовжують працювати, забезпечуючи доступність решти функціональності додатку. Це підвищує надійність та стабільність системи в цілому.

– оскільки кожен мікросервіс є незалежним, його можна розробляти з використанням найбільш підходящих технологій та фреймворків, не турбуючись про вплив на інші частини додатку.

– кожен мікросервіс можна тестувати окремо, локалізуючи та виправляючи помилки в межах конкретного компонента, без необхідності тестувати всю систему цілком.

Недоліки мікросервісної архітектури:

– на відміну від монолітних додатків, розробка та управління розподіленими системами, якими є мікросервіси, вимагають від розробників додаткових знань та досвіду. Це може створювати певні труднощі та потребувати додаткових ресурсів на навчання та підготовку команди.

– оскільки дані розподілені між багатьма мікросервісами, виникає необхідність ретельно керувати узгодженістю цих даних та уникати їх дублювання. Це може спричинити додаткову складність та вимагати додаткових зусиль для забезпечення цілісності та узгодженості даних у системі.

– збільшена кількість міжсервісних викликів, яка є невід’ємною частиною мікросервісної архітектури, може призвести до підвищеного навантаження на мережу та затримок у роботі системи. Це може негативно позначитися на продуктивності та швидкодії додатку, особливо в умовах високого навантаження.

– на відміну від монолітних додатків, де всі компоненти знаходяться в одному місці, у мікросервісній архітектурі необхідно моніторити та відстежувати стан кожного окремого мікросервісу. Це може бути складним

завданням і вимагати використання спеціальних інструментів та практик.

– розгортання та управління численними мікросервісами може бути більш складним порівняно з монолітними додатками. Це пов’язано з необхідністю розгортати та підтримувати кожен мікросервіс окремо, що може вимагати додаткових зусиль та ресурсів.

Незважаючи на певні виклики та потенційні недоліки, притаманні мікросервісній архітектурі, вона залишається привабливим рішенням для багатьох проектів, особливо масштабних та складних систем. Важливо об’єктивно оцінити рівень підготовки команди розробників та її готовність ефективно працювати з мікросервісною архітектурою, яка вимагає додаткових навичок та досвіду. Лише після ґрунтовного аналізу всіх факторів можна приймати виважене рішення щодо доцільності впровадження мікросервісної архітектури в рамках конкретного проєкту.

Подійно-орієнтована архітектура – це архітектурний стиль, в якому компоненти системи взаємодіють шляхом створення, виявлення, передачі та споживання подій. Події є повідомленнями, які описують факт, що стався в системі, і можуть бути опрацьовані різними компонентами асинхронно, що продемонстровано на рис. 2.6.

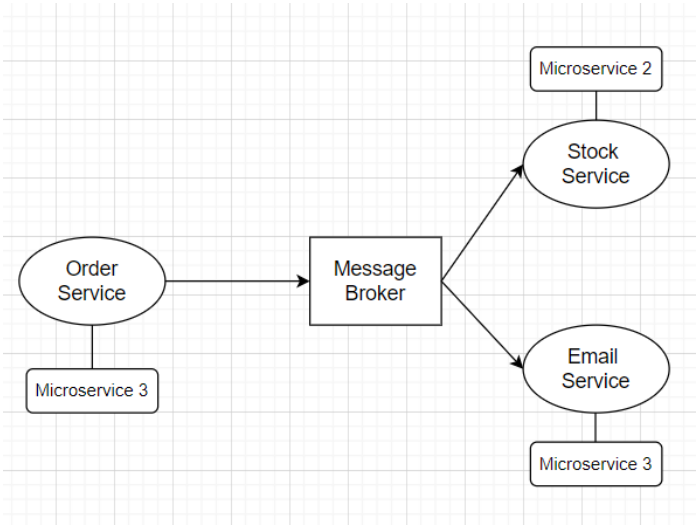

Рисунок 2.6 – Демонстрація подійно-орієнтованої архітектури

Переваги подійно-орієнтованої архітектури:

– компоненти є слабко зв’язаними, що дозволяє їм розвиватися та

розгортатися незалежно, не впливаючи на решту системи. Це забезпечує гнучкість та полегшує внесення змін і масштабування.

– асинхронна обробка дозволяє компонентам працювати паралельно, не блокуючи один одного, що підвищує продуктивність та ефективність системи загалом.

– різні компоненти можуть підписуватися на одну й ту саму подію та опрацьовувати її відповідно до своїх потреб. Це забезпечує гнучкість та можливість повторного використання функціоналу.

– нові компоненти можуть легко підписуватися на існуючі події без необхідності змінювати інші частини системи, що забезпечує відкритість до змін та розширюваність.

– нові компоненти можуть підписуватися на існуючі події без необхідності змінювати інші частини системи.

Недоліки подієво-орієнтованої архітектури:

– оскільки компоненти опрацьовують події асинхронно, підтримувати цілісний стан системи може бути досить складно. Це вимагає ретельного планування та проектування для забезпечення узгодженості даних та стану.

– відстеження шляху події від джерела до всіх підписників може бути нетривіальним завданням, що вимагає спеціальних інструментів та практик.

– правильний вибір черги та її конфігурації (наприклад, стійкості, черговості, фільтрації) є дуже важливим для забезпечення надійної та ефективної обробки подій.

– збільшене навантаження на мережу може призвести до затримок та погіршення продуктивності, якщо мережева інфраструктура не була належним чином спроектована та налаштована.

– необхідність додаткової інфраструктури може збільшити складність розгортання та управління системою, а також вимагати додаткових ресурсів та витрат.

Незважаючи на певні виклики та потенційні недоліки, подійно-орієнтована архітектура залишається привабливим архітектурним рішенням для багатьох

розподілених та масштабованих систем. Її основні переваги, такі як гнучкість, масштабованість та можливість незалежної розробки компонентів, роблять її надзвичайно доцільною для застосування у сучасних розподілених середовищах.

Проте, при впровадженні цієї архітектури необхідно ретельно оцінити потенційні ризики та складнощі, з якими може зіткнутися команда розробників. Важливо застосовувати належні практики та інструменти для мінімізації впливу виявлених недоліків та забезпечення успішної реалізації архітектурного рішення, яке відповідатиме вимогам проекту та забезпечуватиме стабільну та ефективну роботу системи.

Після ретельного вивчення різноманітних архітектурних підходів до проектування програмних систем, було обрано багаторівневу архітектуру. Такий вибір зумовлений низкою істотних переваг, які вона надає для створення якісного та масштабованого фінансового додатку. Перш за все, багаторівнева архітектура забезпечує належний розподіл функцій та відповідальностей між окремими компонентами системи. Кожен рівень має чітко визначену роль та завдання, що спрощує процеси розробки, тестування та подальшого обслуговування програмного забезпечення. Це сприяє підвищенню загальної модульності та гнучкості створеного додатку.

Впровадження такого архітектурного підходу гарантує необхідну масштабованість рішення. Окремі рівні можуть бути незалежно розгорнуті та масштабовані відповідно до потреб користувачів та навантаження на систему. Це дає змогу ефективно керувати ресурсами та забезпечувати належний рівень продуктивності й стабільності роботи фінансового додатку.

Одною з найважливіших переваг обраної архітектури є також покращена безпека системи. Завдяки чіткому розподілу на рівні, критично важливі компоненти, такі як рівень доступу до даних, можуть бути захищені від прямої взаємодії з клієнтським рівнем, мінімізуючи ризики несанкціонованого доступу або зловмисного впливу.

Отже, багаторівнева архітектура дозволяє створити надійне, масштабоване та безпечне програмне рішення для управління персональними фінансами. Чітке

розмежування функцій, гнучкість, масштабованість та підвищений рівень безпеки роблять цей архітектурний підхід оптимальним вибором для розробки якісних сучасних фінансових додатків.

2.3. Огляд обраної архітектури додатку

У галузі розробки програмного забезпечення багаторівнева архітектура виділяється як одна з найбільш популярних та широко впроваджених архітектурних моделей. Ця архітектурна парадигма дозволяє ефективно боротися зі зростаючою складністю сучасних програмних систем, забезпечуючи розподіл функціоналу на окремі рівні, кожен з яких відповідає за певний аспект або послугу. Така модульність не лише полегшує процес розробки та супроводу програмного забезпечення, але й надає необхідну гнучкість, що є критично важливим в епоху домінування DevOps та інших гнучких методологій розробки.

Багаторівнева архітектура, також відома як n-рівнева архітектура, складається з ієрархії рівнів, де кожен рівень виконує специфічні функції та взаємодіє з сусідніми рівнями за допомогою чітко визначених інтерфейсів. Завдяки такому розподілу відповідальностей, зміни або удосконалення можна вносити на конкретному рівні, не впливаючи на інші рівні та не порушуючи цілісність всієї архітектури. Це значно спрощує процес розробки, тестування та супроводу програмного забезпечення, підвищуючи ефективність та скорочуючи витрати.

Багаторівнева архітектура програмного забезпечення є потужною концепцією, яка забезпечує структурований та модульний підхід до розробки складних систем. Перш ніж розглядати її складніші варіації, доцільно почати з найпростішої та найпоширенішої форми – трирівневої архітектури. Ця архітектура є фундаментальною основою для більшості багаторівневих систем і складається з трьох основних компонентів (рис. 2.7):

– Рівень презентації – найвищий рівень архітектури, який забезпечує взаємодію з кінцевим користувачем та відображення даних. Він відповідає за представлення вмісту через графічний інтерфейс користувача (GUI) та може

бути доступним з різноманітних клієнтських пристроїв, таких як настільні комп’ютери, ноутбуки, планшети, мобільні телефони чи тонкі клієнти. Для відображення вмісту цей рівень повинен взаємодіяти з нижчими рівнями архітектури, отримуючи необхідні дані та виконуючи відповідну логіку.

– Прикладний рівень – це проміжний рівень, на якому реалізується основна бізнес-логіка системи. Він містить набір правил та алгоритмів, які визначають поведінку програми відповідно до вимог організації. Компоненти цього рівня зазвичай працюють на одному або кількох серверах додатків і забезпечують обробку даних, виконання операцій та координацію взаємодії між рівнями презентації та даних.

– Рівень даних – це найнижчий рівень трирівневої архітектури, який відповідає за зберігання, управління та отримання даних програми. Дані зазвичай зберігаються на серверах баз даних, файлових серверах або інших сховищах, які підтримують логіку доступу та забезпечують необхідні процедури для гарантування безпеки та цілісності даних. Цей рівень надає прикладному рівневі API для взаємодії з даними, забезпечуючи прозорість операцій без розкриття внутрішніх механізмів зберігання та пошуку. Завдяки цьому зміни або оновлення на рівні даних не впливають безпосередньо на прикладний рівень.

Незважаючи на введення додаткових рівнів, основа багаторівневої архітектури програмного забезпечення залишається незмінною. Вона все одно включає в себе презентаційний рівень, який відповідає за взаємодію з користувачем, та рівень даних, який забезпечує зберігання та доступ до даних системи. Проте, на відміну від трирівневої архітектури, багаторівнева архітектура розділяє та розширює функціональність прикладного рівня, розподіляючи його на додаткові підрівні або служби. Ці додаткові компоненти на прикладному рівні відповідають за різні аспекти бізнес-логіки та функціональності програмного забезпечення.

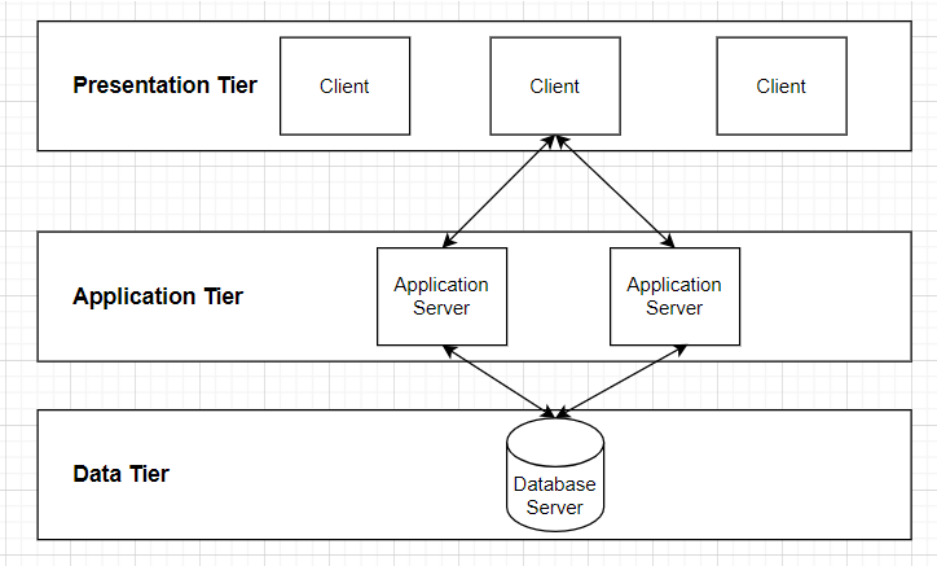

Рисунок 2.7. – Робота простої багаторівневої архітектури з 3 рівнями

Такий підхід дозволяє створювати більш масштабовані та гнучкі системи, забезпечуючи додаткові виміри функціональності та можливості для розширення. Розподіл коду та функцій між різними рівнями може відрізнятися залежно від конкретного архітектурного рішення, однак базова концепція розділення відповідальностей та модульності залишається незмінною.

Важливо зазначити, що збільшення кількості рівнів може призвести до додаткових накладних витрат та складності в управлінні системою. Тому необхідно ретельно зважувати переваги та компроміси перед впровадженням додаткових рівнів у багаторівневу архітектуру.

2.3. Огляд обраної архітектури додатку

Представлена на рис. 2.8 діаграма ілюструє багаторівневу архітектуру програмного забезпечення, спеціалізованого на управлінні персональними фінансами. Ця архітектурна модель демонструє стратифікований підхід до організації компонентів системи, забезпечуючи чітке розмежування функціональних областей та сприяючи високій модульності. Розглянемо кожен структурний елемент цієї архітектури, починаючи з фундаментального рівня і просуваючись до прикладного:

– Рівень джерел даних (Data Sources Layer) є базовим рівнем архітектури, який репрезентує фізичні репозиторії даних, такі як реляційні бази даних (РСКБД) та файлові системи. Він виступає у ролі резервуару для зберігання усіх персистентних даних додатку, включаючи, але не обмежуючись, інформацією про фінансові транзакції, стан рахунків користувачів та історію бюджетування.

– Рівень доступу до даних (Data Access Layer). Цей рівень відповідає за абстрагування механізмів взаємодії з джерелами даних. Він надає вищим рівням уніфікований програмний інтерфейс для виконання операцій створення, читання, оновлення та видалення (CRUD) даних, тим самим інкапсулюючи складності, пов’язані зі специфікою різних систем зберігання.

– Рівень бізнес-логіки (Business Logic Layer). На цьому рівні концентруються основні алгоритми та бізнес-правила, які є специфічними для домену персональних фінансів. Сюди входять модулі, відповідальні за управління рахунками, формування та моніторинг бюджетів, генерацію аналітичних звітів та прогнозування фінансових трендів;

– Рівень фасаду додатку (Application Facade Layer). Фасад виступає посередником між сервісним шаром та логікою бізнесу. Він надає спрощений та уніфікований інтерфейс для взаємодії з комплексними підсистемами бізнес-логіки, тим самим зменшуючи зв’язність між компонентами та підвищуючи загальну модульність системи;

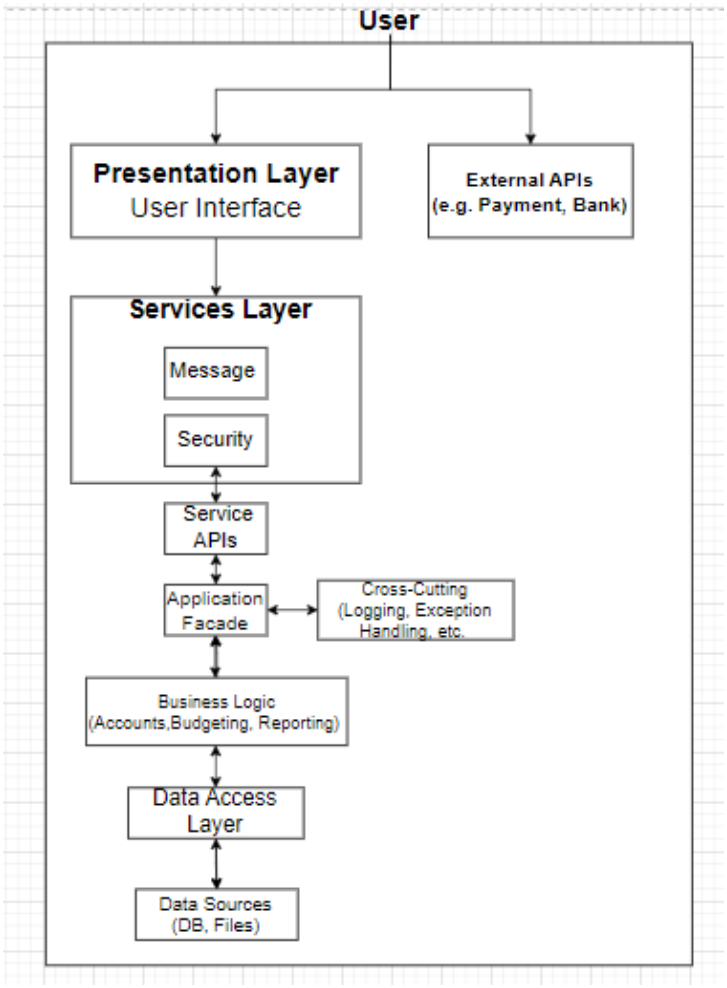

Рисунок 2.8 – Багатошарова архітектура фінансового додатку

– Рівень сервісних API (Service APIs Layer). Цей прошарок розкриває програмні інтерфейси (API), які дозволяють вищим рівням, зокрема презентаційному, взаємодіяти з функціональністю, наданою фасадом додатку та бізнес-логікою. Ці API можуть бути реалізовані як REST-сумісні вебсервіси або gRPC-ендпоінти;

– Рівень безпеки (Security Layer) є важливим для забезпечення конфіденційності, цілісності та доступності даних користувачів. Він інкорпорує механізми автентифікації, авторизації та шифрування для захисту персональної фінансової інформації як у стані спокою, так і під час передачі;

– Підрівень повідомлень (Message Sublayer) відповідає за оркестрацію асинхронної комунікації між різними компонентами системи. Він може використовувати такі паттерни, як черги повідомлень (message queues) або шини

подій (event buses), для підвищення стійкості та масштабованості системи;

– Рівень сервісів (Services Layer) об’єднує функціональність підрівнів безпеки, повідомлень та сервісних APIs. Він надає вищим рівням доступ до набору сервісів, які інкапсулюють основні бізнес-операції, тим самим сприяючи повторному використанню коду та спрощуючи підтримку;

– Презентаційний рівень (Presentation Layer), як найвищий рівень архітектури, він безпосередньо взаємодіє з кінцевими користувачами. Він складається з компонентів користувацького інтерфейсу (UI Components), які відповідають за візуалізацію фінансових даних та обробку користувацьких інтеракцій. Цей рівень може бути представлений вебінтерфейсом, мобільним додатком або десктопною аплікацією;

– Зовнішні API (External APIs). Цей компонент уможливлює інтеграцію зі сторонніми фінансовими сервісами, такими як платіжні шлюзи, банківські API або аналітичні платформи. Він розширює функціональні можливості додатку, наприклад, дозволяючи автоматичне оновлення балансів рахунків або використання машинного навчання для категоризації витрат;

– Наскрізні аспекти (Cross-Cutting Concerns). Цей компонент репрезентує сервіси та інфраструктуру, які використовуються усіма рівнями архітектури. Сюди належать системи логування, моніторингу продуктивності, обробки виключних ситуацій та інші утиліти, що забезпечують надійність, безпеку та підтримуваність всієї системи.

Операційний потік архітектури:

– користувачі ініціюють фінансові операції через презентаційний рівень;

– запити від компонентів UI транслюються на рівень сервісів;

– рівень безпеки проводить верифікацію запитів, застосовуючи політики доступу;

– сервісні API обробляють валідовані запити та делегують їх фасаду додатку;

– фасад координує взаємодії з бізнес-логікою для виконання

фінансових алгоритмів;

– бізнес-логіка використовує рівень доступу до даних для взаємодії з репозиторіями;

– результати операцій поширюються ієрархічно назад до інтерфейсу користувача;

– зовнішні API можуть бути задіяні на різних етапах для збагачення функціональності;

– наскрізні аспекти, як-от журналювання, функціонують трансверсально на всіх рівнях.

Описана багаторівнева архітектура втілює принципи розділення відповідальностей, високого рівня абстракції, масштабованості та безпеки, що є важливими для додатку, який оперує конфіденційними фінансовими даними користувачів. Ця архітектурна парадигма сприяє гнучкості розробки, полегшує процеси тестування та підтримки, а також дозволяє здійснювати незалежне масштабування компонентів відповідно до мінливих вимог навантаження та бізнес-потреб.

РОЗДІЛ 3
ПРОТОТИПУВАННЯ ПЕРСОНАЛЬНОГО ФІНАНСОВОГО ДОДАТКА З УРАХУВАННЯМ АТРИБУТІВ ЯКОСТІ

3.1. Важливість прототипування під час розробки програмного забезпечення

Процес прототипування є невіддільним і надзвичайно важливим етапом у розробці програмного забезпечення. Він надає можливість перевірити та оцінити ідеї, концепції та пропозиції дизайну на ранніх стадіях створення системи, до того, як були залучені значні ресурси та зусилля для повномасштабної реалізації рішення [13].

Ця практика дозволяє виявити потенційні недоліки та проблемні аспекти проекту на початкових етапах, коли їх усунення не потребує надмірних витрат часу та коштів. Також прототипування забезпечує збір відгуків та думок від представників цільової аудиторії та інших зацікавлених сторін, що дає змогу адаптувати рішення відповідно до їхніх реальних потреб та очікувань.

Загалом, створення прототипів сприяє кращій комунікації, допомагає досягти спільного розуміння проекту серед усіх учасників розробки, а також дозволяє наочно візуалізувати майбутню поведінку та принципи взаємодії з програмним продуктом.

При розробці персональних фінансових додатків особливу увагу необхідно приділити забезпеченню ключових атрибутів якості, серед яких зручність використання, безпека, ефективність та масштабованість відіграють визначальну роль.

Забезпечення високого рівня зручності використання є критично важливим аспектом, оскільки додаток має бути інтуїтивно зрозумілим та легким у повсякденній експлуатації для широкого кола користувачів різного віку, досвіду та рівня технічних навичок. Інтерфейс повинен бути зручним, логічно структурованим та відповідати загальноприйнятим принципам дизайну користувацького інтерфейсу. Це дозволить користувачам швидко опанувати додаток та ефективно керувати своїми фінансами без надмірних ускладнень.

Не менш важливим є забезпечення належного рівня безпеки програмного рішення, що має першочергове значення через високу конфіденційність та делікатний характер фінансових даних користувачів, таких як номери банківських рахунків, транзакції, залишки на рахунках тощо. Необхідно впровадити надійні механізми шифрування, автентифікації та авторизації, а також дотримуватися відповідних стандартів та регламентів безпеки для захисту персональних даних від несанкціонованого доступу або витоку.

Ефективність системи також є важливим атрибутом якості фінансового додатку. Висока продуктивність та безперебійна робота забезпечать позитивний досвід користувача та уникнуть ситуацій, коли користувачу доведеться очікувати тривалого завантаження або оброблення даних. Це особливо важливо для мобільних додатків, де обмежені обчислювальні ресурси та з’єднання з мережею.

Масштабованість також є важливим атрибутом якості фінансового додатку. Система має бути спроектована таким чином, щоб ефективно справлятися із зростаючим навантаженням та збільшенням кількості користувачів без суттєвого погіршення продуктивності. Це забезпечить можливість розширення функціоналу, додавання нових можливостей та підтримки великої бази користувачів у майбутньому.

Таким чином, під час розробки персональних фінансових додатків необхідно ретельно спроектувати архітектуру та компоненти системи для гарантування високого рівня зручності використання, безпеки, ефективності та масштабованості – ключових атрибутів якості, що визначають успіх та довготривале задоволення користувачів.

3.2. Розробка прототипу фінансового застосунку

На сучасному етапі розвитку інформаційних технологій ключовим фактором успіху програмного забезпечення є врахування атрибутів якості, зокрема зручності використання (usability). У 2019 році було проведено опитування компанією UX Tools. Дизайнерам був наданий список інструментів

на вибір, перед ними була задача обрати найкращий сервіс для створення дизайну. За результатами опитування, представленими на рис. 3.1, найпопулярнішим інструментом для розробки інтерфейсів користувача (UI) серед дизайнерів був Sketch з часткою 59%. Однак, друга за популярністю програма Figma (29%) стрімко набирає обертів і має низку переваг, зокрема можливість спільної роботи в режимі реального часу, безкоштовну базову версію та кросплатформеність. Третє місце посів інструмент Adobe XD з відсотком розміром у 11.9%. На четвертому місці залишився Photoshop, отримавши 5.9% відсотків [14].

З огляду на важливість ітераційного процесу прототипування для створення якісного та зручного користувацького інтерфейсу, у цьому розділі буде описано методологію розробки прототипу персонального фінансового додатку з використанням інструменту Figma.

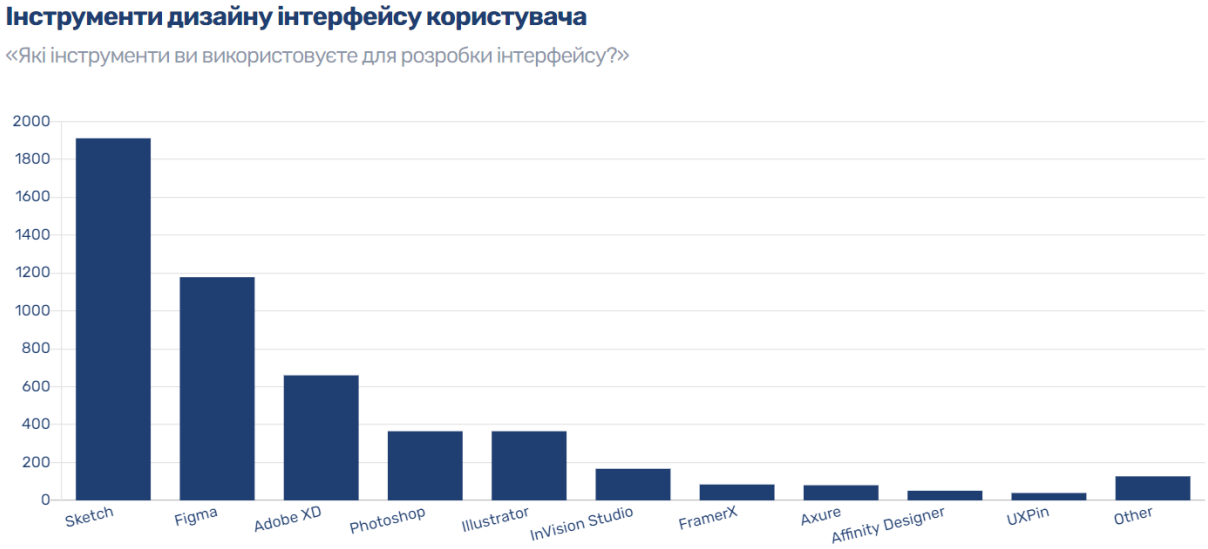

Рисунок 3.1 – Результати опитування UX Tools

3.2.1. Огляд обраного інструменту для прототипування

Figma – це потужний інструмент для створення прототипів та дизайну інтерфейсів користувача, який набуває все більшої популярності серед розробників та дизайнерів. Він пропонує багато переваг, які роблять його привабливим вибором для роботи над проектами будь-якого масштабу.

Починаючи роботу з Figma, перша річ, яку варто зазначити - це її хмарна природа. На відміну від традиційних настільних програм, Figma працює у веб-браузері, що забезпечує зручний доступ з будь-якого пристрою та можливість спільної роботи в режимі реального часу. Це особливо корисно для команд, які працюють віддалено або розподілені географічно [15, 16].

Після реєстрації облікового запису та входу в систему, які представлені на рис.3.2, ви потрапляєте до чистого робочого простору, зображеного на рис.3.3, готового для створення нових проектів. Інтерфейс Figma інтуїтивно зрозумілий і нагадує інші популярні програми для дизайну, такі як Adobe Illustrator або Sketch.

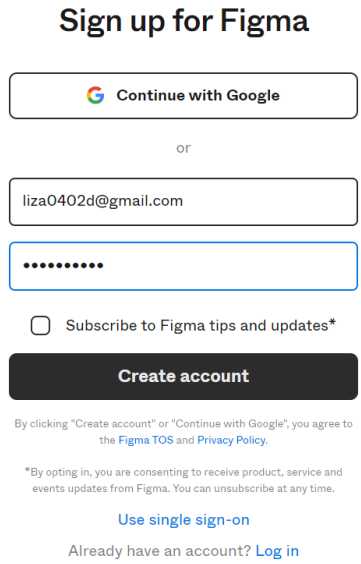

Рисунок 3.2 – Початкове вікно реєстрації у Figma

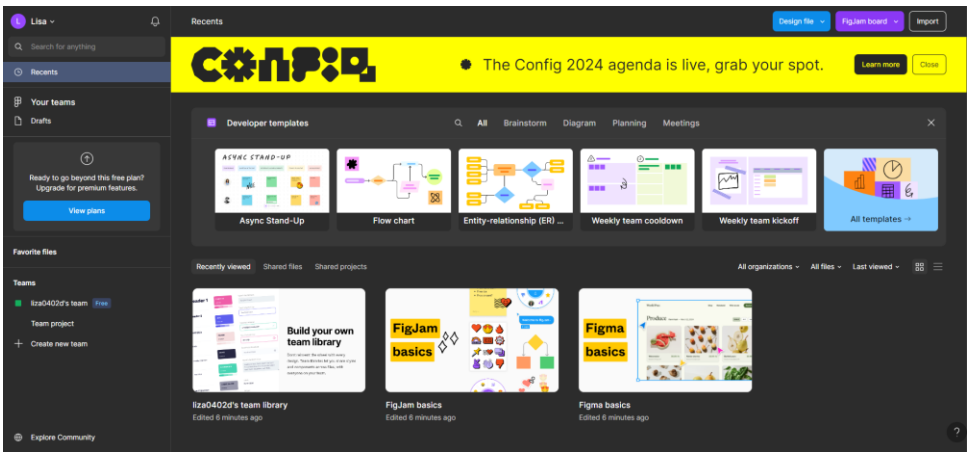

Рисунок 3.3 – Початковий екран робочої області Figma

Одним з найпотужніших інструментів Figma є векторний редактор. Він дозволяє створювати та маніпулювати різноманітними формами, кривими та об’єктами з безпрецедентною гнучкістю, частина з них представлена на рис. 3.4. Ви також можете імпортувати зображення та працювати з ними на тих самих полотнах.

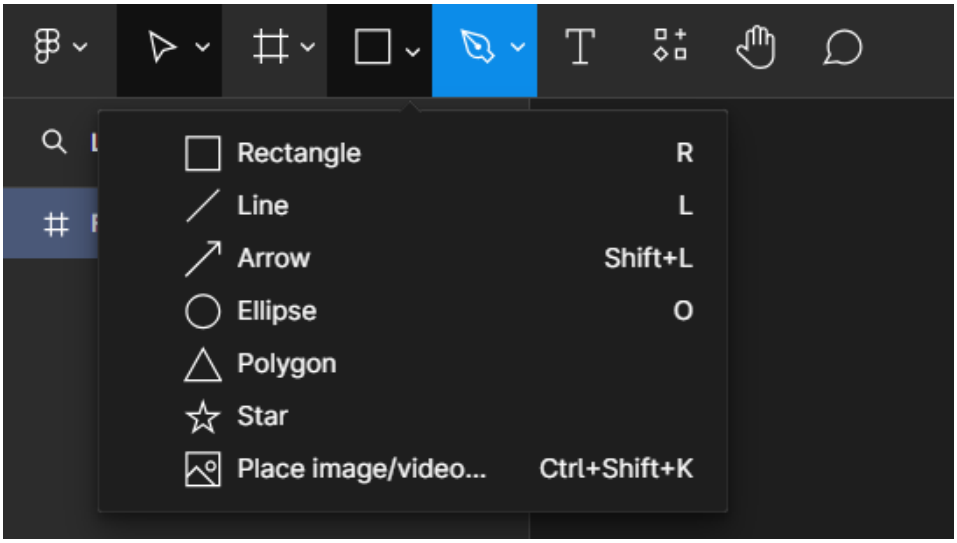

Рисунок 3.4 – Інструмент базових фігур Figma

Figma пропонує широкий вибір інструментів для створення макетів інтерфейсів користувача, включаючи бібліотеки компонентів, стилі, сітки та автоматичне макетування. Ці функції допомагають забезпечити послідовність дизайну та полегшують внесення змін у великі проекти.

Крім того, Figma має потужні можливості прототипування. Ви можете створювати інтерактивні прототипи, з’єднуючи екрани та визначаючи взаємодії, такі як натискання кнопок або перетягування елементів. Це дозволяє ефективно тестувати ваші дизайни та отримувати зворотний зв’язок від користувачів на ранніх етапах процесу розробки.

Одна з найбільших переваг Figma - це її відкритість для інтеграції з іншими інструментами та сервісами через потужну систему плагінів та API. Це дозволяє розширювати можливості Figma та адаптувати її до своїх конкретних потреб.

У підсумку, Figma є гнучким і потужним інструментом для створення прототипів та дизайну інтерфейсів користувача, який забезпечує безперешкодну співпрацю, багаті можливості та відкритість для інтеграції.

3.2.2. Розробка бази даних для фінансового додатку

Для забезпечення ефективного зберігання, обробки та керування фінансовими даними фінансовий додаток потребує надійної та продуктивної бази даних [17].

База даних – це сукупність взаємопов’язаних даних, організованих відповідно до схеми, яка описує характеристики цих даних та взаємозв’язки між їхніми різними складовими. Дані в базі даних зберігаються в логічний та структурований спосіб для забезпечення ефективного пошуку, вибірки, оновлення та керування даними. База даних управляється спеціальною програмою, яка називається системою управління базами даних (СУБД) [18].

Для створення бази даних розглядаються наступні цілі:

– гарантування цілісності та конфіденційності фінансових даних користувачів, включаючи рахунки, транзакції, інвестиції тощо;

– забезпечення швидкого доступу та ефективне виконання операцій читання/запису для належної продуктивності додатку;

– підтримка масштабованості для обробки великих обсягів даних та зростаючої кількості користувачів у майбутньому;

– надання механізму резервного копіювання та відновлення даних для безперервності роботи системи;

– інтегрування з іншими компонентами фінансового додатку, такими як модулі звітності, аналітики та розрахунків.

Ключовими вимогами до бази даних є висока продуктивність для операцій читання та запису, підтримка транзакцій з властивостями ACID, забезпечення цілісності даних за допомогою обмежень та тригерів, масштабованість для зростаючих обсягів даних, захист конфіденційних даних за допомогою контролю доступу та шифрування, можливість створення складних запитів та звітів,

сумісність та інтеграція з іншими компонентами системи, наявність механізмів резервного копіювання, відновлення та архівування даних. Створення бази даних, що відповідає зазначеним цілям і вимогам, є критично важливим для забезпечення надійної та ефективної роботи фінансового додатку.

Для створення бази даних для фінансового застосунку було обрано PostgreSQL – потужну об’єктно-реляційну систему управління базами даних з відкритим вихідним кодом.

PostgreSQL – це вільно поширювана об’єктно-реляційна система управління базами даних (ОРСУБД), що підтримує більшість стандартів SQL та забезпечує надійне зберігання та обробку даних. Вона розвивається всесвітньою спільнотою розробників протягом більше 30 років і відома завдяки своїй надійності, функціональності та розширюваності [19].

Система управління базами даних PostgreSQL з відкритим вихідним кодом користується значною популярністю та затребуваністю. Її використовують численні відомі компанії та організації, серед яких Apple, Skype, Reddit, Spotify, Instagram та інші. PostgreSQL часто обирають для реалізації веб-додатків, фінансових систем, геоінформаційних систем та аналітичних рішень завдяки її надійності, високому рівню безпеки та потужним функціональним можливостям.

Рішення використати PostgreSQL для створення бази даних фінансового застосунку є обґрунтованим з огляду на функціональність, надійність, масштабованість цієї системи та її здатність відповідати вимогам, що висуваються у фінансовій сфері. Однак, необхідно враховувати специфічні вимоги конкретного проекту, наявні ресурси та експертні знання команди розробників для забезпечення ефективного впровадження та подальшої підтримки даної системи управління базами даних.

Система PostgreSQL надає користувачам широкий спектр переваг:

– потужний функціонал, що включає підтримку транзакцій з властивостями ACID, збережених процедур, тригерів, представлень, успадкування даних, посилальної цілісності та багатьох інших можливостей.

– висока надійність та стабільність завдяки ретельному тестуванню та тривалому процесу розробки, що дозволило PostgreSQL стати однією з найнадійніших систем управління базами даних.

– потужні засоби захисту даних, такі як шифрування даних на рівні диска та в транзитних даних, контроль доступу на рівні рядків, аудит та інші механізми безпеки.

– масштабованість та продуктивність, що дозволяє ефективно обробляти великі навантаження та задовольняти зростаючі потреби в сховищах даних.

– кросплатформеність, що забезпечує сумісність з багатьма операційними системами, включаючи Linux, Windows, macOS, та можливість працювати на різних апаратних платформах.

– відкритий вихідний код, що робить PostgreSQL безкоштовною для використання та модифікації, забезпечуючи економію витрат та гнучкість.

Також існують певні потенційні недоліки PostgreSQL:

– підвищена складність налаштування та адміністрування порівняно з деякими іншими системами управління базами даних.

– обмежена підтримка від комерційних постачальників порівняно з пропрієтарними базами даних, хоча комерційні варіанти підтримки доступні.

– повільне впровадження нових функцій через відкритий характер проекту, що може поступатися швидкості впровадження нових можливостей у комерційних продуктах.

Для початку роботи з PostgreSQL потрібно обов’язково виконати низку дій. Першочергово необхідно завантажити та встановити PostgreSQL на свою операційну систему, використовуючи інсталяційний пакет, доступний на офіційному веб-сайті проекту. Процес встановлення супроводжується інструкціями, які слід ретельно дотримуватися. Під час інсталяції можна обрати додаткові компоненти, наприклад, pgAdmin - графічний інструмент для керування базами даних.

Після успішного завершення встановлення вимагається ініціалізація кластера бази даних, який являє собою сховище для майбутніх баз даних. Типовим способом ініціалізації є виконання команди `*initdb*`, що створює необхідні каталоги та файли.

Наступним етапом є запуск сервера бази даних PostgreSQL. Цей крок реалізується за допомогою команди `*pg_ctl*` або відповідної утиліти, яка залежить від операційної системи. Після запуску сервер починає прослуховувати вхідні з’єднання для роботи з базами даних.

Після успішного запуску сервера можна приступати до створення нових баз даних. Це досягається шляхом виконання команди `*createdb*` або використання клієнтської програми psql, яка дозволяє взаємодіяти з сервером PostgreSQL та керувати базами даних.

Для підключення до створеної бази даних використовується клієнтська програма psql, яка запускається з відповідними параметрами, такими як ім’я користувача та назва бази даних. Альтернативно можуть застосовуватися графічні інструменти керування, наприклад, pgAdmin.

Після успішного запуску сервера PostgreSQL, необхідно створити базу даних за допомогою команди `*CREATE DATABASE*` або використовуючи клієнтську програму psql. Наступним кроком є підключення до створеної бази даних через psql або графічний інструмент, наприклад pgAdmin. Кінцевий результат схеми БД представлений на рис. 3.9.

Для реалізації представленої схеми потрібно послідовно створити чотири таблиці шляхом виконання команд `*CREATE TABLE*`. Першою створюється таблиця "users", яка представлена на рис. 3.5, таблиця для зберігання інформації про користувачів системи. Ця таблиця містить стовпці `*user_id*` (серійний первинний ключ), `*username*`, `*email*`, `*password_hash*` та `*created_at*`.

```
CREATE TABLE users (
    user_id SERIAL PRIMARY KEY,
    username VARCHAR(50) NOT NULL,
    email VARCHAR(100) UNIQUE NOT NULL,
    password_hash VARCHAR(255) NOT NULL,
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);
```

Рисунок 3.5 – Створення таблиці users в PostgreSQL

Наступною створюється таблиця "accounts" для обліку рахунків користувачів, таблиця представлена на рис. 3.6. Вона включає стовпці *`account_id`* (серійний первинний ключ), *`user_id`* (зовнішній ключ, пов'язаний з таблицею "users"), *`account_name`*, *`balance`* та *`created_at`*. Зв'язок між таблицями "users" та "accounts" встановлюється за допомогою обмеження зовнішнього ключа на стовпці *`user_id`* в таблиці "accounts".

```
CREATE TABLE accounts (
    account_id SERIAL PRIMARY KEY,
    user_id INT REFERENCES users(user_id) ON DELETE CASCADE,
    account_name VARCHAR(50) NOT NULL,
    balance DECIMAL(10, 2) DEFAULT 0.00,
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);
```

Рисунок 3.6 – Створення таблиці accounts в PostgreSQL

Третьою створюється таблиця "transactions" для відстеження транзакцій користувачів, представлена таблиця на рис. 3.7. Вона містить стовпці *`transaction_id`* (серійний первинний ключ), *`account_id`* (зовнішній ключ, пов'язаний з таблицею "accounts"), *`amount`*, *`transaction_type`*, *`category`*, *`description`* та *`created_at`*.

```
CREATE TABLE transactions (
    transaction_id SERIAL PRIMARY KEY,
    account_id INT REFERENCES accounts(account_id) ON DELETE CASCADE,
    amount DECIMAL(10, 2) NOT NULL,
    transaction_type VARCHAR(10) CHECK (transaction_type IN ('income', 'expense')) NOT NULL,
    category VARCHAR(50),
    description TEXT,
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);
```

Рисунок 3.7 – Створення таблиці transactions в PostgreSQL

Остання таблиця, яку необхідно створити – "budgets" для зберігання інформації про бюджети користувачів, створення таблиці представлено на рис.3.8. Вона включає стовпці *`budget_id`* (серійний первинний ключ), *`user_id`(*зовнішній ключ, пов'язаний з таблицею "users"), *`monthly_budget`*, *`weekly_budget`*, *`daily_budget`* та *`created_at`*.

```
CREATE TABLE budgets (
    budget_id SERIAL PRIMARY KEY,
    user_id INT REFERENCES users(user_id) ON DELETE CASCADE,
    monthly_budget DECIMAL(10, 2) NOT NULL,
    weekly_budget DECIMAL(10, 2) NOT NULL,
    daily_budget DECIMAL(10, 2) NOT NULL,
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);
```

Рисунок 3.8 – Створення таблиці transactions в PostgreSQL

Наведена у рис. 3.9. схема бази даних з таблицями, а також встановленими між ними зв'язками, утворює комплексну структуру для збереження та керування фінансовими даними в межах додатку. Завдяки використанню зовнішніх ключів, що пов'язують записи в різних таблицях, досягається забезпечення цілісності збережених даних. Застосування функціональності каскадного видалення записів у взаємопов'язаних таблицях спрощує процес управління залежними даними, полегшуючи адміністрування бази даних в цілому.

3.3. Проєктування та прототипування користувацького інтерфейсу фінансового додатку у Figma

На початковій стадії процесу прототипування фінансового застосунку в інструменті Figma потрібно визначити основні мети та функціональні можливості майбутнього додатку. Необхідно сформувати розуміння його ключового призначення, зокрема для відстеження витрат, управління бюджетом чи аналізу інвестицій, а також окреслити перелік ключових функцій, які додаток повинен виконувати. Також потрібно створити ескіз майбутнього додатку для початкового розуміння структури та інтерфейсу додатку, для можливого

консультування з іншими користувачами та для внесення правок у разі необхідності. Початковий ескіз фінансового додатку, розробленого у Figma, представлений на рис.3.10.

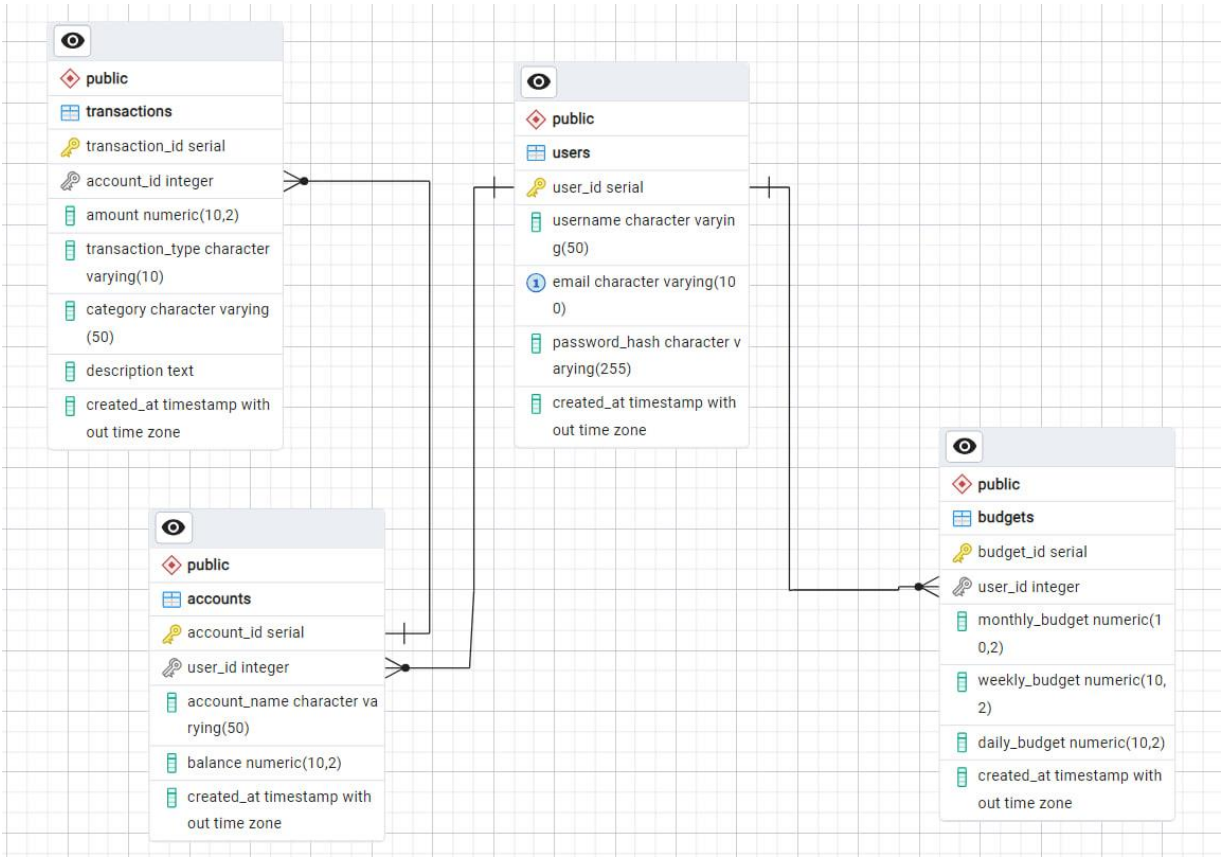

Рисунок 3.9 – Схема таблиць та зв’язків у базі даних PostgreSQL

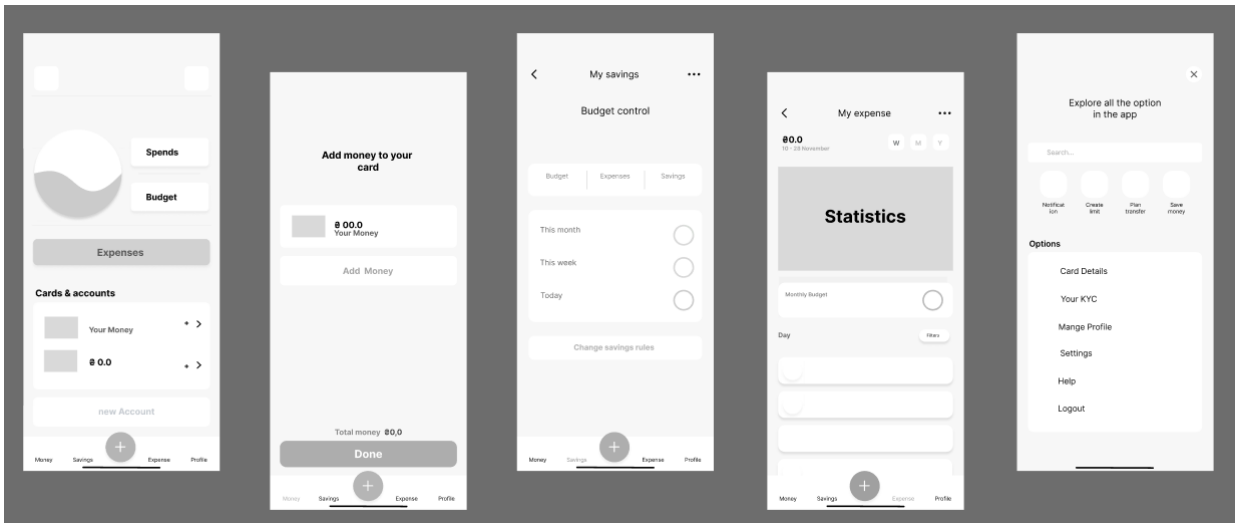

Рисунок 3.10 – Ескіз додатку контролю витрат у Figma

Наступним логічним кроком є розробка інформаційної архітектури, що передбачає визначення основних розділів та екранів додатку у відповідності до його функціональності, проектування структури навігації та взаємозв’язків між різними компонентами додатку, а також створення низькодеталізованих макетів (wireframes) для візуалізації ієрархії контенту та розміщення базових елементів інтерфейсу.

Важливим аспектом є формування стилістичного оформлення та фірмового дизайну майбутнього додатку. Це включає визначення палітри кольорів та шрифтів, які використовуватимуться, розробку фірмового стилю з урахуванням логотипу та іконок, а також створення у Figma набору компонентів інтерфейсу, як от кнопок, полів введення, списків тощо, відповідно до обраного стилю.

Після завершення підготовчих етапів відбувається безпосереднє прототипування інтерфейсу в інструменті Figma, готовий прототип зображений на рис.3.11. Цей процес охоплює створення деталізованих макетів екранів додатку з використанням попередньо розроблених стилів та компонентів, налаштування інтерактивності між екранами, а також формування умовних сценаріїв використання (user flows) для демонстрації основних процесів роботи з додатком.

Врахувавши всі раніше перераховані та розглянуті вимоги, потрібні функції та атрибути, було створено прототип дизайну персонального фінансового додатку, представленого на рис.3.11. Цей прототип демонструє візуальний дизайн і взаємодію різних екранів та елементів інтерфейсу майбутнього застосунку.

Прототип складається з декількох ключових екранів, що відображають основні функціональні можливості додатку. Головний екран надає користувачеві загальний огляд поточного стану фінансів, включаючи візуалізацію загального бюджету, відсотка витрат та залишку коштів. Також на цьому екрані користувач може переглянути список наявних рахунків та платіжних карток, а також додати нові.

Окремий екран дозволяє поповнити баланс обраного рахунку чи картки безпосередньо в додатку. Користувач може вказати суму поповнення та підтвердити операцію.

Прототип також містить екран перегляду заощаджень, де відображається інформація про поточний стан бюджету, встановлені цілі заощаджень та прогрес їх виконання за поточний та попередні періоди.

Є екран детального перегляду витрат за певний період з розбивкою за категоріями у вигляді стовпчикової діаграми. Це дозволяє користувачеві візуально відстежувати розподіл своїх витрат.

Прототип передбачає наявність додаткового розділу з різноманітними опціями та налаштуваннями, такими як управління профілем користувача, доступ до підтримки, вихід з облікового запису тощо.

Загалом, представлений прототип інтерфейсу фінансового додатку охоплює всі ключові функції, визначені на початкових етапах проєктування.

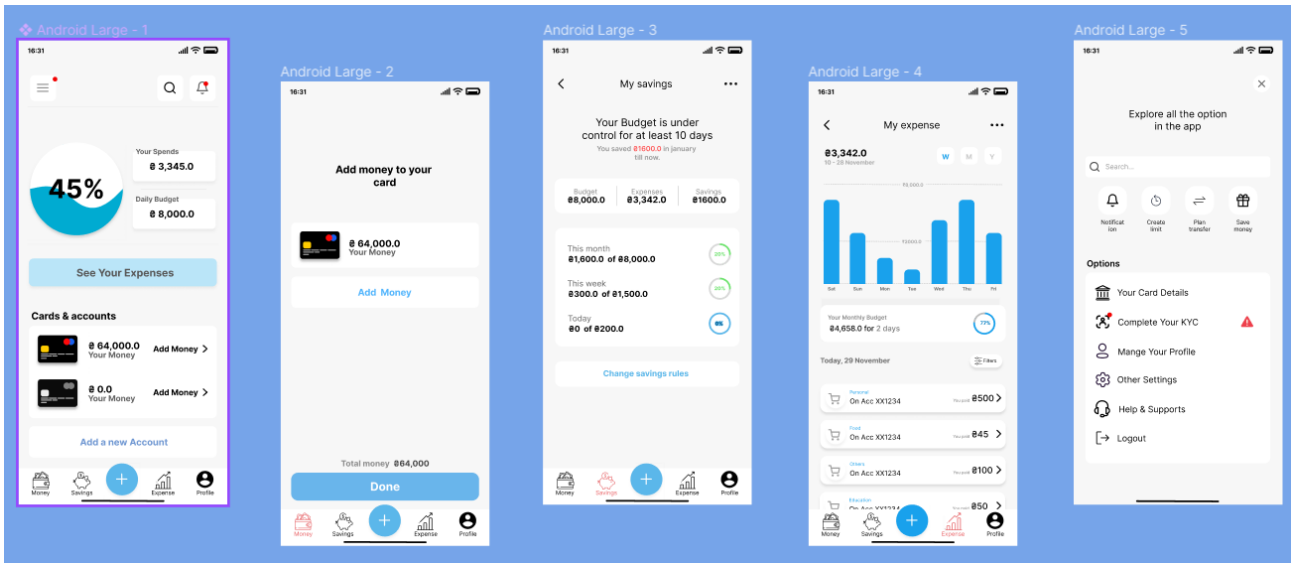

Рисунок 3.11 – Кінцевий прототип дизайну інтерфейсу фінансового додатку у Figma

Завершальним етапом є тестування та ітераційний процес вдосконалення. Необхідно провести юзабіліті-тестування створеного прототипу із залученням представників цільової аудиторії, зібрати відгуки та коментарі стосовно зручності використання інтерфейсу та можливих шляхів покращення.

Юзабіліті-тестування, або тестування зручності використання, є важливим етапом в процесі розробки програмних продуктів та інтерфейсів. Його основна мета - оцінити, наскільки зручним є інтерфейс для цільових користувачів під час виконання типових задач [20].

Для отримання об’єктивної оцінки ергономіки інтерфейсу проведено опитування серед потенційних користувачів, результати якого ретельно проаналізовано та враховано для вдосконалення розробленого прототипу.

За результатами опитування 92,9% користувачів вважають представлене на рис. А.1 (додаток А) початкове вікно прототипу зручним та дуже зрозумілим, результати представлені на рис.3.12.

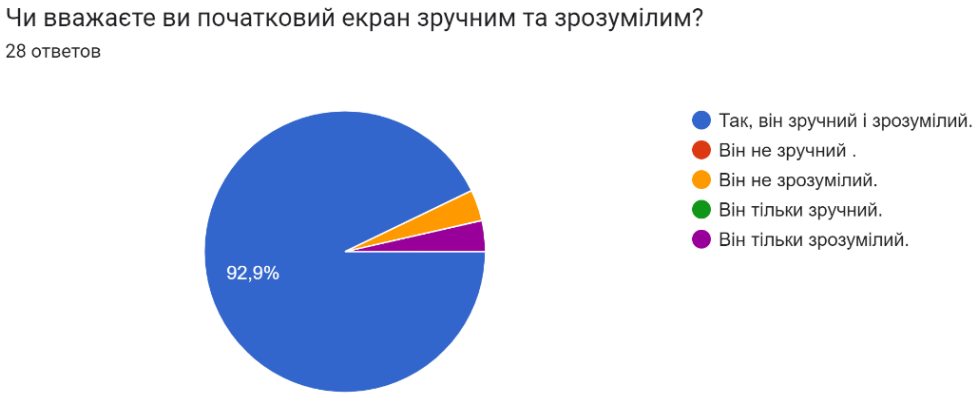

Рисунок 3.12 – Результати першого питання з опитування користувачів

Також великий відсоток користувачів, а саме 85,7% обрали, що вікно поточних коштів, яке представлене на рис. А.2. є зручним та зрозумілим, результати представлені на рис. 3.13.

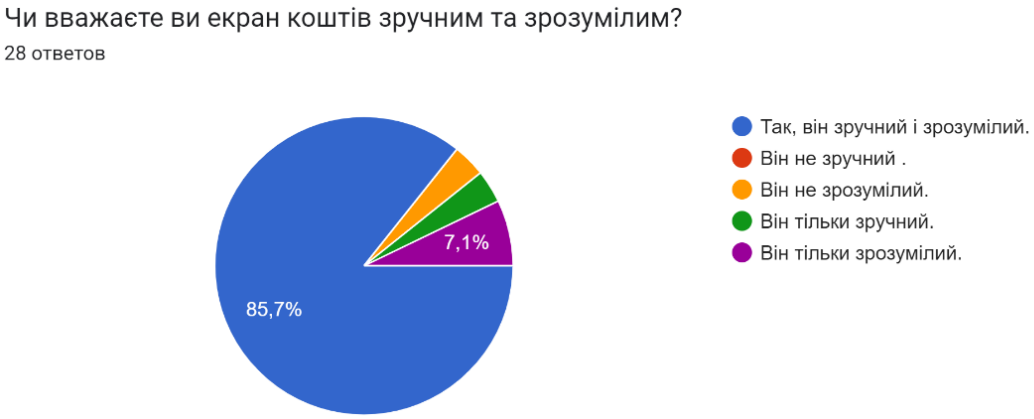

Рисунок 3.13 – Результати другого питання з опитування користувачів

На інші запитання користувачі давали однотипні відповіді без негативних відгуків, що робить розроблений прототип приємним і зручним для користувачів.

Отже, за результатами роботи, виконаної в третьому розділі, запропоновано прототипи для програмної реалізації презентаційного рівня та рівня даних для персонального фінансового додатка. Проведене тестування зручності використання дає підстави вважати запропонований прототип фронтенду доречним для впровадження.

ВИСНОВКИ

У межах кваліфікаційної роботи було проведено ретельне та всебічне дослідження особливостей розробки сучасних персональних фінансових додатків. На першому етапі здійснено ґрунтовний аналіз наявних на ринку аналогічних програмних продуктів, під час якого визначено необхідний функціонал, вимоги та атрибути, що забезпечать конкурентоспроможність майбутнього додатку. Окрему увагу приділено дослідженню різноманітних архітектур програмного забезпечення, за результатами якого обрано найбільш оптимальний варіант та розроблено відповідну базу даних для зберігання фінансових даних користувачів.

Важливим кроком стало створення прототипу інтерфейсу додатку з метою перевірки зручності його використання. Для отримання об’єктивної оцінки ергономіки інтерфейсу проведено опитування серед потенційних користувачів, результати якого ретельно проаналізовано та враховано для вдосконалення розробленого прототипу. У кваліфікаційній роботі комплексно висвітлено як теоретичні, так і практичні аспекти створення сучасного персонального фінансового додатку, що відповідає вимогам та потребам цільової аудиторії. Окрему увагу приділено детальному аналізу ринку аналогів, вибору оптимальної архітектури програмного забезпечення, розробці бази даних та створенню зручного і зрозумілого інтерфейсу на основі відгуків реальних користувачів.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. White A. Millennials and Gen Z are the most likely to use mobile banking apps–here's why, plus budgeting tips. *CNBC*. URL: https://www.cnbc.com/select/why-millennials-gen-z-use-mobile-banking-apps/. (дата звернення: 24.12.2023).

2. Sensor tower's Q1 2021 data digest: finance app installs surged 34% year-over-year in the U.S. and Europe. *Sensor Tower - Market-Leading Digital Intelligence*. URL: https://sensortower.com/blog/q1-2021-data-digest. (дата звернення: 24.12.2023).

3. Топ додатків для обліку фінансів. *Створюємо капітал для отримання пасивного доходу, Е. Зубковський*. URL: https://e-zubkovskiy.com/blog/top-dodatkiv-dlya-obliku-finansiv. (дата звернення: 15.03.2024).

4. ДОДАТКИ ДЛЯ ОБЛІКУ ФІНАНСІВ У 2024 РОЦІ – 10 найкращих. *Інтернет-магазин Moyo.ua* URL: https://www.moyo.ua/ua/news/luchshie_prilozheniya_dlya_ucheta_finansov_v_2023_godu_9_interfeysov_dlya_tochnogo_planirovaniya_kazhdoy_kopeechki.html. (дата звернення: 15.01.2024).

5. How to be a financially successful person? | Money Lover. *Moneylover*. URL: https://moneylover.me/ (дата звернення: 16.05.2024).

6. Money manager & budget planner | spendee. *Money Manager & Budget Planner | Spendee*. URL: https://www.spendee.com/ (дата звернення: 06.02.2024).

7. Розробка фінтех-додатків: тенденції, особливості та перспективи. *Custom Software Development Company | Stfalcon.com*. URL: https://stfalcon.com/uk/blog/post/fintech-app-development-trends-features-perspective . (дата звернення: 17.02.2024).

8. Архітектура веб-додатків - як вибрати?: стаття з блогу IT-школи Hillel. *Корисні матеріали: Статті та новини IT-індустрії | Комп'ютерна школа Hillel*. URL: https://blog.ithillel.ua/articles/web-application-architecture.(дата звернення: 25.02.2024).

9. Архітектура програмного забезпечення | Wezom. *IT-компанія повного циклу розробки програмних продуктів WEZOM - Київ, Україна*.

URL: https://wezom.com.ua/ua/blog/arhitektura-programmnogo-obespecheniya (дата звернення: 09.03.2024).

10. Types of software architecture patterns - geeksforgeeks. *GeeksforGeeks*. URL: https://www.geeksforgeeks.org/types-of-software-architecture-patterns/ (дата звернення: 21.03.2024).

11. Stevanovic V. Six modern software architecture styles. *Multiplayer · Blog*. URL: https://www.multiplayer.app/blog/six-modern-software-architecture-styles/.(дата звернення: 23.03.2024).

12. Editor. Web application architecture: how the web works. *AltexSoft*. URL: https://www.altexsoft.com/blog/web-application-architecture-how-the-web-works/. (дата звернення: 23.03.2024).

13. Y V. Прототипування мобільних застосунків – важливий етап розробки. *Просто про фриланс, штатну роботу і не тільки: блог Freelancehunt*. URL: https://freelancehunt.com/blog/prototipuvannia-iak-vazhlivii-etap-rozrobki-mobilnikh-zastosunkiv/#2. (дата звернення: 10.04.2024).

14. 2019 Design Tools Survey. *UX Tools*. URL: https://uxtools.co/survey/2019/ (дата звернення: 13.04.2024).

15. Gavriluk V. The importance of UX in financial applications & latest trends. *Design Agency Arounda - Digital Product Design & Development solutions*. URL: https://arounda.agency/blog/the-importance-of-ux-in-financial-applications-and-latest-trends (дата звернення: 17.04.2024).

16. UX/UI курс з нуля. Урок 4. Основи Figma I. Реєстрація. Інтерфейс. Перші кроки. Навігація. *Frusia PRO – Любов Фурсеєва – блог про UX/UI дизайн*. URL: https://frusia.pro/p/10. (дата звернення: 07.05.2024).

17. How to create a budget app | EPAM startups & smbs. *Software Development Services for Companies | EPAM Startups & SMBs*. URL: https://startups.epam.com/blog/how-to-build-a-finance-app-to-create-personal-budget (дата звернення: 08.05.2024).

18. Що таке база даних? | Кафедра АПЕПС ТЕФ КПІ – програмна інженерія. *Інженерія програмного забезпечення та комп'ютерні науки в КПІ |*

Кафедра АПЕПС ТЕФ КПІ – програмна інженерія. URL: https://apeps.kpi.ua/shco-take-basa-danykh (дата звернення: 18.05.2024).

19. PostgreSQL: documentation. *PostgreSQL: The world's most advanced open source database*. URL: https://www.postgresql.org/docs/ (дата звернення: 19.05.2024).

20. Nielsen J. Usability 101: introduction to usability. *Nielsen Norman Group*.URL: https://www.nngroup.com/articles/usability-101-introduction-to-usability/ (дата звернення: 25.05.2024).

ДОДАТКИ

Додаток А

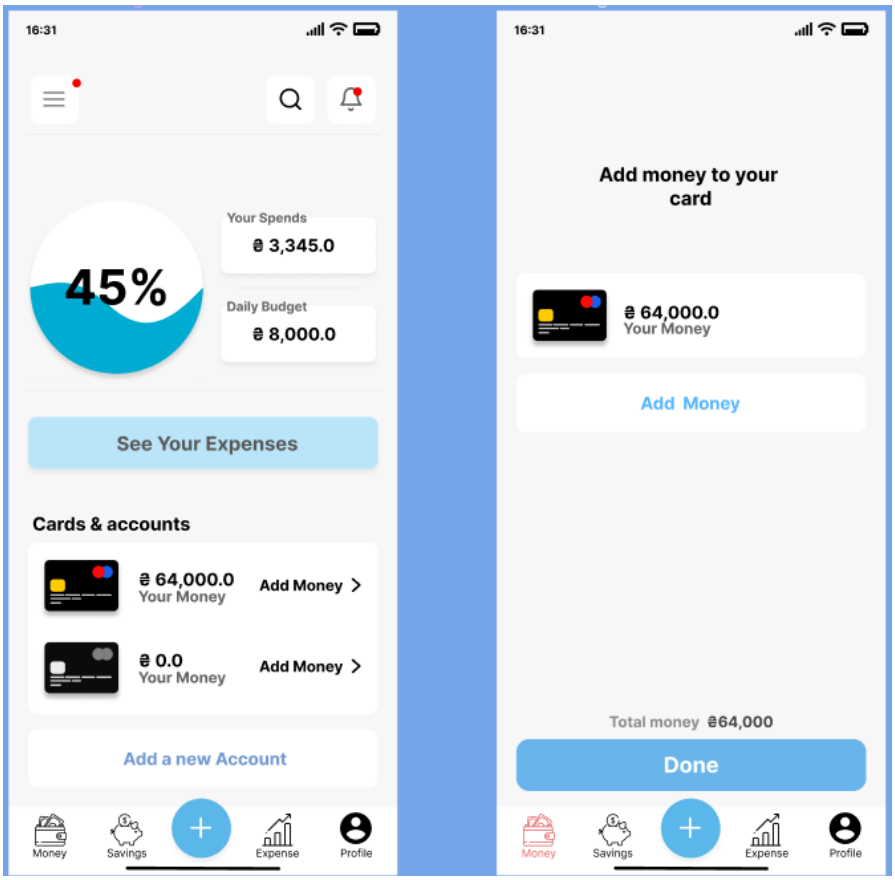

Рисунок А.1 – Початкове вікно та вікно поточних коштів

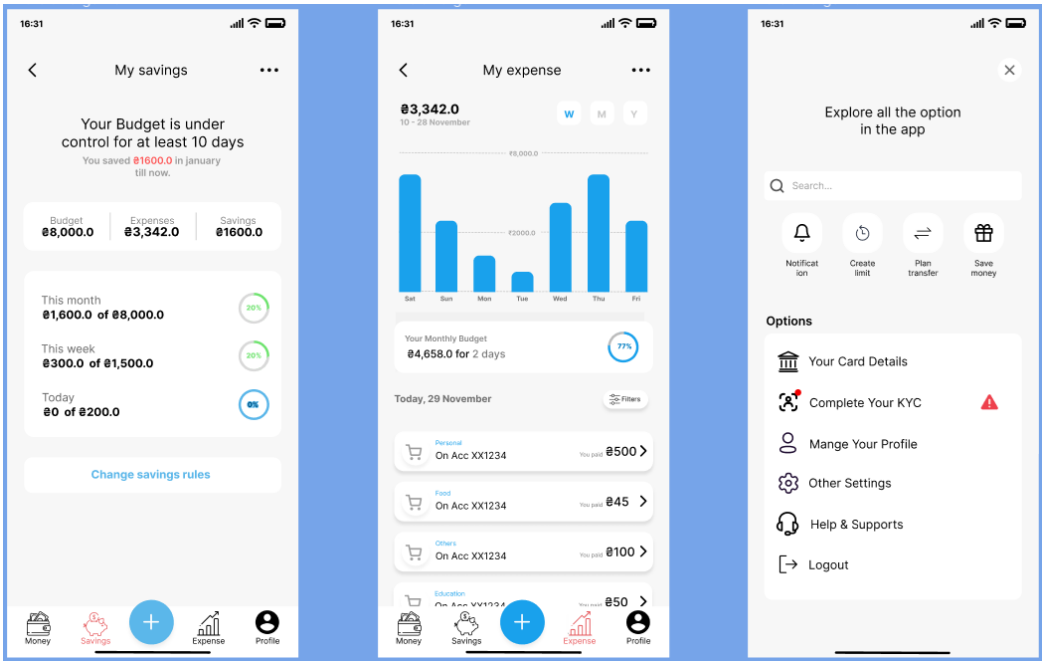

Рисунок А.2 – Вікна збережень, витрат та додаткового меню