МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
НАЦІОНАЛЬНИЙ АВІАЦІЙНИЙ УНІВЕРСИТЕТ
ФАКУЛЬТЕТ КОМП'ЮТЕРНИХ НАУК ТА ТЕХНОЛОГІЙ
Кафедра Комп'ютерних інформаційних технологій

ДОПУСТИТИ ДО ЗАХИСТУ
Завідувач кафедри
_____ Аліна САВЧЕНКО
«_____»_____2023 р.

КВАЛІФІКАЦІЙНА РОБОТА

(ДИПЛОМНА РОБОТА, ПОЯСНЮВАЛЬНА ЗАПИСКА)

ВИПУСКНИКА ОСВІТНЬОГО СТУПЕНЯ

"МАГІСТРА"

ЗА ОСВІТНЬО-ПРОФЕСІЙНОЮ ПРОГРАМОЮ "ІНФОРМАЦІЙНІ УПРАВЛЯЮЧІ СИСТЕМИ ТА ТЕХНОЛОГІЇ"

Тема: «Автоматизована система обслуговування клієнтів з використанням чат-бота для банківських систем»

Виконав: студент групи УС-211М Соколов Владислав Павлович

Керівник: професор Воронін Альберт Миколайович

Нормоконтролер Ігор РАЙЧЕВ

Київ – 2023

НАЦІОНАЛЬНИЙ АВІАЦІЙНИЙ УНІВЕРСИТЕТ

Факультет *Комп'ютерних наук та технологій*

Кафедра *Комп'ютерних інформаційних технологій*

Галузь знань, спеціальність, освітньо-професійна програма: 12 "Інформаційні технології",122 "Комп'ютерні науки", "Інформаційні управляючі системи та технології "

ЗАТВЕРДЖУЮ

Завідувач кафедри

_____Аліна САВЧЕНКО

"___"_____ 2023 р.

ЗАВДАННЯ

на виконання кваліфікаційної роботи студента

Соколов Владислав Павлович

(прізвище, ім'я, по батькові)

1. **Тема роботи:** «Автоматизована система обслуговування клієнтів з використанням чат-бота для банківських систем», затверджена наказом ректора від "29" вересня 2023р. за № 1976/ст.

2. **Термін виконання роботи :** з 02 жовтня 2023р. по 31 грудня 2023р.

3. **Вихідні дані до роботи:** діючі системи організації бюджету, технічна документація програмних засобів, літературні джерела з досліджуваної проблеми.

4. **Зміст пояснювальної записки:** вступ, аналітичний огляд і постановка завдання, розгляд завдання створення чат-бота, дослідження технологій та засобів, розробка програмного продукту, тестування роботи розробленого проекту, висновки.

5. **Перелік обов'язкового ілюстративного матеріалу:** інтерфейс програм фінансового обліку, структура бази даних, скріншоти роботи розробленого продукту.

6. **Календарний план-графік:**

№ п/п	Завдання	Термін виконання	Підпис керівника
1	Аналіз літератури та джерел за темою дипломного проекту.	02.10.2023 – 10.10.2023	
2	Розробка та затвердження плану дипломного проекту.	10.10.2023 – 15.10.2023	
3	Проведення консультації з науковим керівником щодо створення першого розділ.	15.10.2023 – 18.10.2023	
4	Аналітичний огляд і постановка задачі.	18.10.2023 – 24.10.2023	
5	Порівняльний аналіз існуючих програм для ведення бюджету.	24.10.2023 – 06.11.2023	
6	Тестування варіантів заміни існуючих програм та створення чат-боту на базі API Monobank та API Telegram.	06.11.2023 – 19.11.2023	
7	Висновки та оформлення пояснювальної записки дипломного проекту.	19.11.2023 – 27.11.2023	
8	Подати дипломну роботу керівнику	11.12.2023	
9	Підготовка до захисту дипломної роботи	12.12.2023 – 20.12.2023	

7. **Дата видачі завдання:** 02.10.2023р.

Керівник дипломної роботи _____ Альберт ВОРОНІН
(підпис керівника)

Завдання прийняв до виконання _____ Владислав СОКОЛОВ
(підпис випускника)

РЕФЕРАТ

Пояснювальна записка до кваліфікаційної роботи «Автоматизована система обслуговування клієнтів з використанням чат-бота для банківських систем» містить: 70 сторінок, 32 рисунка, 2 таблиці, 9 літературних джерел.

Ключові слова: ЧАТ-БОТ, МОБІЛЬНИЙ БАНКІНГ, ЛІМІТИ, СІМЕЙНИЙ БЮДЖЕТ, TELEGRAM, MONOBANK.

Об’єкт дослідження: оптимізація організації бюджету.

Предмет дослідженя: продукт для планування бюджету за допомогою лімітів.

Мета дипломної роботи: створення чат-боту з функціями автоматичного введення витрат, планування бюджету за допомогою лімітів, перегляду статистики та одночасного користування банківською карткою двома людьми.

Метод дослідження: розробка програмних бібліотек, порівняльний аналіз, обробка літературних джерел.

Область застосування: чат-боти. Під час написання дипломної роботи досліджувалися наявні чат-боти та додатки для автоматизаціїї витрат, програмні інструменти для розробки чат-ботів.

Результати роботи: створено продукт з усіма зазначеними функціями. Проаналізовано інструменти створення чат-боту для Telegram. Порівняно додатки зі схожим функціоналом.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СКОРОЧЕНЬ, ТЕРМІНІВ 6
ВСТУП 7
РОЗДІЛ 1. ЗАГАЛЬНИЙ ОГЛЯД ТЕХНОЛОГІЙ УПРАВЛІННЯ ВИТРАТАМИ 9
1.1. Оцінка роботи мобільного банкінгу 9
1.2. Варіантний аналіз особливостей розробки 12
1.3. Огляд аналогічних програм 20
1.4. Постановка задачі 27
Висновки до розділу 1 29
РОЗДІЛ 2. ПОЯСНЕННЯ ВИБОРУ КОНКРЕТНИХ МЕТОДІВ ДЛЯ ПРОВЕДЕННЯ ДОСЛІДЖЕННЯ 30
2.1. Вибір конкретної форми втілення 30
2.2. API як засіб взіємодії 32
2.3. Перетворення принципу розробки веб-сервісів за допомогою Express фреймворку 34
2.4. Огляд унікальних аспектів створення бази даних 36
2.5. Контроль змін та версій за допомогою GitHub 46
Висновки до розділу 2 50
РОЗДІЛ 3. АНАЛІЗ НАУКОВОГО ДОСЛІДЖЕННЯ 51
3.1. Структура проекту 51
3.2. Проведення тестування розробленого чат-боту 64
Висновки до розділу 3 68
ВИСНОВКИ 69
СПИСОК БІБЛІОГРАФІЧНИХ ПОСИЛАНЬ 70

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СКОРОЧЕНЬ, ТЕРМІНІВ

ПЗ	Програмне забезпечення
API	Application Programming Interface
URL	Uniform Resource Locator
CPU	Central Processing Unit
RAM	Random Access Memory
CSV	Comma-Separated Values
PWA	Progressive Web App

ВСТУП

Програми для фінансового обліку володіють можливістю деталізації операцій та відсіювання зайвої інформації. Мобільні фінансові додатки набагато зручніші за десктопні програми або Excel, дозволяючи швидко вводити дані, переглядати завершені операції, експортувати базу даних та відновлювати її. Тим не менш, десктопні програми залишаються незамінними для важливих завдань, таких як управління фінансами організацій. Однак для особистого обліку грошових потоків мобільні додатки є кращим вибором, завжди доступними на руці.

У найпопулярніших мобільних додатках для фінансового обліку можна створювати кілька типів рахунків (наприклад, кредитні чи дебетові картки, вклади, різні активи), вказуючи їх назву та початковий баланс. Часто підтримуються різні валюти, такі як рублі, долари чи євро, і при введенні операцій можна встановлювати курс конвертації вручну.

Деякі фінансові менеджери мають веб-версії та інструменти для експорту в різні формати, такі як XLS, CSV та інші. Навіть якщо немає власного сервісу в Інтернеті, деякі програми інтегруються з Dropbox та Google Drive, дозволяючи зберігати резервні копії та дані у хмарі.

Метою цієї роботи є: створення чат-боту з функціями автоматичного введення витрат, планування бюджету за допомогою лімітів, перегляду статистики та одночасного користування банківською карткою двома людьми.

Завданнями дослідження є:

1. Дослідження існуючих платформ та інструментів: Оцінка доступних платформ чат-ботів, а також аналіз можливостей інструментів для автоматизації фінансових операцій.
2. Функціональність чат-боту: Розробка функціоналу для автоматичного введення витрат, встановлення лімітів та планування бюджету через чат-бота.
3. Безпека і конфіденційність: Дослідження питань безпеки при використанні банківських даних та засобів автентифікації для одночасного користування банківською карткою двома користувачами.

4. Інтеграція з банківською системою: Вивчення API банківських систем для можливості взаємодії з банківськими даними через чат-бота.

5. Аналіз та відображення даних: Розробка функціоналу для аналізу витрат, відображення статистики та можливості одночасного перегляду фінансової інформації для двох користувачів.

6. Тестування та вдосконалення: Проведення тестів з метою виявлення помилок, забезпечення стабільності та ефективності роботи чат-бота.

РОЗДІЛ 1
ЗАГАЛЬНИЙ ОГЛЯД ТЕХНОЛОГІЙ УПРАВЛІННЯ ВИТРАТАМИ

1.1. Оцінка роботи мобільного банкінгу

Основною перевагою Інтернет-банкінгу є його зручність, який дозволяє клієнтам дистанційно керувати своїми рахунками з будь-якого комп'ютера чи ноутбука, підключеного до Інтернету, не обмежуючи себе територіально або за часом, що приносить економію коштів та часу.

Основні послуги, доступні в режимі онлайн, включають таке:

- Інформаційний сервіс, що надає інформацію про поточні та карткові рахунки, надсилає повідомлення про проведені транзакції та містить стрічку новин.
- Платіжний сервіс, який дозволяє здійснювати регулярні та одноразові платежі, включаючи перекази між рахунками, оплату житлово-комунальних послуг, мобільного зв'язку і т.д..
- Сервіс роботи з поточними, депозитними та картковими рахунками.
- Сервіс роботи з кредитними продуктами банку (майбутній доступ до онлайн-кредитів, відкриття / закриття депозитів тощо).

Переваги використання дистанційного банківського обслуговування:

- Значна економія коштів: програма дозволяє зменшити річні витрати на обслуговування рахунків майже у половину або ще більше.
- Збереження часу: менше потреби у відвідуванні банку або точок самообслуговування.
- Підвищена ефективність: менше потреби у паперовій роботі, можливість оперативно виправляти помилки в платіжних дорученнях.
- Швидкість передачі даних: платіжні документи відправляються протягом кількох хвилин.

- Автоматична перевірка реквізитів документів, включаючи номери рахунків, ІПН / КПП одержувача та банківські реквізити.
- Використання шаблонів для зручності проведення розрахунків у різних валютах.
- Отримання оперативної інформації про курси валют, нові послуги та зміни тарифів.
- Постійний контроль стану рахунку через зручний та зрозумілий інтерфейс.
- Легкий доступ та простота використання.

Проте, разом із численними перевагами, дистанційне банківське обслуговування має свої недоліки. Головною проблемою, що стримує його розвиток, є низький рівень кваліфікації клієнтів і брак довіри до таких операцій. Це особливо актуально з огляду на спроби неправомірного отримання особистої інформації користувачів дистанційних банківських систем.

Крім того, клієнтські ризики є ще однією перешкодою у розвитку цієї форми обслуговування. Вони пов'язані, переважно, з недостатньою захищеністю інформації від несанкціонованого доступу через Інтернет. Навіть при намаганнях розробників покращити системи захисту, існують потенційні загрози, які продовжують виникати.

Причини недоліків у дистанційному банківському обслуговуванні включають проблеми операційних систем, програм зв'язку та браузерів, а також вплив людського фактору. Підтримка високого рівня захисту вимагає значних матеріальних витрат, які можуть собі дозволити переважно великі банки, що розраховують на значні прибутки від надання подібних послуг.

Проте, недоліки, які супроводжують використання дистанційного банківського обслуговування, поступово подолуються різними організаційними та технічними методами. Незважаючи на це, цей вид обслуговування залишається перспективним і важливим напрямком розвитку. Ринок дистанційного банківського обслуговування в України стрімко розвивається, і за останні роки спостерігається значний прогрес у цій галузі.

Різноманітність послуг та функцій — це головна перевага найкращих банківських програм, що дозволяють користувачам виконувати практично всі банківські операції. Це включає фіксацію кредитів, обслуговування нарахування заробітної плати, мобільні чекові депозити, можливості заощадження та інше.

Більше часу — для клієнтів немає часу відвідувати банк та стояти в черзі, щоб отримати готівкові чеки або отримати консультацію.

Безпаперове ведення записів — використання мобільного банкінгу дозволяє фінансовим установам спілкуватися з клієнтами та опрацьовувати юридичні питання в цифровому форматі. Це зменшує бюрократію та дозволяє клієнтам уникати розголошення конфіденційної інформації на папері.

Зменшення витрат на обслуговування клієнтів — технологія мобільного банкінгу може знизити вартість обслуговування клієнтів на 50-70% згідно з дослідженням McKinsey.

Підвищений рівень безпеки — провідні банки приділяють велику увагу своїй репутації, тому вони застосовують протоколи шифрування та безпеки для захисту конфіденційності та цілісності інформації клієнтів.

Персональний контроль користувачів — це можливість для клієнтів відстежувати свої витрати та краще управляти фінансами. Миттєвий банкінг створив зручний простір для міських користувачів, забезпечивши швидкий доступ до облікового запису через смартфон для виконання складних транзакцій у кілька секунд.

Наступним переворотом у банківській сфері є роботизована технологія, що відіграє важливу роль у забезпеченні високої швидкості та цілодобового обслуговування клієнтів. Ці роботи не лише підвищують продуктивність, але й працюють у режимі 24/7, готові надати різноманітні банківські послуги.

Ця дослідницька робота показує, які ролі та послуги можуть відігравати роботи в майбутньому банківському секторі. Роботизовані технології можуть застосовуватися для надання фінансових консультацій, запобігання боргам та виявлення нерегулярних фінансових транзакцій.

Роботизована технологія, що впроваджується в банківській справі, дозволяє уникнути бюрократії та спростити ведення бізнесу. Застосування масштабних сховищ даних та когнітивних обчислювальних робіт забезпечує зниження витрат, виявлення помилок та забезпечує безпеку інформації клієнтів.

Роботизовані технології в банківській сфері змінюють парадигму способу ведення бізнесу, надаючи швидкість, ефективність та безпеку. Хоча ці технології ще не повністю впроваджені, вони вже вносять суттєві зміни у працю банків та визначають напрямок майбутнього.

1.2. Варіантний аналіз особливостей розробки

Особливості розробки платформ для створення мобільних додатків різняться за набором функцій, цінами та можливостями, які вони надають. Остання ознака класифікує їх у дві основні категорії:

Генератори: Ці платформи створюють мобільний додаток на основі існуючої веб-сторінки. Шляхом вказання URL-адреси сайту, генератор автоматично формує мобільний додаток з тим же розділами і контентом, що й на веб-сайті.

Конструктори: Ці платформи дозволяють самостійно збирати додаток з готових елементів, а контент для нього створять користувачі у майбутньому. Конструктори містять готові шаблони і елементи інтерфейсу, а також фрагменти функціональності, такі як геопозиціонування, відправлення повідомлень, робота з банківськими картами та інше.

Існують два типи програм, які можуть створювати ці платформи:

- Гібридні (PWA): Це фактично додатки для вебу, адаптовані для відображення на екранах мобільних пристроїв. Вони запускаються на смартфоні через веб-браузер.
- Нативні. Це програми, які призначені для встановлення безпосередньо в операційну систему мобільного пристрою. Вони є найбільш зручними для користувача та вигідними для підприємця.

Створення програми саме по собі може бути безкоштовним. Після цього вибирається один із двох шляхів. По-перше, можна придбати вихідний код від

сервісу та самостійно його підтримувати та поширювати додаток. Або, по-друге, можна придбати платну підписку, і тоді команда сайту сама опублікує додаток в App Store / Google Play та буде підтримувати його.

Окрім плати за підтримку, також потрібно придбати аккаунт в App Store або Google Play, які коштують $99 та $25 відповідно. Щоб окупити витрати, багато платформ мають програми лояльності, що дозволяють не лише створити додаток, а й заробляти на ньому - наприклад, через підключення реклами.

Наразі існує велика кількість генераторів мобільних додатків, які різняться функціоналом, можливостями та інструментами. Розглянемо детальніше найбільш використовувані.

"Appy Pie" є однією з найдинамічніших платформ для створення додатків у світі. Ця платформа особливо корисна для новачків у цій сфері. Популярність цієї платформи зумовлена широким спектром унікальних функцій. Наприклад, за допомогою Appy Pie можна додавати в додаток вбудовані покупки, рекламу, завантажувати електронні книги чи інший контент, підключати бази даних, інтегрувати соціальні мережі, створювати додатки для обміну миттєвими повідомленнями та багато іншого.

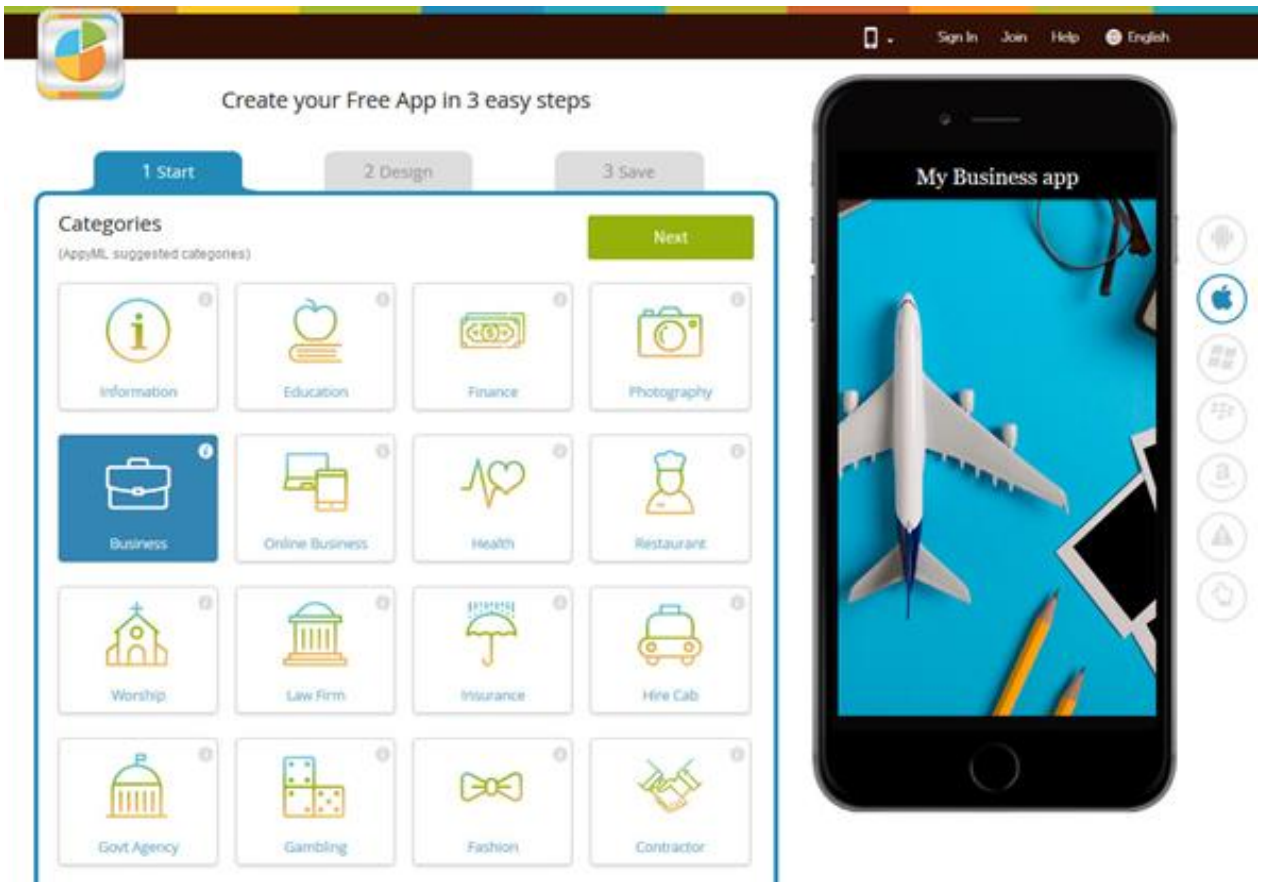


Рис. 1.1. Платформа Appy Pie

Shoutem — це інноваційний продукт на ринку, який з 2011 року постійно розвивається. У своїй останній версії V5 платформа була оновлена, з основною увагою на поліпшення користувацького інтерфейсу. Нові шаблони мають безліч налаштувань, що дозволяє кожному додатку мати унікальний зовнішній вигляд та дизайн. Додатки, створені через цей конструктор, не лише естетично привабливі, а й високофункціональні. Ця платформа ідеально підходить для заходів та спільнот, завдяки соціальній стіні (Social Wall), що дозволяє користувачам обмінюватися коментарями та фотографіями.

Shoutem відкриває свій код для розробників, що стимулює залучення програмістів для розширення можливостей платформи та створення нових функцій. Однак, не дивлячись на чудовий дизайн, користувацький досвід та шаблони, платформа має певні обмеження в функціоналі для створення додатків.

Їхній конструктор, що зараз на 5-й версії, пропонує значні можливості для різноманітних типів програм. Вони також ризикують своїми ставками на

розробників, відкриваючи їхній вихідний код, щоб вони могли пропонувати свої розширення для програми. Головна ідея полягає в створенні моделі, аналогічної до WordPress, де ви, як користувач, можете повністю налаштувати свій додаток за допомогою розширень.

Одним із великих покращень у порівнянні з попередніми версіями цієї платформи є користувацький досвід. Використання редактора Shoutem із чіткими інструкціями та вражаючим дизайном дозволяє зрозуміти, як саме працюватиме ваш додаток - це справжнє задоволення. Шаблони вишукані, сучасні та захоплюючі, а параметри налаштування - на високому рівні.

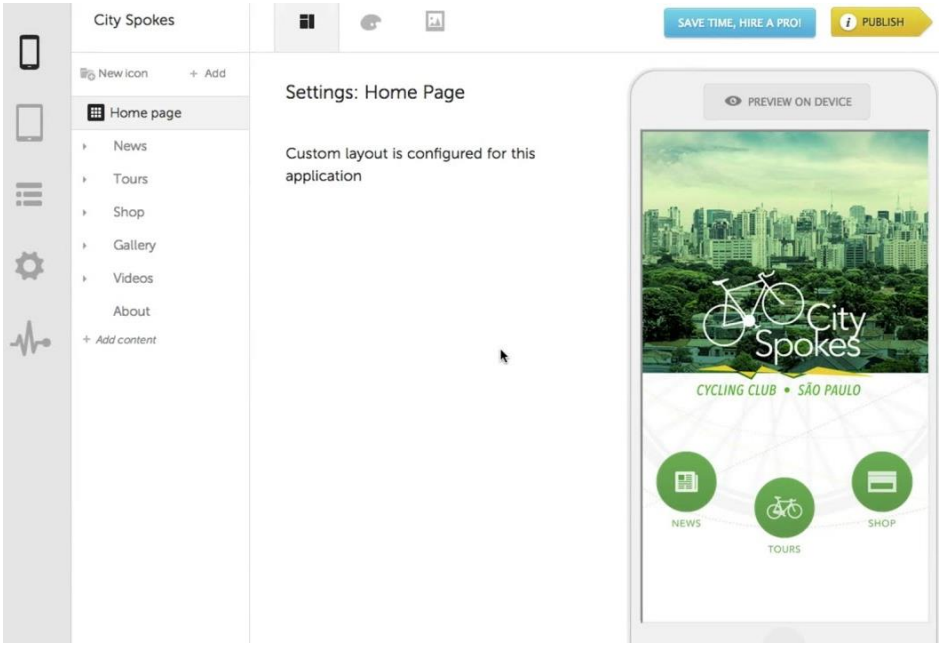


Рис. 1.2. Платформа Shoutem

Заснований у 2010 році, Swiftic походить від ізраїльської компанії Como та став місцем створення понад мільйона додатків по всьому світу. Компоненти, що надаються цією платформою, різноманітні й дозволяють створювати карту лояльності, планувальник зустрічей, e-commerce магазин, а також збирати відгуки та оцінки від користувачів або створювати додатки для заходів.

Більшість додатків, розроблених на цій платформі, призначені для ресторанів, музичних груп та інших організацій, які проводять заходи. Swiftic пропонує сім різних шаблонів, які можна поєднувати з шістьма стилями навігації. Щоб зробити

додаток особистішим, користувач може налаштувати кольори, фонові зображення та іконки на свій смак. Редактор конструктора простий у використанні, проте працює ретельно, надаючи розгалужені можливості у дизайні та функціях.

Кожна програма від Swiftic включає в себе повний набір унікальних інноваційних програм лояльності, що дозволяють підприємствам зручно взаємодіяти зі своїми клієнтами навіть під час їх подорожей та побудовувати міцні та довгострокові відносини із своєю аудиторією.

На даний момент Swiftic вже створив понад мільйон додатків для малих бізнесів по всьому світу, створюючи 4500 нових програм щодня. Ми глибоко розуміємо світ малого бізнесу і прагнемо використовувати ці знання, щоб допомагати власникам бізнесу конкурувати, розвиватися та досягати успіху за допомогою першокласних програм, які не лише привертають увагу клієнтів, але й забезпечують їхню вірність кожен раз, коли вони користуються додатком.

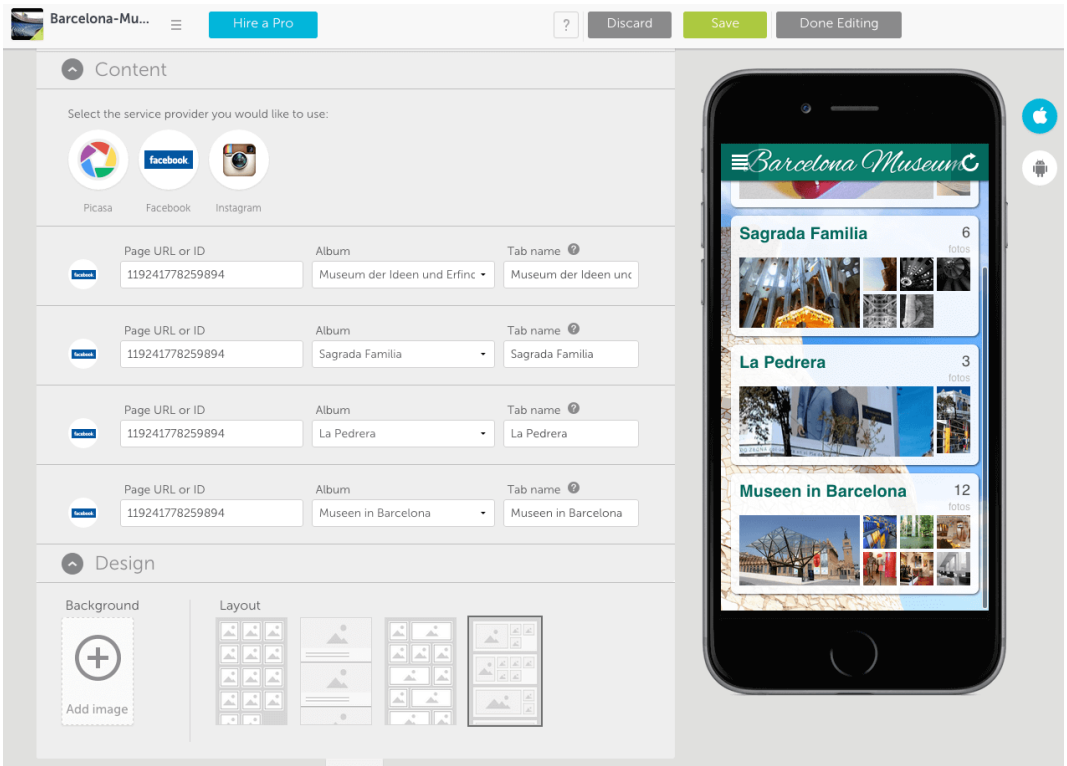


Рис. 1.3. Платформа Swiftic

GoodBarber — це популярний інструмент для створення мобільних додатків та прогресивних веб-програм "Зроби сам". Ця платформа дозволяє створювати мобільні додатки за допомогою широкого спектру шаблонів, зробивши процес створення доступним для будь-якого користувача.

Після створення додатка на базі шаблонів GoodBarber, ви маєте можливість додати до нього додаткові функції, такі як push-сповіщення та реклама, для привертання користувачів та здобуття прибутку через додаток.

Якщо ваші плани включають публікацію контенту у вашому додатку, в GoodBarber навіть є власна система управління контентом (CMS), яка дозволяє публікувати контент безпосередньо у вашому додатку.

Все в цьому конструкторі додатків, від самої назви до функціоналу, зачаровує. Розташована на французькому острові Корсика, ця платформа для створення додатків пропонує одні з найвражаючіших шаблонів. Крім стильного дизайну, конструктор також впроваджує передові функції, такі як соціальні мережі, чати, геофенсінг, iBeacons та інші.

Платформа з гордістю демонструє додатки, розроблені за допомогою їхньої системи, щоб продемонструвати якість та можливості, які вона пропонує. Не лише шаблони привабливі зовні, але вони також пропонують конкурентоспроможні ціни порівняно з нативними додатками. Різноманітні додаткові функції, такі як push-повідомлення, чати тощо, надають велику гнучкість у розробці додатків.

Однак, одним із недоліків є відсутність власного компонента інтернет-магазину. Тим не менше, є можливість інтегрувати сторонні магазини, такі як Amazon, Etsy, Shopify та інші.

Рис. 1.4. Платформа GoodBarber

BuildFire — це одна з надійних платформ, що дозволяє понад 30 000 компаніям створювати свої додатки. Більшість клієнтів цієї платформи — це підприємства та відомі бренди. BuildFire визначає себе як одну з провідних систем для швидкої розробки додатків на ринку.

Зручні панелі управління та адміністрування значно спрощують процес випуску оновлень. Платформа отримала популярність серед клієнтів завдяки своїй простоті використання, можливостям швидкого налаштування та розширеним можливостям зміни програми. Вона дозволяє вносити зміни у реальному часі та тестувати їх на льоту.

Ви здатні повністю контролювати зовнішній вигляд та відчуття вашого додатка, при цьому розробка серверної частини виконується автоматично. Це подібно до того, як у вас є ціла команда розробників завжди під рукою.

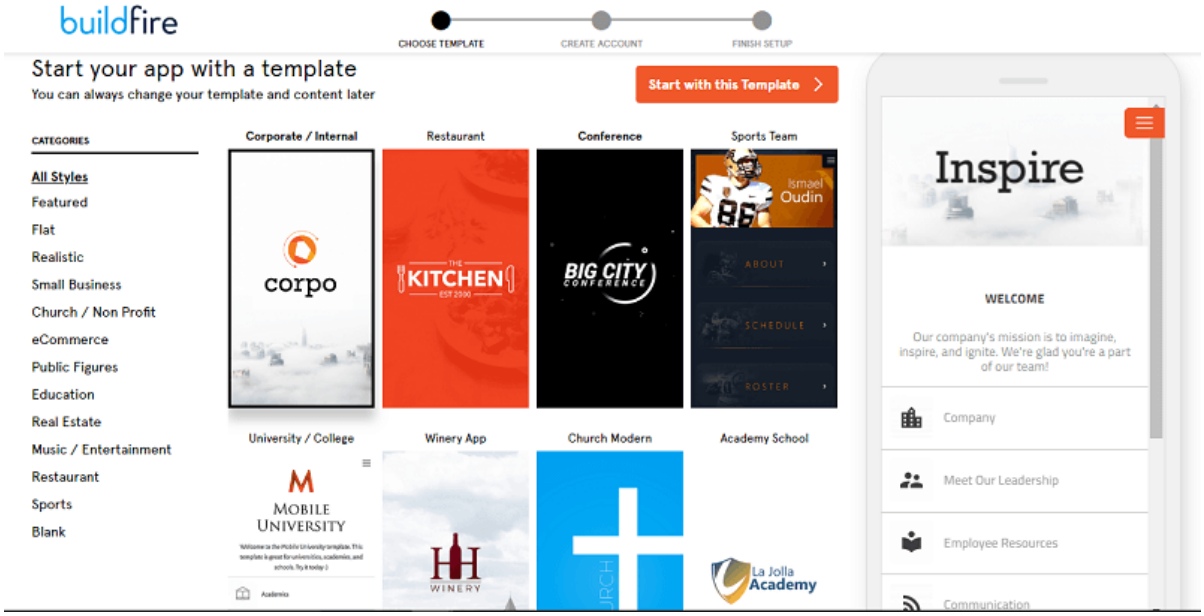


Рис. 1.5. Платформа BuildFire

Незважаючи на широкий спектр можливостей та функцій, які пропонують усі перераховані програми, є одна загальна недолік — наявність платного контенту. Це перешкоджає використанню цих додатків для навчання або особистих потреб. Тому розробка власного генератора мобільних додатків стає більш актуальною, оскільки вона дозволить створювати додатки з вільним доступом до контенту.

Розробка власного генератора мобільних додатків може бути актуальною для тих, хто хоче уникнути обмежень платних платформ. Це може дати більшу гнучкість у розробці, особливо для освітніх цілей або власного використання.

Створення власного генератора додатків може вимагати значних зусиль у розробці, тестуванні та підтримці. Важливо враховувати, що це може бути складним завданням і вимагатиме достатніх знань з програмування та розробки мобільних додатків.

Якщо ціль — створення власного генератора мобільних додатків, то важливо ретельно обміркувати потрібні функції, вимоги до безпеки, інтерфейсу користувача та інші аспекти. Деякі відкриті рішення або програмні бібліотеки можуть допомогти у розробці. Проте, це потребуватиме значного часу, енергії та ресурсів.

Необхідно обдумано підходити до цієї задачі, зважаючи на власні потреби та можливості

1.3. Огляд аналогічних програм

Monefy — це популярний фінансовий менеджер, який користується великою популярністю серед багатьох користувачів. Одна з його ключових переваг — зручний і інформативний вигляд витрат через зрозумілі графіки та докладні списки витрат. Крім цього, програма пропонує функцію бюджетування для ефективного контролю над витратами протягом певного періоду.

Ще одна популярна програма — Money Manager. Вона відзначається тим, що має компаньйон-програму для персональних комп'ютерів, що робить її подібною до Finance PM. Важливою характеристикою є можливість управління кредитними та дебетовими картками через цю програму.

Ці фінансові менеджери надають користувачам зручний інтерфейс та різноманітні функції для керування витратами і фінансами.

Для тесту використовувався мобільний телефон IPhone 13 (Android 11, процесор Apple A15 Bionic 6 ядер; накопичувач 128 ГБ; діагональ дисплея 6,1).

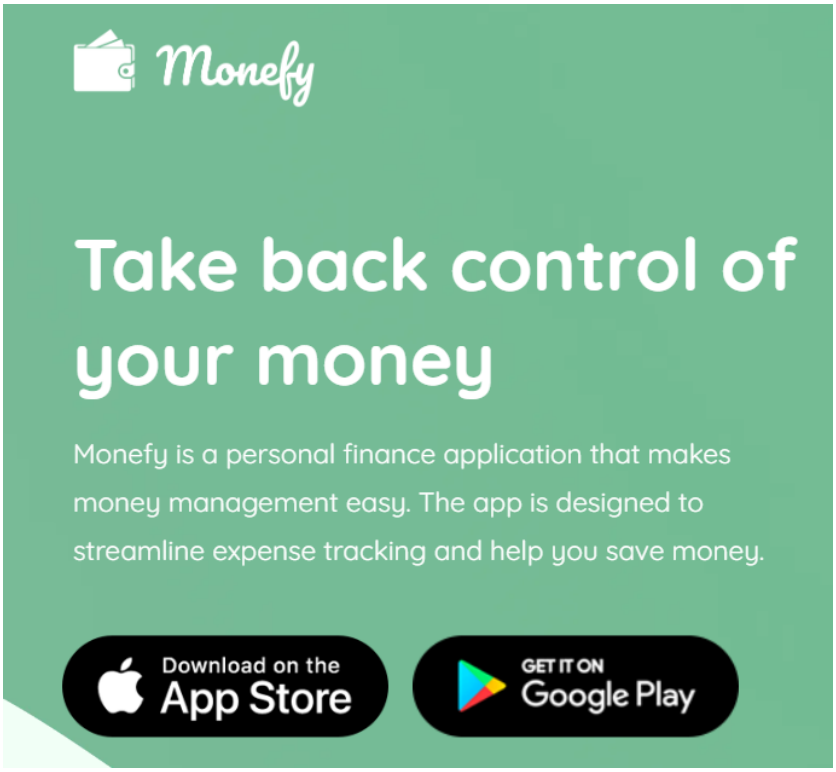


Рис. 1.6. Monefy

Monefy — дуже простий у використанні. Просто клацніть "плюс" для нового доходу або "мінус" для витрат, виберіть категорію для трансакції і завершіть її. Хоча

він не має веб-версії, Monefy може бути синхронізований з Dropbox, щоб оновлювати фінанси без телефону. Ця можливість виглядає дуже професійно. Monefy є безкоштовним і без реклами, але деякі функції доступні тільки для користувачів Pro, такі як захист паролем, синхронізація з Dropbox, створення нових категорій і зміна піктограм. На щастя, ці функції не є абсолютно необхідними. Фактичні дані можна експортувати (у форматі CSV) одним кліком до певної папки на SD-картці з вказаною назвою та періодом. Це легко використовувати в спільній папці на телефоні і налаштувати джерело даних у Excel для завантаження файлу. У Monefy всі записи додаються вручну, що означає, що програма не потребує вашої банківської інформації.

Цей додаток підтримує будь-яку країну, оскільки він не повністю залежить від фінансових установ. Якщо ваш банк не підтримується програмами, такими як Mint і BillGuard, просто скористайтеся Monefy та змініть валюту.

Це дійсно важлива можливість фінансового менеджера — можливість синхронізації між будь-якими пристроями через особистий Dropbox-аккаунт. Це дозволяє редагувати бюджет одночасно на декількох пристроях, забезпечуючи зручність сімейного планування.

Основні функції включають:

- Інтуїтивний та зручний інтерфейс.
- Миттєве додавання нових записів.
- Графічне відображення витрат і детальний перелік у списку.
- Управління категоріями витрат.
- Безпечна синхронізація через особистий Dropbox-аккаунт.
- Вибір звітного періоду.
- Режим бюджету для ефективного контролю витрат протягом певного періоду.
- Можливість створення резервних копій та експорт даних в один клік.

Таблиця 1.1

Характеристики додатку Monefy

Інтерфейс	відмінно
Здатність до індивідуальної конфігурації	добре
Максимальне навантаження, яке система може витримати (CPU / RAM)	0.1% / 52.2-62.6 Мбайт
Можливість обробки різних валютних одиниць	+
Управління	відмінно
Планувальник	+
Кілька гаманців	+
Захист паролем	+
Резервне копіювання	+
Стабільність	відмінно
Веб-інтерфейс	-
Робота з хмарою	+

Звучить, ніби процес додавання нового доходу досить простий і інтуїтивний. Для додавання нового доходу треба натиснути на кнопку «+» або «дохід». Після цього вводиться сума доходу, обирається категорія, наприклад, "зарплата", і вказується рахунок - чи це готівкові кошти, чи карта. Цей процес займає мінімум часу, завдяки дуже інтуїтивному і простому управлінню.

Це дозволяє користувачеві швидко і ефективно вводити нові дані про доходи, не витрачаючи зайвого часу на складні процедури.

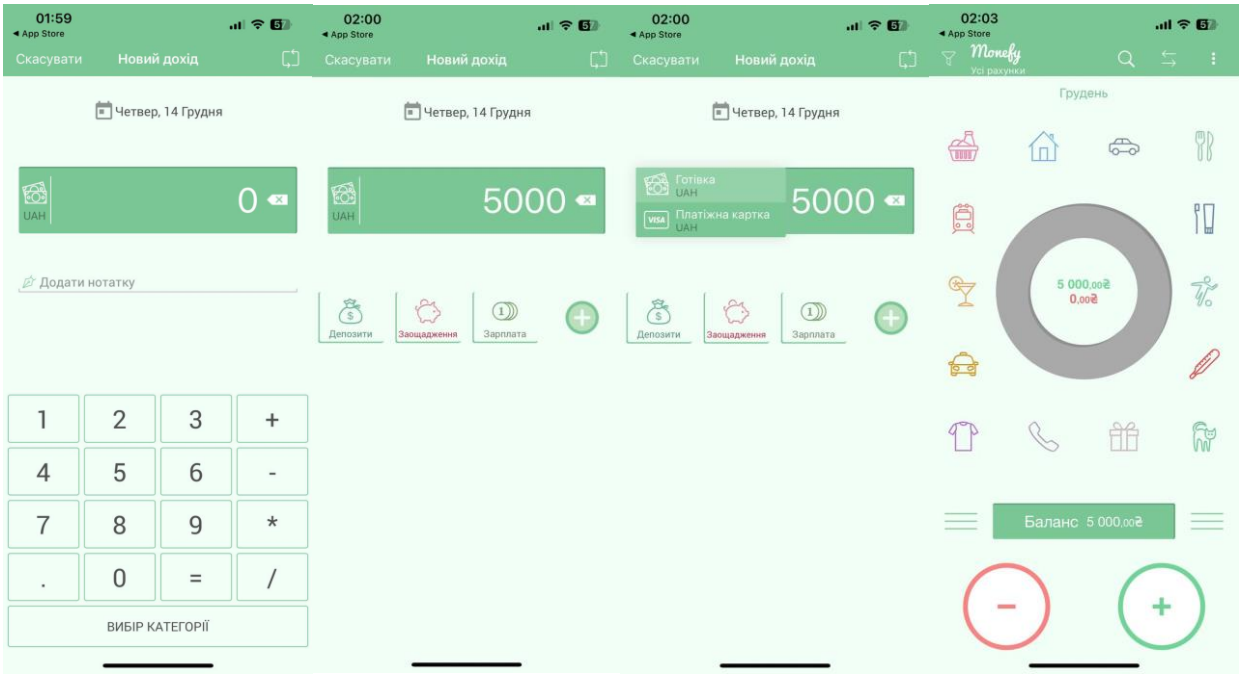


Рис. 1.7. Дохід

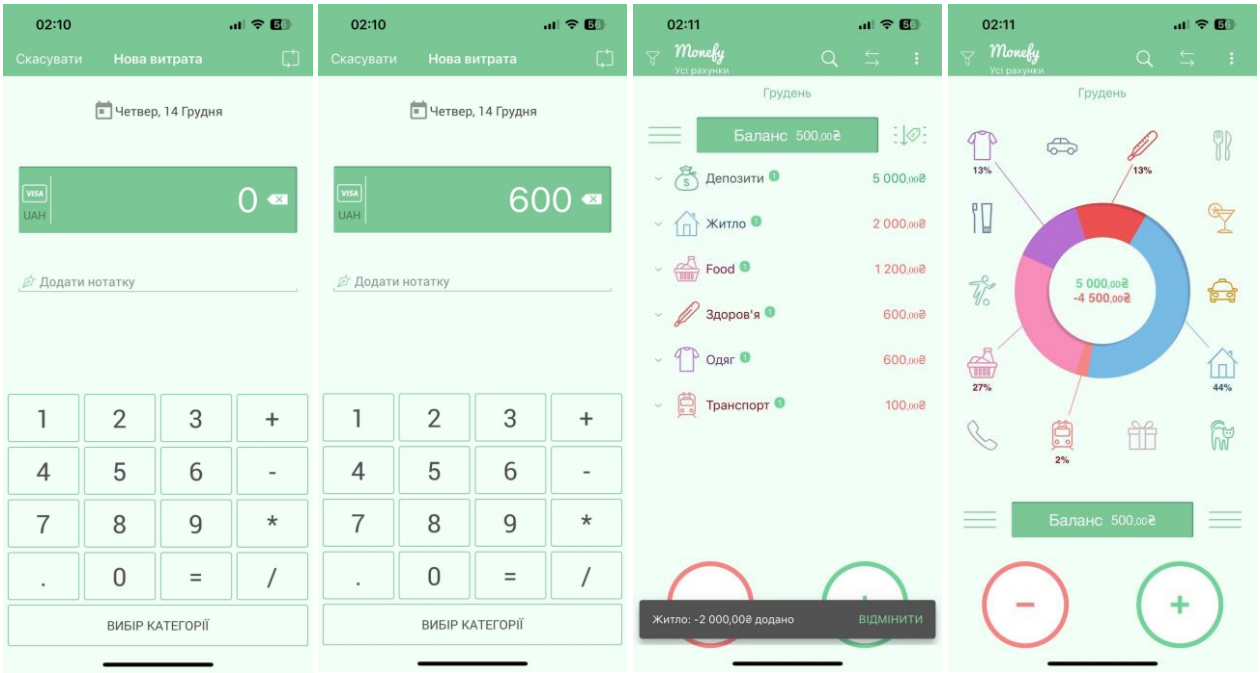


Рис. 1.8. Витрати

KeepSoft — це програмне забезпечення для особистих фінансів, що дозволяє оцінювати фінансове становище, керувати боргами та відстежувати витрати і доходи. Воно допомагає в плануванні майбутніх витрат і, що найважливіше, його

інтуїтивний інтерфейс робить його доступним для будь-якого користувача, навіть без спеціальних навичок.

Програма має ряд корисних функцій, включаючи калькулятор для розрахунків, графіки, що відображають фінансовий стан, і можливість захистити конфіденційні дані за допомогою пароля.

Переваги включають:

- Простота в користуванні та експлуатації.
- Мобільний додаток, що робить його доступним навіть без комп'ютера.
- Зручний інтерфейс користувача.
- Ефективне обслуговування клієнтів.

Недоліки включають:

- Непідтримка формату деяких фінансових програм, таких як Microsoft Money, хоча підтримує HTML, Excel, MS Access, Lotus і XML.
- Відсутність оновлень.
- Неясна структура ціноутворення, що ускладнює визначення відповідності бюджету.

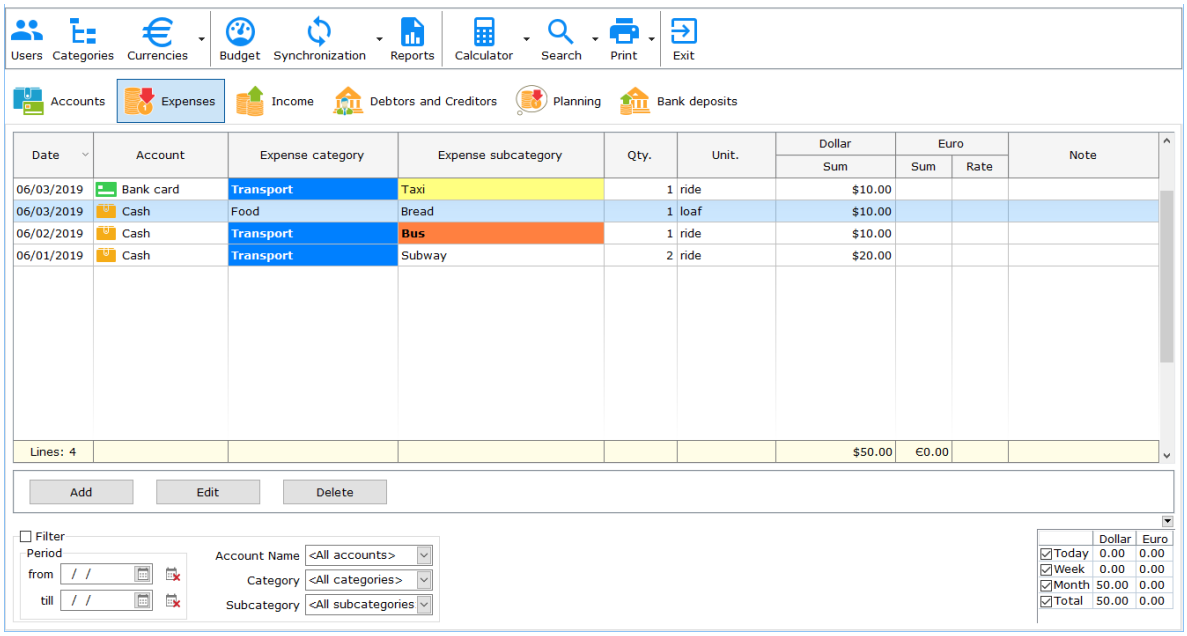


Рис. 1.9. Keepsoft

Функція додавання кредитів та боргів є однією з ключових можливостей цієї програми. Вона дозволяє користувачам вносити інформацію про кредити або борги, які вони взяли. Програма також надає можливість розрахувати графік повернення цих кредитів або боргів. Більше того, якщо необхідно, можна налаштувати регулярні нагадування про відшкодування цих сум.

Це корисна функція, що дозволяє вести контроль та керувати виплатами кредитів або боргів, забезпечуючи користувачеві чіткий розуміння про графік повернення та допомагаючи не пропустити жодних платежів.

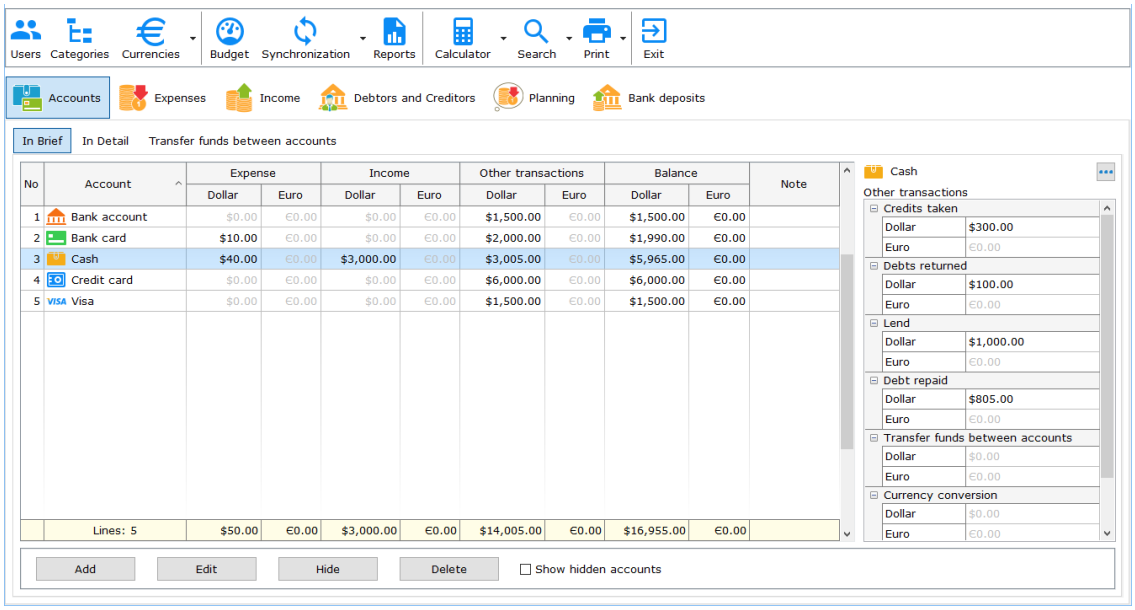


Рис. 1.10. Keepsoft

Ці звіти та діаграми в програмі дозволяють проводити детальний аналіз фінансової інформації та робити відповідні висновки. Окрім цього, важливо відзначити можливість імпорту та експорту даних у файли звичайних форматів Microsoft Office та інших популярних форматах. Це забезпечує зручність обміну даними та їхню сумісність з іншими програмами.

Таблиця 1.2

Характеристики додатку Keepsoft

Інтерфейс	задовільно
Здатність до індивідуальної конфігурації	добре
Максимальне навантаження, яке система може витримати (CPU / RAM)	0.1% / 44,8 Мбайт
Можливість обробки різних валютних одиниць	+
Управління	відмінно
Планувальник	+
Кілька гаманців	+
Захист паролем	+
Резервне копіювання	+
Стабільність	відмінно
Веб-інтерфейс	-
Робота з хмарою	+

Також варто відзначити, що в програмі доступні практично всі валюти світу для вибору, що може бути дуже корисним для різноманітних користувачів з різних країн та регіонів. Це робить програму більш гнучкою та придатною для використання на міжнародному рівні.

Ваш аналіз мобільного банкінгу та додатку Monobank, які автоматично відслідковують витрати, є дуже цінним. І справді, підходи, описані вище, мають свої обмеження:

- Ручне введення витрат: Вимагає багато часу і зусиль для ручного введення інформації про кожну витрату, що може бути незручним і часомістким.
- Відсутність інструментів для планування витрат: Деякі програми можуть бути обмеженими у засобах планування витрат чи недостатньо оптимізованими для цієї мети.
- Вартість продукту: Деякі програми можуть бути платними, що може стати проблемою для користувачів, особливо якщо є обмежені функції.
- Непотрібні функції: Може бути ситуація, коли програми мають зайві функції, які вам не потрібні, що може ускладнити користування.

- Відсутність одночасного сумісного ведення: Деякі програми можуть не забезпечувати зручного одночасного сумісного ведення бюджету для кількох користувачів або сімей.

Дійсно, мобільні банкінгові рішення, як Monobank, надають зручність, не потребуючи вручного введення даних. Але при використанні інших інструментів, важливо враховувати їхні обмеження та здатність задовольнити ваші потреби у плануванні бюджету. Можливо, вам потрібен більш спеціалізований інструмент, що враховуватиме ваші конкретні вимоги щодо управління фінансами.

1.4. Постановка задачі

Задача передбачає проектування та розробку системи для управління особистими фінансами з наступними функціональними можливостями:

1. Ведення особистих фінансів:
- Система повинна дозволяти користувачам вносити інформацію про доходи та витрати.
- Забезпечити можливість реєстрації різних джерел доходу та витрат, таких як зарплата, покупки, рахунки і т.д.
2. Розподіл витрат:
- Користувачам слід мати можливість категоризувати витрати для кращого розуміння того, куди йдуть кошти (харчування, транспорт, розваги і т.д.).
- Забезпечення можливості створення власних категорій витрат та їхнього управління.
3. Аналіз витрат за вказаний період:
- Система повинна надавати можливість перегляду та аналізу витрат за обраний період (день, тиждень, місяць, рік).
- Забезпечити візуальне представлення даних через графіки, діаграми для зручного аналізу витрат.
4. Одночасне сумісне використання:
- Система має підтримувати можливість одночасного користування кількома користувачами (наприклад, для сімейного бюджетування), забезпечуючи окремі профілі користувачів з доступом до власних фінансових даних.

5. Наявність інструменту планування (ліміти):

- Реалізація можливості встановлення лімітів на різні категорії витрат для кожного періоду.
- Автоматичні нагадування або сповіщення користувачам про наближення до встановлених лімітів.

Завдання полягає в розробці функціональної системи, яка забезпечить зручний та ефективний контроль фінансових потоків користувача чи групи користувачів.

Висновки до розділу 1

У першому розділі роботи було розглянуто теоретичні основи та передумови розробки чат-боту для автоматизації контроля витрат клієнтів банку.

Зокрема, в підрозділі 1.1. проведено оцінку мобільного банкінгу, яка базується на його зручності, функціональності, безпеці та якості підтримки. Зручний інтуїтивний інтерфейс, розширені функції, висока рівень безпеки та ефективна підтримка впливають на задоволення користувачів та їх довіру до цього сервісу.

В підрозділі 1.2. зроблено варіативний аналіз особливостей розробки, яка дозволяє вибрати оптимальний набір інструментів, технологій та підходів для створення програмного продукту. Аналізуючи різні варіанти, розробники можуть знайти оптимальний шлях досягнення поставлених цілей.

В підрозділі 1.3. проведено Огляд аналогічних програм, що дозволяє зіставити різноманітні функції, особливості та можливості конкуруючих програм, що сприяє збагаченню функціональності власного продукту та виявленню його конкурентних переваг.

В підрозділі 1.4. Чітко сформульована постановка завдання визначає мету та обсяг робіт, що дозволяє орієнтуватися при розробці програмного продукту. Вона створює основу для управління процесом розробки та є важливим етапом для досягнення успіху в проекті.

Ці етапи оцінки, аналізу та постановки завдань є ключовими у процесі створення та розвитку програмного продукту, допомагаючи досягти успіху та задоволення користувачів. Отримані висновки сприяють формулюванню вимог до чат-боту і формуванню необхідної бази для подальших кроків у проектуванні та розробці. Ці етапи будуть виконані у наступних розділах роботи.

РОЗДІЛ 2
ПОЯСНЕННЯ ВИБОРУ КОНКРЕТНИХ МЕТОДІВ ДЛЯ ПРОВЕДЕННЯ ДОСЛІДЖЕННЯ

2.1. Вибір конкретної форми втілення

У зв’язку з виникненням потреби у сімейному бюджеті для спільних покупок та грошових переказів, було обрано Monobank. Користувач отримує фізичну карту та мобільний варіант, який працює на базі NFC. Однак, функція сповіщень про транзакції та баланс рахунку доступна тільки через мобільний додаток Monobank, не забезпечуючи швидкого доступу для користувача фізичної карти.

Для вирішення цієї проблеми було обрано платформу Telegram, яка є високонадійним сервісом обміну повідомленнями та дозволяє створювати чат-ботів. Чат-боти, що базуються на штучному інтелекті, імітують розмови з користувачами природною мовою та не потребують завантаження чи оновлення, забираючи мінімальне місце в пам'яті пристрою.

Однією з великих переваг чат-ботів є їхня універсальність: вони дозволяють інтегрувати кілька ботів в одному чаті, уникнувши потреби переходити від одного додатка до іншого для задоволення потреб користувача в реальному часі.

Чат-боти можуть бути відкритими чи закритими. Відкриті чат-боти використовують штучний інтелект для обробки мови та взаємодії з користувачами, тоді як закриті чат-боти просто виконують певні сценарії без врахування контексту.

З плином часу зростає популярність мобільних додатків, із цим збільшується кількість проектів, які, однак, часто потребують більш гнучкої архітектури для впровадження нових функцій. У даному проекті передбачається велика кількість модулів та їхнє подальше зростання, тому основною метою архітектури є можливість простого тестування окремих модулів та масштабування системи без необхідності повного переписування додатку.

У зв'язку з цим, вибір паверхневої архітектури Clean Architecture Роберта Мартіна є обгрунтованим, оскільки цей підхід дозволяє створити модульну систему, що легко тестується та піддається змінам без значного впливу на весь додаток.

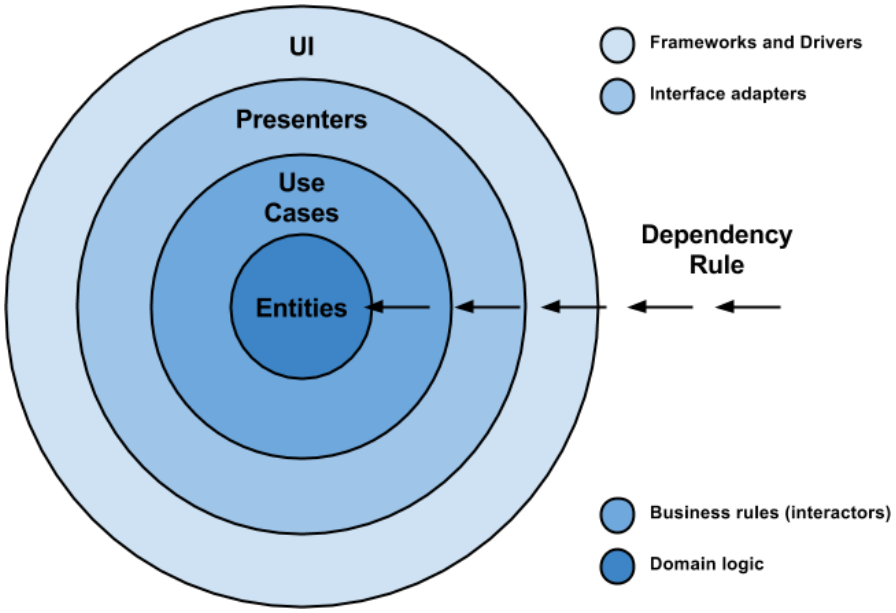


Рис. 2.1. Графічна ілюстрація підходу Clean Architecture

Основні принципи Clean Architecture, які представлені на Рис. 2.1., можна узагальнити наступним чином:

- **Автономність від архітектурних обмежень**: Архітектура програми не залежить від конкретних бібліотек чи SDK. Вона не обмежена рамками, запропонованими певними бібліотеками.
- **Безпосереднє тестування бізнес-логіки незалежно від інших компонентів системи:** Бізнес-логіку можна тестувати без участі користувацького інтерфейсу, баз даних, веб-серверів та інших зовнішніх елементів.
- **Автономія від конкретного інтерфейсу для користувача:** Інтерфейс користувача може змінюватися без впливу на інші частини системи. Наприклад, веб-інтерфейс можна замінити на консольний, не змінюючи при цьому бізнес-правила.

- **Незалежність від конкретної бази даних**: Важливо, щоб можливість замінити одну базу даних на іншу (наприклад, Oracle на Mongo) була простою і без необхідності внесення змін у бізнес-логіку.

У розробці програмного забезпечення за Clean Architecture рекомендується розділити додаток на три основні блоки:

- **Presentation** : Тут розташовується патерн MVP (Model-View-Presenter) та компоненти відображення.
- **Domain** : Він включає в себе чисті об'єкти на Java, які відповідають за бізнес-логіку.
- **Data** : Цей модуль містить опис методів доступу до даних, які можуть бути з мережі або збережених на пристрої файлів. Це показано на Рис. 2.2.

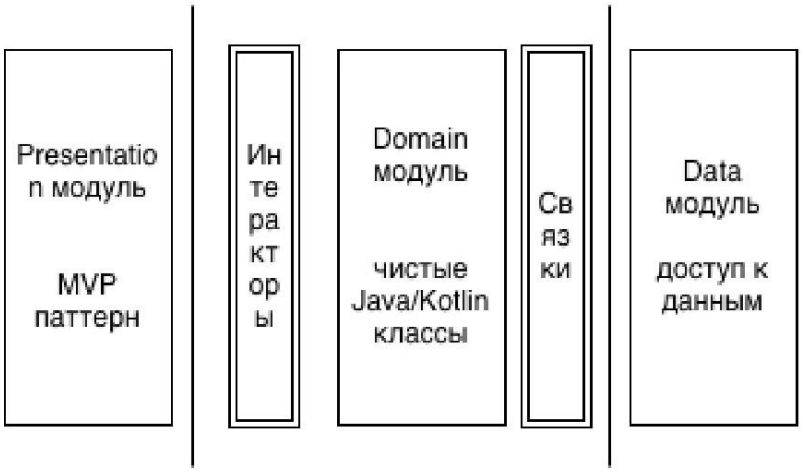


Рис. 2.2. Поділ програми на модулі по Clean Architecture

2.2. API як засіб взаємодії

API, як засіб взаємодії, представляє специфікацію можливих обмінів з компонентом програмного забезпечення. Це схоже на інформацію про функції машини, де можна вказати, що машина може прискорюватися, гальмувати і включати радіо. Також, API містить інструкції щодо способів прискорення. Робота API охоплює три рівні:

- **Додаток**: Це програми, які використовуєте на смартфоні чи ПК.

- **Програмування**: Розробники використовують API для написання програмного коду.
- **Інтерфейс**: Це ваш спосіб взаємодії з додатком.

Для прикладу, уявіть офіціанта в ресторані. Він є посередником між вами (клієнтом) та кухнею (виконавцем замовлення). Цей посередник, або API, приймає ваше замовлення, повідомляє кухні та принесе вам їжу. Подібно до цього, публічні API, такі як Slack або Shopify, надають набір вступних параметрів для використання розробниками. Приватні API, використовані всередині компанії, допомагають програмам взаємодіяти між собою, змінюючись за бажанням компанії.

API (інтерфейси прикладного програмування) складаються з двох головних компонентів: технічної специфікації, що описує обмін даними між програмами, та самого програмного інтерфейсу, який реалізує цю специфікацію. API дозволяють програмам взаємодіяти між собою, викликаючи певні функції або отримуючи доступ до даних з іншого програмного забезпечення.

API може бути розглянутий як контракт між програмами, де документація API надає інформацію про те, яким чином програми можуть взаємодіяти. Це визначає, які дані можна отримати та які дії можна виконати через це API.

API використовує виклики функцій для запитування конкретних дій або послуг. Ці виклики функцій можуть бути виконані різними способами, наприклад, створенням сеансу, отриманням інформації, тощо.

Використання API може спрощувати і прискорювати розробку програмного забезпечення. Розробники можуть використовувати функціональні можливості інших постачальників для розширення функціоналу своїх програм або для створення нових програм.

У даній роботі використовуються два види API: Telegram та Monobank, які є публічними та застосовуються на рівні інтерфейсу. Monobank API використовується для ідентифікації та отримання інформації про транзакції за допомогою токену користувача. Також, Telegram API використовується для обробки запитів користувача та інформації від банку та бази даних.

2.3. Перетворення принципу розробки веб-сервісів за допомогою Express фреймворку

Express — довгостроковий фреймворк для створення додатків, що займає провідну позицію в усвідомленні Node.js. Він володіє високою стабільністю та надійністю, роблячи його одним із найпопулярніших фреймворків у світі Node.js.

Express надає велику кількість детальних інструкцій, підготовлених розробниками, які перевірили його ефективність на практиці. Це оптимальний вибір для початку роботи з Node.js, оскільки з його допомогою можна оволодіти створенням додатків.

Основна особливість Express полягає в його невеликому обсязі базового функціоналу. Додаткові функції можна легко додати, використовуючи зовнішні модулі, оскільки Express сам по собі представляє собою мінімальний сервер і може бути пустим без жодних модулів.

Такий мінімалізм дозволяє розробникам отримати швидкий та легкий інструмент, який можна розширювати за потреби.

Однак варто зауважити, що вибір модулів для Express не обмежений жодними рамками, ні кількісними, ні функціональними. Цей фреймворк дає можливість розробникам вирішувати будь-які завдання, не обмежуючи їх у виборі інструментів.

Це може бути і перевагою і недоліком. Відсутність універсальних рішень означає, що кожен створений додаток буде унікальним. З іншого боку, розробнику доведеться самостійно організовувати модулі, що може вимагати більше часу та зусиль.

Наприклад:

```
var express = require ( 'express');
var app = express.createServer ();
app.get ( '/', function (req, res) {
   res.send ( 'Hello World');
});
app.listen (3000);
```

Зображення web-сервера, що прослуховує порт 3000 та відповідає на запити до "/", виводячи "Hello World", підтримується Express. Цей фреймворк має міцну систему маршрутизації, яка дозволяє створювати правила для різних шляхів. Наприклад, код "/user/:id" автоматично встановлює значення ":id" як "foo" для шляху "/user/foo". Можливо також використовувати регулярні вирази для опису маршрутів.

Якщо потрібно обробляти POST-запити, для використання їх у вашому додатку вам слід використовувати спеціальний middleware - bodyParser. Додавання його до додатку легко: app.use(express.bodyParser()). BodyParser обробляє тіла запитів у форматі "application/x-www-form-urlencoded" та "application/json" і надає доступ до них через req.body. Наприклад, код може вивести значення змінної "foo" з тіла запиту на консоль і повернути "ok" як відповідь.

В Express також доступні інші middleware, такі як express.logger(...), express.cookieParser(...), express.session(...), express.static(...), express.errorHandler(...). Кожен з них відповідає за певні аспекти, такі як логування, обробка cookies, робота з сесіями, статичний контент і обробка помилок. Докладніше про них можна дізнатися з документації.

Express також підтримує різні шаблонні движки, такі як Jade, який дозволяє створювати шаблони з розширенням ".jade". Ці движки вимагають модуля, що експортує метод "exports.compile(str, options)", що забезпечує обробку тіла шаблону. Наприклад, за допомогою коду app.get('/', ...) можна використовувати шаблон "index.jade", передаючи йому об'єкт з параметром "title: 'My Site'".

Основні компоненти та функції Express.js:

- Маршрутизація: Express надає простий спосіб визначення URL-адрес для обробки запитів. Маршрутизація дозволяє вам встановлювати шляхи (routes) для обробки HTTP-запитів (GET, POST, PUT, DELETE тощо) та відправлення відповідей.
- Проміжне програмне забезпечення (middleware): Це ключовий елемент Express. Проміжне програмне забезпечення - це функції, які обробляють запити перед тим, як вони досягнуть функцій-обробників маршруту. Вони

можуть виконувати певні дії з запитами, модифікувати їх, обробляти аутентифікацію, логування тощо.

- Шаблонування: Express дозволяє використовувати різноманітні двигуни шаблонів (наприклад, Pug, EJS), які допомагають генерувати HTML-код на сервері. Це корисно для створення динамічного вмісту на сторінках веб-сайту.
- Робота з HTTP-запитами та відповідями: Express дозволяє легко отримувати дані з HTTP-запитів (від користувача) та надсилати відповіді (наприклад, HTML-сторінки, JSON-дані).
- Налагодження (Debugging): Він надає простий спосіб обробки помилок та відстеження їх місця в коді. Це полегшує пошук та виправлення помилок в програмах.

Express надає розробникам багато можливостей для створення веб-додатків, дозволяючи працювати з HTTP-запитами, керувати маршрутами та використовувати проміжне програмне забезпечення для обробки запитів до сервера.

Основні можливості Express включають гнучку систему маршрутизації, перенаправлення, уточнення контенту, продуктивність, підтримку шаблонів, конфігураційні налаштування, сесії, високий рівень тестування та інші. Він також підтримує сесії, кешування, роботу з mime, ETag, постійні сповіщення, куки, JSON RPC, логування та інше.

2.4. Огляд унікальних аспектів створення бази даних

Сучасні інформаційні технології ґрунтуються на базах даних, які відіграють ключову роль у житті більшості людей. Ці бази розповсюджені різноманітно - від малих підприємств, що мають свої власні бази товарів, працівників, партнерів та постачальників, до глобальних баз даних, таких як державні системи, що містять реєстраційні дані про людей чи автомобілі. Бази даних дозволяють ефективно організувати та систематизувати інформацію, що сприяє швидкому пошуку та обміну даними. Тому вони займають провідні позиції на ринку програмних продуктів.

База даних (БД) — це набір даних, що зберігається згідно зі схемою та відображає стан об'єктів та їх взаємозв'язків, які обробляються відповідно до правил моделювання даних. Важливі характеристики:

- БД зберігається та обробляється в обчислювальних системах. Інші сховища (архіви, бібліотеки, картотеки) не є базами даних;
- Дані в БД структуровані логічно з метою ефективного пошуку та обробки. Ця структурованість дозволяє відокремлювати елементи, їх зв'язки та типізацію, що упорядковує операції;
- БД включає схему чи метадані, що формально описують логічну структуру БД.

Значення реляційної бази даних полягає в колекції пов'язаних таблиць. Відмінність між базою даних і реляційною базою даних виявляється у структурі таблиць. У реляційній базі даних таблиці організовані таким чином, що між ними існує логічний зв'язок. За допомогою інформації, що міститься в одній таблиці, можна отримати відповідні дані з іншої таблиці.

Процес розробки баз даних включає кілька етапів.

Перший етап, концептуальне проектування, відображає опис і синтез вимог щодо інформації від користувачів у початковому проекті бази даних. Початкові дані можуть представляти собою набір користувацьких документів у класичному підході або алгоритми програм у сучасному підході. Результатом цього етапу є високорівневе представлення (у вигляді системи таблиць бази даних) вимог користувачів на основі різних методів підходу.

Спочатку обирається модель бази даних. Потім створюється структура бази даних, яка заповнюється інформацією через системи меню, екранні форми або у режимі перегляду таблиць бази даних. Також на цьому етапі забезпечується захист і цілісність даних (включаючи посилальні) за допомогою систем управління базами даних або шляхом застосування тригерів.

У процесі логічного проектування використовується високорівневе представлення даних у структуру системи управління базами даних, що використовується. Головною метою цього етапу є усунення надмірності даних за

допомогою спеціальних правил нормалізації. Основна мета нормалізації полягає в уникненні повторення даних і можливих структурних змін у базі даних під час процедур оновлення. Це досягається шляхом розбиття (декомпозицією) однієї таблиці на дві або кілька, за допомогою яких потім виконуються операції навігації при запитах. Варто зауважити, що навігаційний пошук може знизити продуктивність бази даних, збільшуючи час відповіді на запити.

Логічна структура бази даних, отримана в результаті цього етапу, може бути оцінена за допомогою різних кількісних характеристик, таких як кількість обращень до логічних записів, обсяг даних у кожному додатку та загальний обсяг даних. На основі цих оцінок логічна структура може бути оптимізована з метою підвищення ефективності.

Особливу увагу варто звернути на процедуру управління базою даних. У простому випадку, коли один користувач працює з базою даних, ця процедура є простішою. Проте, у випадку багатокористувацького режиму або розподілених баз даних, цей процес стає складнішим. Одночасний доступ декількох користувачів без вжиття відповідних заходів може призвести до порушення цілісності бази даних. Для уникнення цього використовуються системи транзакцій та режими блокування таблиць або окремих записів.

Транзакція представляє собою процес зміни файлу, запису або бази даних, ініційований відправленням одного вхідного повідомлення. Особливості блокування та його варіанти будуть розглянуті окремо.

На етапі фізичного проектування бази даних вирішуються питання, що стосуються продуктивності системи, визначаються структури зберігання даних та методи доступу до них.

Взаємодія між етапами проектування та словниковою системою потребує окремого розгляду. Процедури проектування можуть застосовуватися незалежно від наявності словникової системи. Сама словникова система може розглядатися як складова частина автоматизації проектування.

Засоби проектування і оціночні критерії застосовуються на всіх етапах розробки. В теперішній час невизначеність при виборі критеріїв є найбільш

вразливим моментом у проектуванні баз даних. Це пов'язано з труднощами у описі та ідентифікації великої кількості альтернативних рішень.

Процес стає простішим при роботі з кількісними критеріями, такими як час відповіді на запит, вартість модифікацій, обсяг пам'яті, час на створення, вартість реорганізації. Ускладнення може виникнути через протиріччя між цими критеріями.

Тим часом, існує багато критеріїв оптимальності, які представляють собою невимірні властивості, складно виражаються у кількісному вигляді або у формі цільової функції.

До якісних критеріїв можуть відноситися гнучкість, адаптивність, доступність для нових користувачів, сумісність з іншими системами, можливість конвертації в інше обчислювальне середовище, можливість відновлення, розподілу та розширення.

Так, процес проектування баз даних є складним і тривалим завданням, яке часто займає кілька місяців. MongoDB представляє новий підхід до створення баз даних, де не використовуються таблиці, схеми, запити SQL, зовнішні ключі та інші елементи, характерні для традиційних реляційних баз даних.

Основна відмінність MongoDB полягає в тому, що вона пропонує модель даних, орієнтовану на документи. Це дозволяє прискорити роботу бази даних, покращити її масштабованість і зробити її використання більш простим.

Проте, навіть при урахуванні недоліків традиційних баз даних та переваг MongoDB, важливо розуміти, що кожне завдання може потребувати свого підходу. У деяких випадках MongoDB може дійсно покращити продуктивність вашої програми, зокрема, коли потрібно зберігати складну за структурою інформацію. Проте, в інших ситуаціях краще використовувати традиційні реляційні бази даних. Також можна використовувати комбінований підхід: зберігати один тип даних у MongoDB, а інший тип даних — у традиційних базах даних.

MongoDB володіє можливістю представляти не лише одну базу даних на одному фізичному сервері. Її функціонал дозволяє розміщувати кілька баз даних на різних фізичних серверах, при цьому ці бази даних можуть легко обмінюватися даними і зберігати цілісність.

JSON (JavaScript Object Notation) є популярним стандартом для обміну даними та їх подання. MongoDB використовує формат даних, що подібний до JSON, який називається BSON (Binary JSON). BSON дозволяє працювати з даними швидше, що робить пошук та обробку ефективнішими, хоча може займати більше місця у порівнянні з JSON.

MongoDB, написана на C++, має можливість легко переноситися на різні платформи, такі як Windows, Linux, MacOS, Solaris. Її вихідний код також доступний для компіляції, але рекомендується використовувати бінарні файли з офіційного сайту.

Система зберігання даних в MongoDB базується на наборі реплік. Цей набір включає основний вузол і набір вторинних вузлів. Усі вторинні вузли підтримують цілісність і автоматично оновлюються відповідно до змін, що відбуваються на основному вузлі. У разі відмови основного вузла один з вторинних вузлів автоматично стає новим основним для продовження роботи системи.

MongoDB відрізняється від традиційних реляційних баз даних тим, що в ньому відсутня жорстка схема, що суттєво полегшує роботу з базами даних та їх масштабуванням. Зміни в концепції зберігання даних не вимагають перестворення схеми, що заощаджує час розробників і виключає необхідність перебудови складних запитів.

Проблема зберігання даних великого обсягу виникає при роботі з будь-якими системами баз даних. Для зберігання таких даних можна використовувати файли та різні мови програмування. Деякі системи управління базами даних (СУБД), наприклад, MySQL, пропонують спеціальні типи даних, такі як BLOB, для зберігання бінарних об'єктів.

У відміну від реляційних СУБД, MongoDB дозволяє зберігати різноманітні документи з різними наборами даних, але обмеження розміру документу становить 16 МБ. Тим не менше, MongoDB пропонує рішення - технологію GridFS, що дозволяє зберігати дані, розмір яких перевищує 16 МБ.

GridFS включає дві колекції. У першій колекції, files, зберігаються імена файлів та їх метадані, такі як розмір. У другій колекції, chunks, дані файлів зберігаються у вигляді невеликих сегментів, зазвичай розміром по 256 КБ.

Отже, для такої бази даних ми можемо мати колекції: familyBudget, transactions, users. Кожна з них може містити документи з власними наборами даних, а при необхідності GridFS дозволить ефективно зберігати файли, розмір яких перевищує обмеження у 16 МБ.

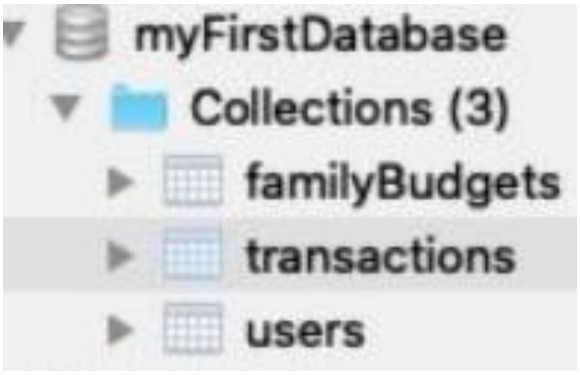


Рис. 2.3. Структура бази даних

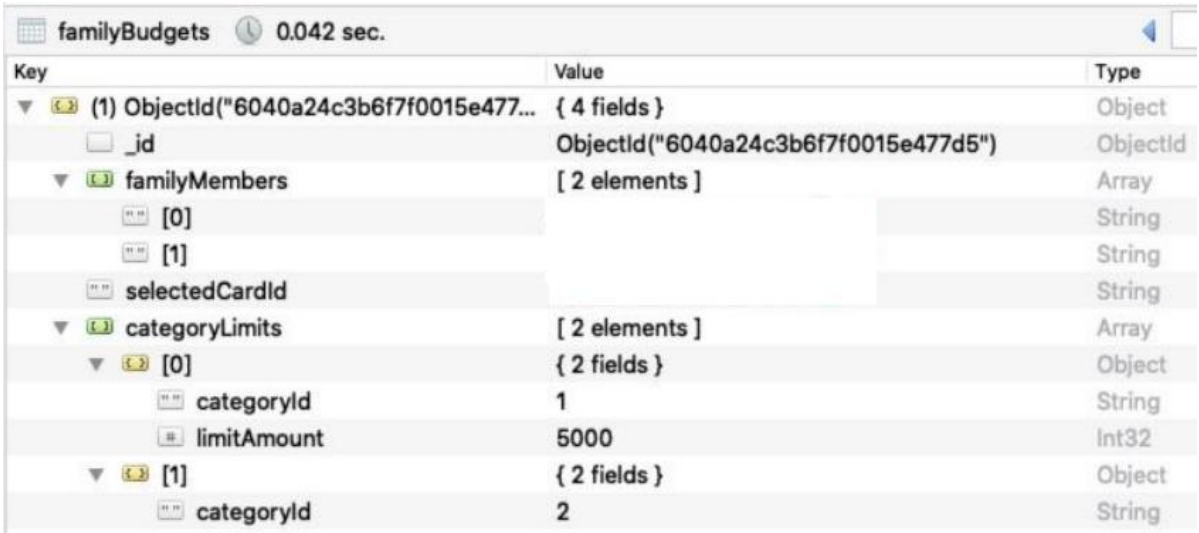


Рис. 2.4. Колекція familyBudget

У колекції "familyBudget" у MongoDB ми можемо зберігати інформацію про сімейний бюджет, включаючи членів сім'ї, обрану карту та встановлені ліміти на даний рахунок. Вибірка документу з цієї колекції може мати такий вигляд у вигляді JSON-подібної структури:

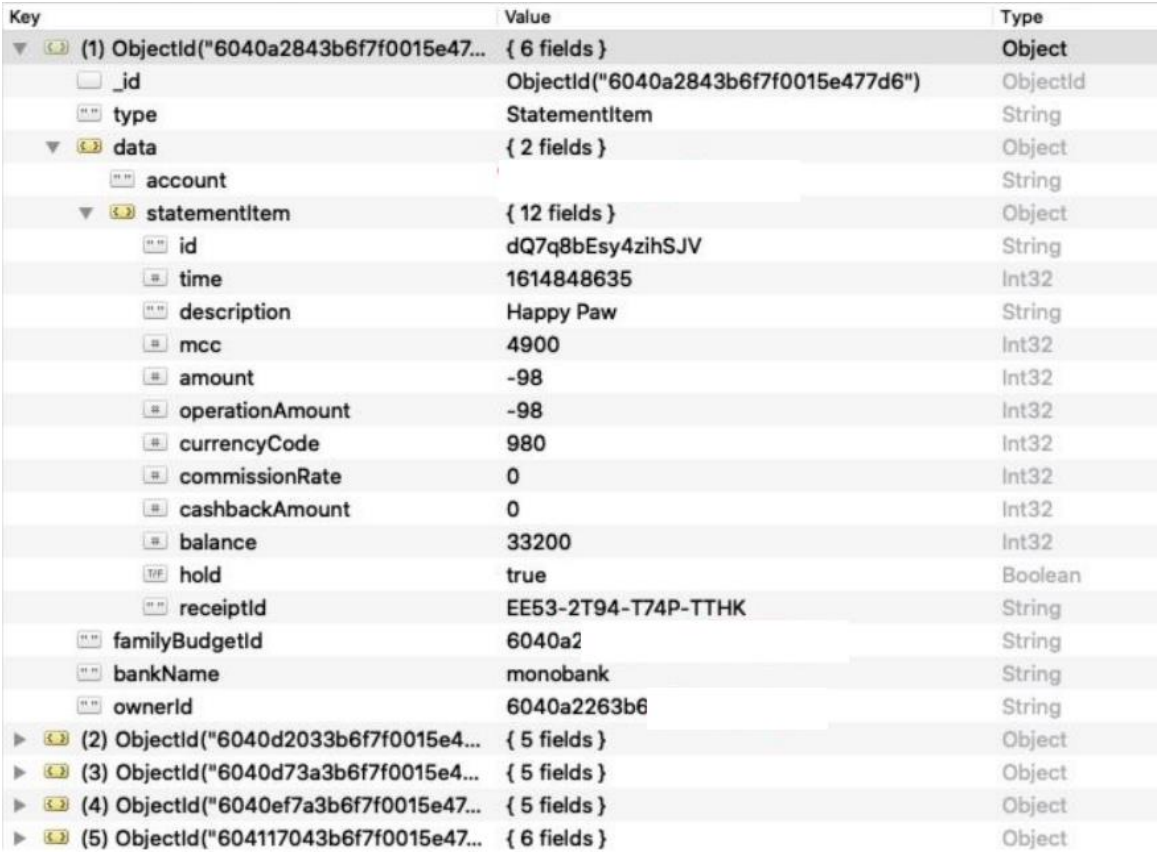
{

```
"_id": ObjectId("60a51f292a9a3e45f8661b8d"),
"familyName": "Сім'я Іванових",
"members": [
  {
    "name": "Іван Іванов",
    "age": 40,
    "role": "голова сім'ї"
  },
  {
    "name": "Марія Іванова",
    "age": 38,
    "role": "дружина"
  },
  {
    "name": "Олексій Іванов",
    "age": 15,
    "role": "син"
  },
  {
    "name": "Анна Іванова",
    "age": 10,
    "role": "дочка"
  }
],
"selectedCard": {
  "cardType": "кредитна",
  "cardNumber": "**** **** **** 1234",
  "expiryDate": "12/25"
},
"limits": {
```

```
    "dailyLimit": 1000,
    "monthlyLimit": 25000
  }
}
```

Тут ми маємо основні дані про сім'ю Іванових. Перераховані члени сім'ї з їхніми особистими даними (ім'я, вік, роль у сім'ї). Також є вказівка на обрану карту з її типом, номером та строком придатності. Крім цього, вказані обмеження на рахунок, такі як денний та щомісячний ліміти на витрати.

Це лише один із можливих форматів збереження інформації в колекції "familyBudget" в MongoDB, і його можна налаштувати відповідно до конкретних потреб і структури даних програми чи додатку.



Key	Value	Type
(1) ObjectId("6040a2843b6f7f0015e47...	{ 6 fields }	Object
_id	ObjectId("6040a2843b6f7f0015e477d6")	ObjectId
type	StatementItem	String
data	{ 2 fields }	Object
account		String
statementItem	{ 12 fields }	Object
id	dQ7q8bEsy4zihSJV	String
time	1614848635	Int32
description	Happy Paw	String
mcc	4900	Int32
amount	-98	Int32
operationAmount	-98	Int32
currencyCode	980	Int32
commissionRate	0	Int32
cashbackAmount	0	Int32
balance	33200	Int32
hold	true	Boolean
receiptId	EE53-2T94-T74P-TTHK	String
familyBudgetId	6040a2	String
bankName	monobank	String
ownerId	6040a2263b6	String
(2) ObjectId("6040d2033b6f7f0015e4...	{ 5 fields }	Object
(3) ObjectId("6040d73a3b6f7f0015e4...	{ 5 fields }	Object
(4) ObjectId("6040ef7a3b6f7f0015e47...	{ 5 fields }	Object
(5) ObjectId("604117043b6f7f0015e47...	{ 6 fields }	Object

Рис 2.5. Колекція transactions

У колекції "transactions" в MongoDB можна зберігати дані про кожну транзакцію з детальною інформацією, яка описує цю операцію. Нижче наведений

приклад структури документу, що містить інформацію про транзакцію у форматі JSON:


```
{
  "_id": ObjectId("60a51f292a9a3e45f8661b8e"),
  "transactionId": "TXN123456789",
  "type": "purchase",
  "userAccount": "user123",
  "timestamp": "2023-12-14T10:30:00",
  "description": "Online shopping at XYZ Store",
  "MCCCode": "7531",
  "amount": 120.50,
  "currencyCode": "USD",
  "commission": 2.00,
  "cashback": 5.00,
  "accountBalance": 2500.00,
  "receiptId": "RCPT789",
  "budgetId": "BUDG456",
  "bankName": "ABC Bank",
  "ownerId": "owner789"
}
```


У цьому прикладі ми маємо код що описує одну транзакцію. Поля документу включають:

- **_id**: унікальний ідентифікатор документу.
- **transactionId**: ідентифікатор транзакції.
- **type**: тип транзакції (наприклад, покупка, переказ, зняття готівки тощо).
- **userAccount**: аккаунт користувача, який здійснив транзакцію.
- **timestamp**: час здійснення транзакції.
- **description**: опис транзакції.
- **MCCCode**: код категорії мерчанта.
- **amount**: сума транзакції.

- **currencyCode**: код валюти.
- **commission**: комісія за транзакцію.
- **cashback**: сума кешбеку.
- **accountBalance**: баланс рахунку після транзакції.
- **receiptId**: ідентифікатор чеку.
- **budgetId**: ідентифікатор бюджету.
- **bankName**: назва банку.
- **ownerId**: ідентифікатор власника.

Ця структура може бути адаптована відповідно до конкретних вимог системи, і вона дозволяє зберігати докладну інформацію про кожну окрему транзакцію в базі даних MongoDB.

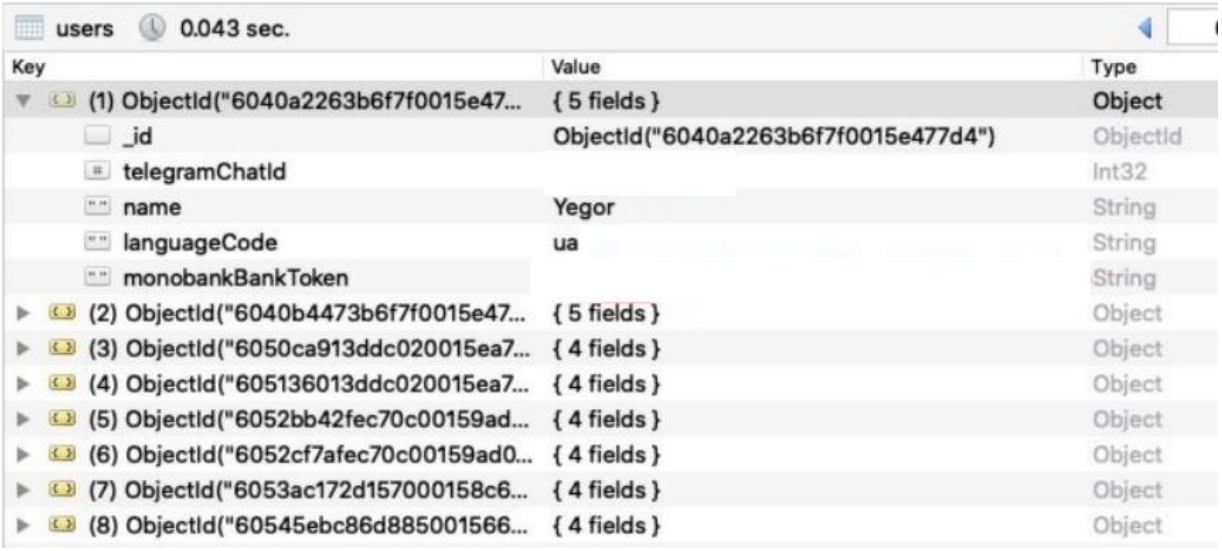


Рис. 2.6. Колекція users

Оскільки MongoDB базується на документно-орієнтованій структурі, для збереження даних про користувачів у колекції "users" можна використовувати наступну структуру документу у форматі JSON:

```
{
  "_id": ObjectId("60a51f292a9a3e45f8661b8f"),
  "userId": "user123",
  "telegramChatId": "123456789",
  "username": "JohnDoe",
```

```
  "languageCode": "en",
  "monobankToken": "YOUR_MONOBANK_TOKEN_HERE"
}
```

У цьому прикладі ми маємо наступні поля:

- **_id**: унікальний ідентифікатор документу користувача.
- **userId**: ідентифікатор користувача.
- **telegramChatId**: ідентифікатор чату в Telegram.
- **username**: ім'я користувача.
- **languageCode**: мовний код користувача (наприклад, "en" для англійської мови).
- **monobankToken**: токен Monobank для користувача, який може використовуватися для доступу до даних банківських операцій через API Monobank.

Ця структура документу в колекції "users" дозволяє зберігати різні дані про кожного користувача, включаючи їхні ідентифікатори, ім'я, мовний код та необхідні дані для використання сервісу Monobank. Треба замінити "YOUR_MONOBANK_TOKEN_HERE" на реальний токен Monobank для конкретного користувача, якщо це потрібно.

2.5. Контроль змін та версій за допомогою GitHub

Git є розподіленою системою керування версіями, яка дозволяє розробникам ведення контролю над змінами у своїх проектах шляхом зберігання та відстеження різниць між версіями файлів. Основні концепції Git полягають у можливості робити зміни в локальному репозиторії, створювати гілки для розробки нових функцій чи виправлення помилок, а потім об'єднувати ці зміни з основною гілкою.

Порівняно з попередніми централізованими системами контролю версій, у Git кожен розробник має свій власний повноцінний клон репозиторію, що дозволяє працювати над проектом навіть без підключення до центрального сервера. Кожен клон має повну історію змін, що дозволяє виконувати різноманітні операції навіть офлайн.

Коли розробник готовий поділитися своїми змінами, він комітує їх у власний локальний репозиторій, а потім може зберегти їх на центральному сервері за допомогою команди push. Інші розробники також можуть оновити свої локальні репозиторії, щоб отримати ці зміни, використовуючи команду pull.

Git забезпечує ефективну співпрацю між багатьма розробниками та дозволяє ефективно керувати проектами будь-якого типу, не обмежуючись лише програмним кодом, що робить його популярним інструментом для спільної роботи над будь-якими видами проектів.

Маккалоу та Поллак у своїх поясненнях розкривають переваги GitHub у контексті співпраці та спільної роботи над проектами. Вони висвітлюють, що GitHub, хоча базується на Git, надає додаткові можливості та інтерфейс для більш зручної роботи з репозиторіями.

Основним функціоналом GitHub є можливість створювати "форки" проектів, що дозволяє копіювати сховище з одного облікового запису в інший. Це дозволяє розробникам створювати свої версії проектів та вносити зміни, не впливаючи на оригінальний проект. Після внесення змін власникам форка можна відправити "запит на витяг", що дозволяє об'єднати зміни з оригінальним проектом. Ця функціональність спрощує процес взаємодії та співпраці між розробниками, що дозволяє швидше та зручніше вносити зміни у проекти.

Додатково, GitHub надає графічний інтерфейс для роботи з репозиторіями, а також функції контролю доступу, управління завданнями та вікі для кожного проекту, що полегшує співпрацю та координацію роботи над проектами різної складності.

Ці функції (форки, запити на витяг та об'єднання змін) відображають силу GitHub, дозволяючи розробникам участь у проектах з відкритим кодом швидше та ефективніше, ніж це було можливо раніше.

Цитований уривок демонструє важливість соціальної складової GitHub як спільноти розробників, що сприяє співпраці, взаємодії та відображенню репутації кожного учасника. Дозвілля учасників обговорювати, коментувати та об'єднувати

зміни дозволяє створювати продуктивне середовище для співпраці над проектами, навіть якщо вони використовують командний рядок для взаємодії з репозиторіями.

Платформа GitHub відіграє значну роль у сприянні розвитку проектів з відкритим кодом та спільноти розробників в цілому. Забезпечуючи зручний інтерфейс для співпраці, обговорень та об'єднання змін, GitHub сприяє зростанню та популяризації багатьох відкритих проектів.

Крім того, GitHub використовується як платформа для співпраці не лише серед розробників з відкритим кодом, але і для команд, що працюють у комерційних проектах. Приватні сховища та локальні екземпляри програмного забезпечення GitHub також надають можливості співпраці та зручний інтерфейс для управління розробкою проектів.

Усі ці функції та можливості GitHub сприяють розвитку спільноти розробників, створюючи доступну та продуктивну платформу для співпраці та розвитку різних видів програмного забезпечення.

Компанія Atlassian, яка придбала BitBucket та запустила Stash, входить у сферу розробки програмного забезпечення з різними інструментами співпраці для розробників, такими як відстежування помилок Jira та wiki Confluence. Їхні продукти спрямовані на полегшення робочих процесів команд розробників, надаючи зручний інтерфейс для співпраці над проектами та управління різними аспектами розробки ПЗ.

GitHub, у свою чергу, став відомим як майданчик для розміщення та спільного використання коду. Система контролю версій Git, на якій ґрунтується GitHub, дозволяє детально відстежувати зміни у коді, надаючи можливість вивчати процеси розробки інших програмістів. Це сприяє вирішенню проблем, а також дозволяє користувачам вчитися та отримувати приклади коду від досвідчених розробників.

Отже, обидві компанії - Atlassian і GitHub - зосереджуються на певних аспектах розробки програмного забезпечення, надаючи інструменти для співпраці та управління проектами. Хоча Atlassian сконцентрована на приватних та локальних хостингах засобів розробки, а GitHub відомий як майданчик для спільного

використання коду та навчання через приклади, конкуренція між ними може стати стимулом для подальшого розвитку їхніх функцій та послуг у майбутньому.

Ось кроки для створення репозиторію і налаштування користувача в Git:

- Створення репозиторію: Відкрийте командний рядок (термінал) і перейдіть у кореневу папку вашого проекту. Введіть команду git init, щоб ініціалізувати новий репозиторій Git у поточній папці проекту.

  bashCopy code

  git init

- Налаштування користувача: Для того, щоб Git знаходив ваші дані про авторство при збереженні змін, ви можете налаштувати ім'я користувача та електронну адресу.

Для налаштування імені користувача, ви можете використати команду:

bashCopy code

git config --global user.name "Ваше_ім'я"

Якщо ви хочете встановити електронну адресу, то введіть команду:

bashCopy code

git config --global user.email "ваша_електронна_адреса@example.com"

Команда --global вказує, що ці дані будуть застосовуватися для всіх репозиторіїв на вашому комп'ютері. Якщо ви хочете вказати ім'я користувача або електронну адресу тільки для певного репозиторію, використовуйте ці команди без параметра --global.

Після виконання цих команд, Git буде використовувати вказані ім'я користувача та електронну адресу для кожного коміту, який ви робите в цьому репозиторії.

Висновки до розділу 2

У розділі 2 нашої дипломної роботи, що фокусується на створенні чат-боту для автоматизації контролю витрат, ми провели всебічний аналіз та реалізацію ключових складових системи. Головна увага була приділена вимогам до системи, її структурі, інтерфейсу, реалізації основних модулів, а також на процесі тестування розробленого веб-додатку.

Начало формы

У розділі 2.1. ми провели оцінювання різноманітних форм втілення дозволило обрати оптимальну стратегію, що враховує ключові параметри та сприяє кращій реалізації поставлених завдань.

У розділі 2.2. ми визначилися, що використання API виявилося ефективним способом комунікації між різними програмними компонентами, сприяючи швидкій та надійній обміну даними.

У розділі 2.3. ми дізнались, що використання Express фреймворку виявилося дієвим методом для швидкої та гнучкої розробки веб-сервісів, дозволяючи ефективно управляти різноманітними запитами і відповідями.

Розділ 2.4. дослідження унікальних аспектів створення бази даних виявило ключові принципи оптимізації, які сприятимуть ефективній роботі системи.

У розділі 2.5. GitHub виявився незамінним інструментом для контролю версій та спільної роботи, дозволяючи зберігати, контролювати та здійснювати зміни в коді, що сприяє зручності та систематизації розробки.

У цілому, цей розділ відображає глибокий та всебічний підхід до розробки чат-боту, з основним акцентом на його функціональності, надійності, безпеці та зручності в користуванні. Отримані результати на даному етапі будують міцну підставу для наступних кроків у проекті та його успішної впровадження в роботу.

РОЗДІЛ 3
АНАЛІЗ НАУКОВОГО ДОСЛІДЖЕННЯ

3.1. Структура проекту

Почнемо з розгляду структури файлів.

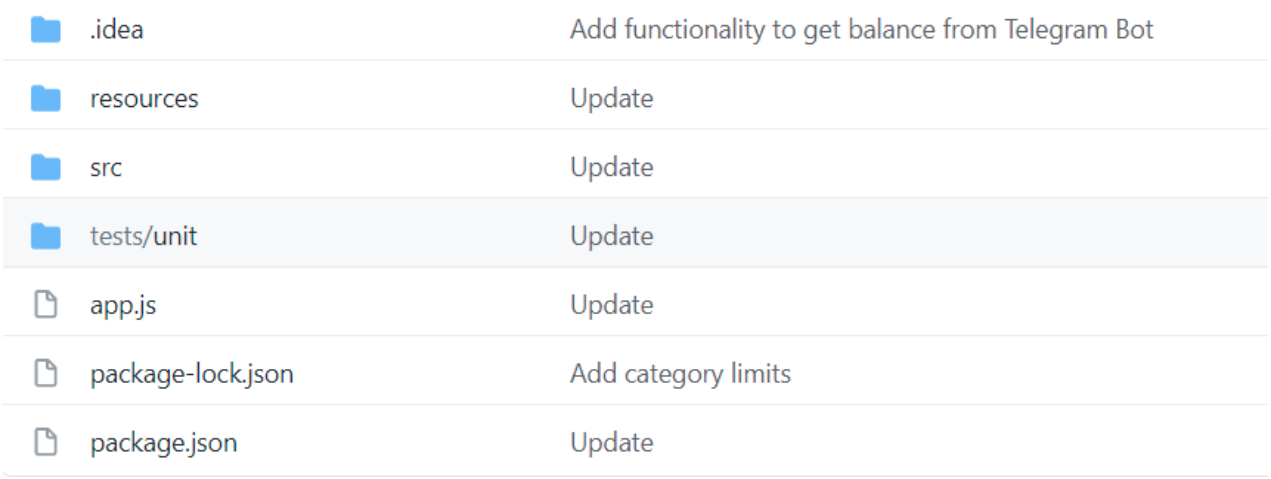


Рис. 3.1. Структура файлів і тек

Папка .idea зберігає інформацію про налаштування середовища, що використовується для написання коду — Intellij Idea. Це не обов'язково і не стосується роботи з чат-ботами, але це важливо при роботі з різних комп'ютерів.

Заповнення файлу app.js:

```
require('dotenv').config();
const path = require('path');
const { I18n } = require('i18n');
const useServer = require('./src/server');
const useTelegramBot = require("./src/telegramBot");
const useDBGateway = require("./src/dbGateway");
const useAPIGateway = require("./src/apiGateway");
```

```
const useBotGateway = require("./src/botGateway");

if (!process.env.PORT) {

    throw TypeError('PORT variable is required')
}

if (!process.env.TELEGRAM_BOT_TOKEN) {
    throw TypeError('TELEGRAM_BOT_TOKEN variable is required')
}

if (!process.env.BASE_URL) {
    throw TypeError('BASE_URL variable is required')
}

if (!process.env.MONGO_URI) {
    throw TypeError('MONGO_URI variable is required')
}

if (!process.env.MONGO_DB_NAME) {
    throw TypeError('MONGO_DB_NAME variable is required')
}

(async () => {
    const i18n = new I18n({
        locales: ['en', 'ru', 'ua'],
        directory: path.join(__dirname, 'resources', 'locales')
    });
    const dbGateway = await useDBGateway();
```

```
  const apiGateway = await useAPIGateway();
  const telegramBot = await useTelegramBot({dbGateway, apiGateway, i18n});
  const botGateway = await useBotGateway({telegramBot});

  useServer({
    telegramBot, dbGateway, botGateway, i18n, apiGateway, });
})();
```

Отже, цей код є відправною точкою для усього проекту: ініціюються модулі, активується сервер і перевіряються змінні оточення.

Далі – package.json:

```
{
 "name": "fbos",
 "version": "1.0.0",
 "main": "index.js",
 "scripts": {
  "start": "node app.js"
 },
 "repository": {
  "type": "git",
  "url": "git+https://github.com/yegorHeiz/fbos.git"
 },
 "author": "",
 "license": "ISC",
 "bugs": {
  "url": "https://github.com/yegorHeiz/fbos/issues"
 },
 "homepage": "https://github.com/yegorHeiz/fbos#readme",
 "description": "",
 "dependencies": {
```

```
    "axios": "^0.21.0",
    "body-parser": "^1.19.0",
    "dotenv": "^8.2.0",
    "express": "^4.17.1",
    "i18n": "^0.13.2",
    "joi": "^17.3.0",
    "lodash": "^4.17.20",
    "moment": "^2.29.1",
    "mongodb": "^3.6.3",
    "node-telegram-bot-api": "^0.50.0",
    "numeral": "^2.0.6"
  },
  "devDependencies": {
    "chai": "^4.2.0",
    "mocha": "^8.2.1"
  }
}
```

Тут можна побачити всі бібліотеки, які використовуються, а також залежності для розробки (devDependencies), які не потрапляють на продакшен.

У папці tests/unit знаходяться тести для боту, які охоплюють основну технічну частину. Деякі функції також були протестовані користувачем.

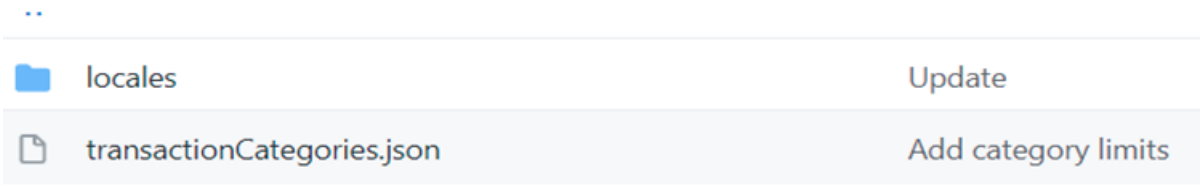


Рис. 3.2. Вміст теки resources

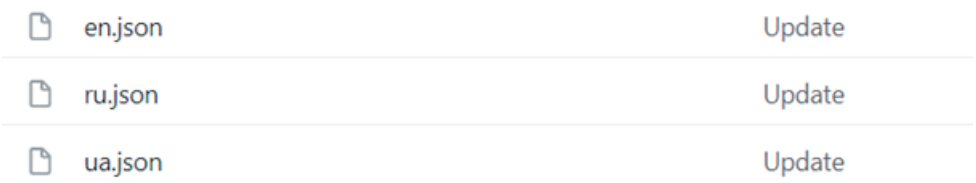


Рис. 3.3. Вміст теки locales

У папці locales знаходяться файли локалізації для бота на трьох мовах (Рис. 3.3.), а також файл з класифікацією MCC-кодів за категоріями витрат.

```
"cafeAndRestaurants": {
   "id": "2",
   "title": "cafeAndRestaurants",
   "mccCodes": [
     5811,
     5812,
     5813,
     5814
   ]
 },
 "cinema": {
   "id": "3",
   "title": "cinema",
   "mccCodes": [
     7829,
     7832,
     7841
   ]
 },
```

Приклади локалізаційй:

"cafeAndRestaurantsShort": "Кафе і ресторани",

"cinema": "Кіно. Послуги та товари кінотеатрів, оренда і купівля товарів в спеціалізованих магазинах",

"cinemaShort": "Кіно",

"taxi": "Таксі. послуги таксі",

"taxiShort": "Таксі",

"medicine": "Краса і медицина. Товари та послуги в масажних або косметичних салонах, і аптеки",

"medicineShort": "Краса і медицина",

Далі до папки src з основним кодом.

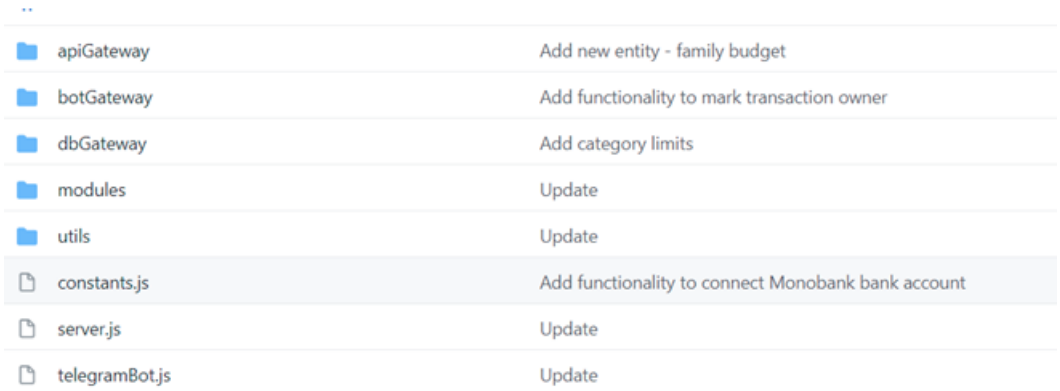


Рис. 3.4. Вміст теки src

У каталозі apiGateway розміщений модуль, через який проходять всі запити до Monobank. Тут ми маємо запити на дані користувача та налаштування веб-хуку для отримання інформації від Monobank на наш сервер.

```
const axios = require('axios');


const MONOBANK_API_BASE_URL = 'https://api.monobank.ua';


module.exports = () => {
   return {
      users: {
         getUserInfo: async bankToken => {
```

```
            const url = `${MONOBANK_API_BASE_URL}/personal/client-info`;

            const response = await axios.get(url, {headers: {'X-Token': bankToken}});
            return response.data;
        },
        setupWebhook: async (bankToken, bankName, familyBudgetId) => {
            const url = `${MONOBANK_API_BASE_URL}/personal/webhook`;
            const webHookUrl =
`${process.env.BASE_URL}/webhooks/familyBudgets/${familyBudgetId}/banks/${bank
Name}`;

            const response = await axios.post(url, {webHookUrl}, {headers: {'X-Token':
bankToken}});
            return response.data;
        }
    }
  }
};
```

Після цього у нас є botGateway, призначений для взаємодії з чат-ботом - відправка повідомлень у відповідний чат.

```
const {TELEGRAM} = require('./../constants');

module.exports = ({telegramBot}) => {
  const sendMessage = (chatID, text, options = {}) => {
    return telegramBot.sendMessage(chatID, text, options)
  };

  return {
    [TELEGRAM]: {
      sendMessage       }
```

```
}};
```

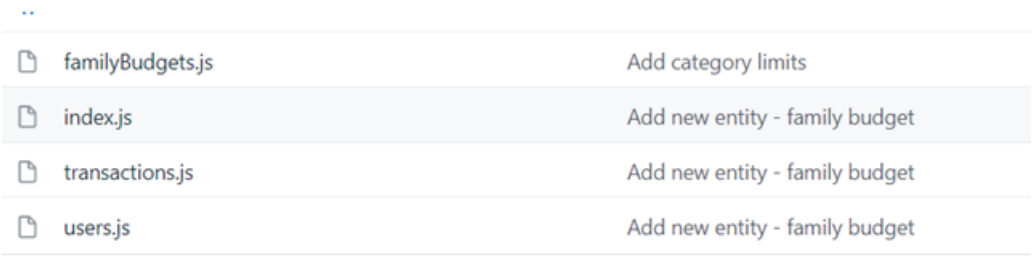


Рис. 3.5. Вміст теки dbGateway

Початковим файлом для модуля бази даних є index.js, в якому здійснюється завантаження та експорт користувачів, транзакцій та даних про сімейний бюджет.

```
const {MongoClient} = require('mongodb');

const useUsers = require('./users');
const useTransactions = require('./transactions');
const useFamilyBudgets = require('./familyBudgets');

module.exports = async () => {
  const client = await MongoClient.connect(process.env.MONGO_URI);
  const db = client.db(process.env.MONGO_DB);

  return {
    users: useUsers({db}),
    transactions: useTransactions({db}),
    familyBudgets: useFamilyBudgets({db}),
  }
};
```

У файлі familyBudgets.js є функція оновлення колекції familyBudgets або створення нового запису в ній та встановлення ліміту. Також можна отримати ідентифікатор власника картки або члена родини.

```
const _ = require('lodash');
```

```
const {ObjectID} = require('mongodb');
module.exports = ({db}) => {
  const familyBudgetsCol = db.collection('familyBudgets');

  const upsertFamilyBudget = async (familyMemberId, selectedCardId) => {
    const query = {
      $or: [
        {selectedCardId},
        {familyMembers: familyMemberId}
      ]
    };

    const existingFamilyBudget = await familyBudgetsCol.findOne(query);

    if (existingFamilyBudget) {
      await familyBudgetsCol.updateOne(
        query,
        {
          $set: {
            familyMembers: _.uniq([...existingFamilyBudget.familyMembers, familyMemberId]),
            selectedCardId
          }
        }
      );
      return familyBudgetsCol.findOne(query);
    }

    const {insertedId} = familyBudgetsCol.insertOne({
```

```
        familyMembers: [familyMemberId],
        selectedCardId
    });

    return familyBudgetsCol.findOne({_id: insertedId})
};

const getById = id => {
    return familyBudgetsCol.findOne({_id: ObjectID(id)})
};

const getByFamilyMemberId = familyMemberId => {
    return familyBudgetsCol.findOne({familyMembers: familyMemberId})
};

const setCategoryLimit = async (id, categoryId, limitAmount) => {
    const familyBudgetBeforeUpdate = await familyBudgetsCol.findOne({_id:
ObjectID(id)});

    await familyBudgetsCol.updateOne({_id: ObjectID(id)}, {
        $set: {
            categoryLimits: [
                ...(familyBudgetBeforeUpdate.categoryLimits || []).filter(x => x.categoryId
!== categoryId),
                {categoryId, limitAmount}
            ]
        }
    });
```

```
    return familyBudgetsCol.findOne({_id: ObjectID(id)});
  };
  return {upsertFamilyBudget, getById, getByFamilyMemberId, setCategoryLimit};
};
```

Аналогічно, для операцій з транзакціями (створення, оновлення, отримання транзакцій по ідентифікатору сім'ї) та для користувачів існують відповідні модулі, що реалізують подібний підхід.

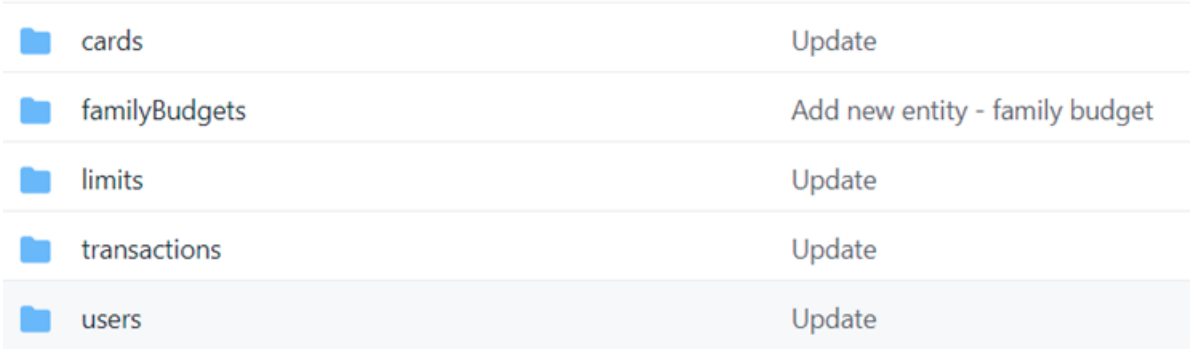


Рис. 3.6. Вміст теки modules

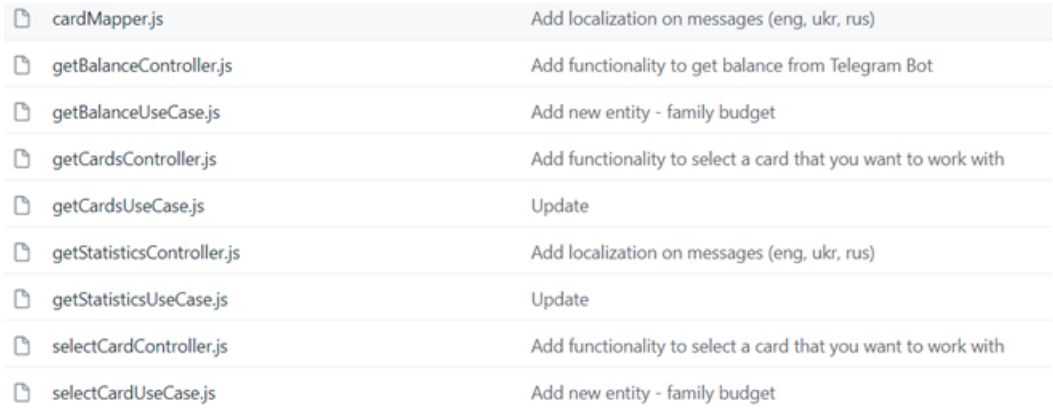


Рис. 3.7. Вміст теки cards

У каталозі cards файл cardMapper.js містить об'єкт, який перетворює карту в рядок.

Файл getBalanceUseCase.js знаходить користувача, його сім'ю, інформацію про користувача, знаходить карту, обрану для ведення бюджету, і надсилає номер цієї карти в Telegram.

Файл getCardsUseCase знаходить користувача, його інформацію і локалізує дані для підключення з ботом.

Файл getStatisticsUseCase.js повертає статистику користувачеві в чат.

Файл selectCardUseCase.js надсилає повідомлення про те, що карта була обрана, та оновлює цю інформацію в базі даних.

Усі файли з суфіксом 'Controller' викликають функції з такою самою назвою.

Модуль familyBudget (Рис. 3.6) додає нові дані або оновлює їх у колекції.

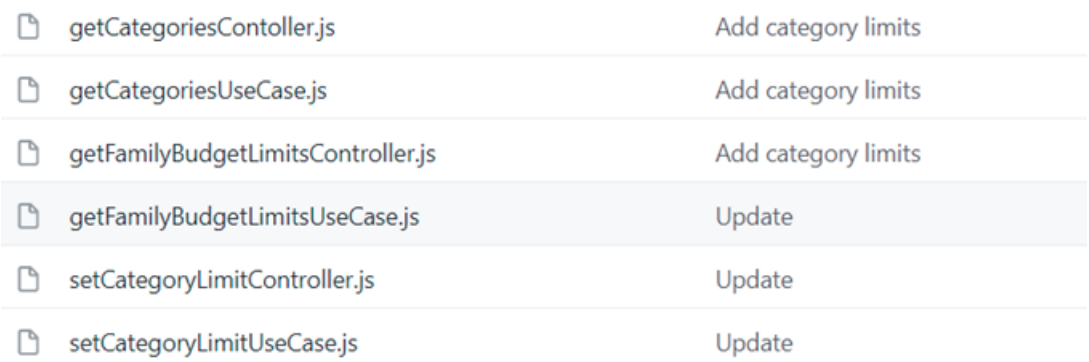


Рис. 3.8. Вміст теки limits

Модуль limits використовується для контролю лімітів на обрані категорії.

Файл getCategoriesUseCase.js використовується для розпізнавання назв категорій та передачі даних в Telegram.

Файл getFamilyBudgetLimitsUseCase.js повертає користувачеві всі його існуючі ліміти в чат.

Файл getCategoryLimitUseCase.js використовується для встановлення лімітів на обрану категорію.

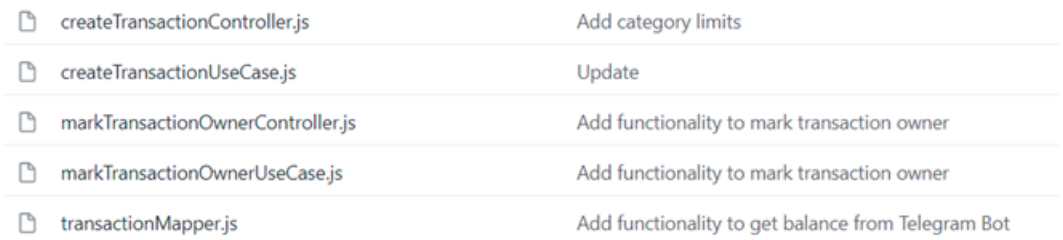


Рис. 3.9. Вміст теки transactions

Файл createTransactionUseCase.js використовується для створення транзакції та передачі даних про неї в чат з користувачем.

Файл markTransactionOwnerUseCase.js використовується для визначення того, хто її провів - один із членів родини або спільно.

Файл transactionMapper.js перетворює об'єкт транзакції у рядкове значення.

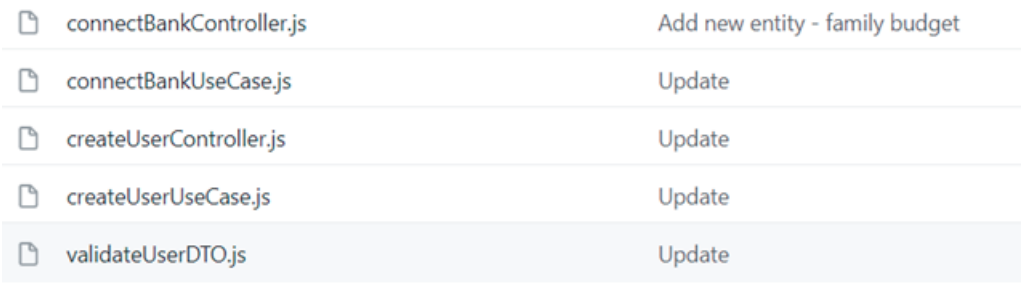


Рис. 3.10. Вміст теки users

Модуль users використовується для підключення до банку (connectBankUseCase.js) та створення запису користувача (createUserUseCase.js). Файл validateUserDTO.js використовується для передачі об'єкту між модулями та його валідації.

```
const Joi = require('joi');

const schema = Joi.object({
   telegramChatId: Joi.number().required(),
   name: Joi.string().required(),
   languageCode: Joi.string().required()
});
module.exports = data => {
   const {error, value} = schema.validate(data);

   if (error) {
      throw error;
   } else {
```

```
    return value;
}};
```

dateToUnix.js	Add functionality to display spends by owners
getCategory.js	Add category limits
getCategoryById.js	Add category limits
getCategoryByMcc.js	Add category limits
getCurrencySignByCode.js	Add functionality to get balance from Telegram Bot
getCurrentMonthDates.js	Add category limits
getLanguageCode.js	Update
getLimitsFromTransactions.js	Update
getOr.js	Add functionality to send transaction messages to Telegram Bot

Рис. 3.11. Вміст теки utils

У каталозі utils зібрані корисні функції, такі як ініціалізація категорій за різними даними, ініціалізація валюти, мови, поточного місяця, а також лімітів на здійснені транзакції.

3.1. Проведення тестування розробленого чат-боту

Огляд роботи розробленого проекту:

Рис. 3.12. Стартова ознайомча інформація

При відкритті чат-бота FBOS вперше, ви отримаєте коротке введення до основних можливостей та переваг цього бота.

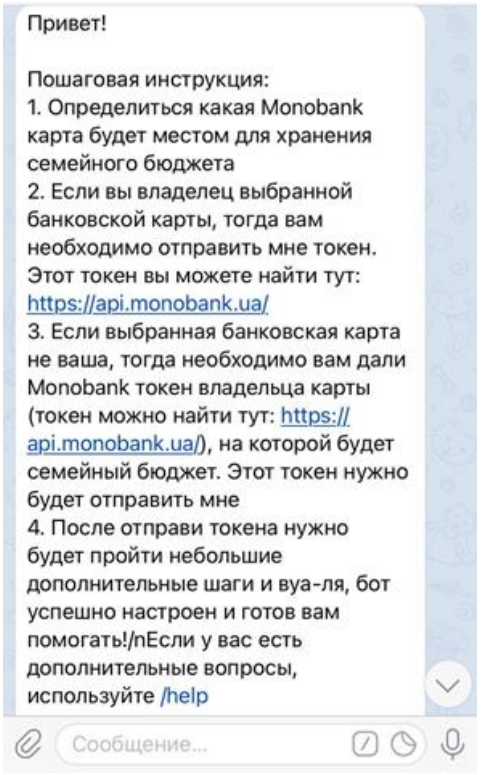


Рис. 3.13. Пошагова інструкція для підключення

Користувач зможе легко підключити свій рахунок, оскільки бот надає чітку та просту інструкцію з необхідними посиланнями.

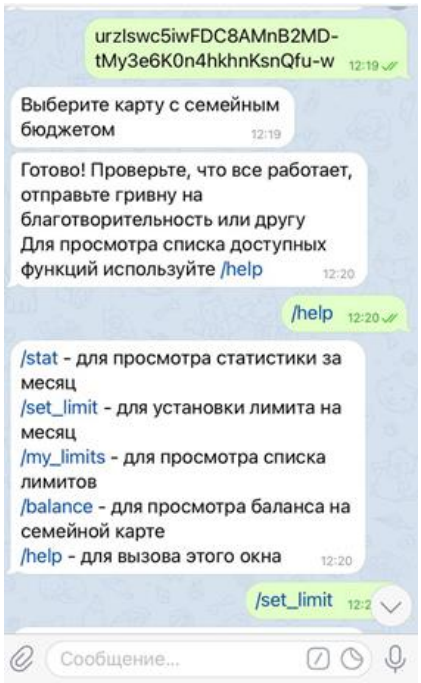


Рис.3.14. Етап під’ єднання банківської карти

Після того, як токен отримано, бот видає список доступних банківських карт, з яких є можливіст вибрати одну для ведення бюджету. Після цього доступні такі команди (функції): перегляд статистики за місяць, встановлення лімітів на категорію, перегляд списку встановлених лімітів, перегляд поточного балансу рахунку та виклик цього вікна.

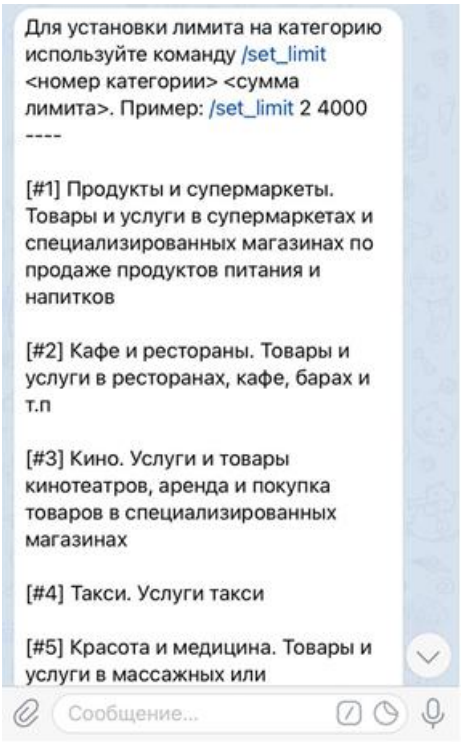


Рис. 3.15. Етап встановлення грошових лімітів

Після виклику команди /set_limit ми отримуємо вікно з повідомленням, де можна переглянути усі доступні категорії та отримати інструкцію з встановлення лімітів.

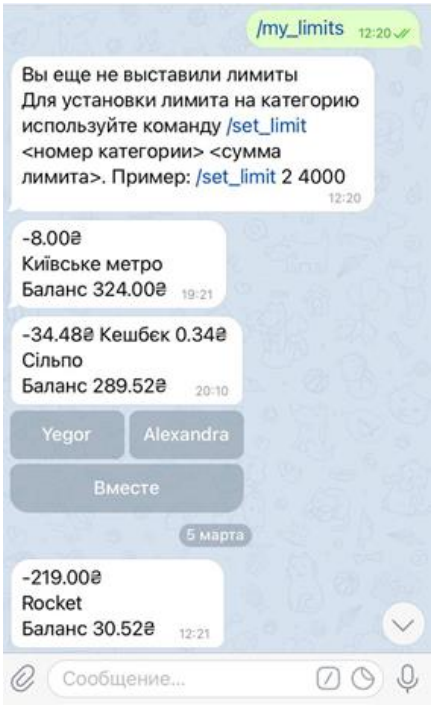


Рис. 3.16. Етап перегляд грошових лімітів та список транзакцій

Доступна команда перегляду встановлених лімітів та визначення того, хто зробив транзакцію. Користувач може самостійно вказати хто витратив кошти.

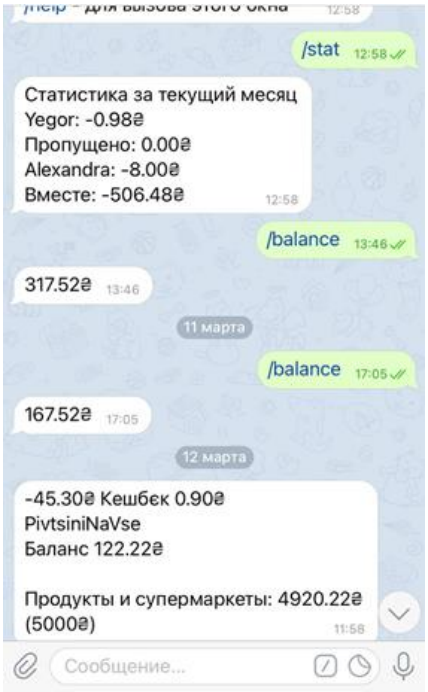


Рис.3.17. Етап перегляду підсумкової статистики

На Рис. 3.17 можна ознайомитися з тим, як виглядає статистика за місяць та команда перевірки балансу.

Висновки до розділу 3

У третьому розділі розглянуто практичні аспекти впровадження та використання розробленого чат-боту для автоматизації контролю витрат клієнтів банку.

У підрозділі 3.1. ми показали, що структура проекту відображає чітко визначену організацію та розподіл ресурсів для реалізації чат-боту. Вона надає основу для ефективного управління завданнями, розвитку, та координації дій між учасниками команди. Чітко визначені ролі та взаємодія між різними модулями проекту дозволяють забезпечити організовану та структуровану розробку, що є важливим кроком до успішної реалізації задач.

У підрозділі 3.2. ми провели тестування чат-боту та виявили його високу ефективність, стабільність та відповідність вимогам. Ретельна перевірка різних аспектів функціональності, включаючи безпеку, надійність та відповідність вимогам користувача, дозволила виявити та виправити потенційні проблеми ще на етапі розробки. Це створило підґрунтя для стабільної та надійної роботи чат-боту в реальних умовах використання.

Результати третього розділу створюють практичну базу для успішного впровадження та застосування чат-боту, що дозволить максимізувати ефективність автоматизації контролю витрат у користувачів Monobank.

Таким чином, поєднання теоретичних та практичних аспектів дослідження створює надійну основу для створення та використання ефективного рішення з автоматизації банківської системи.

ВИСНОВКИ

Під час роботи над дипломним проектом було успішно спроектовано та розроблено чат-бот на платформі Telegram для ефективного управління та організації фінансових операцій. Цей чат-бот, створений для Monobank, включає в себе широкий спектр функцій:

- **Керування та встановлення лімітів**: Користувач може зручно керувати фінансами, встановлюючи обмеження для різних категорій витрат.
- **Розподіл витрат та оновлення про транзакції**: Система дозволяє розподіляти витрати та отримувати оновлення щодо кожної транзакції, що сприяє більш ефективному контролю фінансів.
- **Автоматизоване редагування інформації про витрати**: Чат-бот автоматизує процес редагування та ведення інформації про витрати, що спрощує роботу користувача та підвищує точність даних.
- **Надання інформації про витрати за певний період**: Користувач може легко отримати детальну інформацію про свої витрати за визначений період часу.
- **Мобільність версії**: Враховуючи платформу Telegram, чат-бот має мобільну версію, що робить його доступним та зручним для користувачів з різних пристроїв та місць розташування.

Для розробки цього проекту використовувалися сучасні технології та засоби, зокрема платформа Telegram, API Monobank та система контролю версій GitHub. Результатом цієї роботи стало створення потужного та зручного інструменту для управління фінансами, який може використовуватися широким колом користувачів для оптимізації їхньої фінансової поведінки та контролю витрат.

СПИСОК БІБЛІОГРАФІЧНИХ ПОСИЛАНЬ

1. Thinkingin Java. [Електронний ресурс] – Режим доступу: https://www.livelib.com/book/1000236176-thinking-in-java-bruce-eckel (дата звернення 12.11.23р) – Назва з екрана.

2. Design Goalsof the Java TM Programming Language. [Електронний ресурс] – Режим доступу: https://cs.gmu.edu/~eclab/projects/mason/ (дата звернення 10.05.23р) – Назва з екрана.

3. Пасічник В. В. Організація баз даних та знань / В. В. Пасічник, В. А. Резніченко. – К.: Видавнича група BHV, 2006. – 384 с.

4. Different Isn't Always Better, But Better's Always Different. [Електронний ресурс] – Режим доступу:https://jonathanischwartz.wordpress.com/2007/08/30/different-isnt-always-better-but-betters-always-different (дата звернення 15.11.23р) – Назва з екрана.

5. Feigenbaum, Barry. SWT, Swingor AWT: Which is right for you? [Електронний ресурс] – Режим доступу: https://www.programmersought.com/article/7105203895/ (дата звернення 16.11.23р) – Назва з екрана.

6. Г. Гарсиа-Молина Системы баз данных / Д. Ульман, Д. Уидом. Полный курс. – М.: Вильямс, 2003, 1088 с.

7. J. Gosling, B. Joy, G. Steele, G. Brachda. The Java Language Specification [Електронний ресурс]. – Режим доступу: https://www.computerworld.com/au/ (дата звернення 16.11.23р) – Назва з екрана.

8. Different Isn't Always Better, But Better's Always Differ Google win scrucial API ruling, Oracle's case decimated. [Електронний ресурс] – Режим доступу: https://arstechnica.com/tech-policy/2012/05/google-wins-crucial-api-ruling-oracles-case-decimated (дата звернення 17.11.23р) – Назва з екрана.

9. The Java History Time line. [Електронний ресурс] – Режим доступу: https://support.edubrite.com/oltpublish/site/document.do?dispatch=show&id=070e1788-014a-102d-a6f6-00301b4433e5 (дата звернення 20.11.23р) – Назва з екрана.