КИЇВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ
ІМЕНІ ТАРАСА ШЕВЧЕНКА
Факультет комп'ютерних наук та кібернетики
Кафедра теорії та технології програмування

Кваліфікаційна робота
на здобуття ступеня бакалавра
за спеціальністю 122 Комп'ютерні науки
на тему:

СИСТЕМА ОБЛІКУ ФІНАНСІВ

Виконав студент 4-го курсу
СЕМІТКІН Михайло Сергійович
(підпис)

Науковий керівник:
доцент, кандидат технічних наук
Ткаченко Олексій Миколайович
(підпис)

Засвідчую, що в цій роботі немає запозичень з праць інших авторів без відповідних посилань.

Студент
(підпис)

Роботу розглянуто й допущено до захисту на засіданні кафедри Теорії та технологій програмування
«___»_____ 2023 р., протокол № __

Завідувач кафедри
М. С. Нікітченко
_____
(підпис)

Київ — 2023

РЕФЕРАТ

Загальний обсяг роботи: п’ятдесят шість сторінок. Робота містить двадцять вісім рисунків, шість таблиць, чотирнадцять посилань, чотири додатки. Основна частина викладена на сорок одній сторінці.

Ключові слова: TELEGRAM, АВТОМАТИЗАЦІЯ, БЮДЖЕТ, ОБЛІК ВИТРАТ, РОЗРОБКА, ТРЕКЕР ВИТРАТ, ФІНАНСИ, ЧАТ-БОТ

Об’єкт дослідження: грошові витрати, зроблені у готівковому, безготівковому чи будь-якому іншому способі, а також процес їх обліку.

Предмет дослідження: програмні системи, призначені для обліку витрат.

Мета роботи: розробка Telegram боту для обліку грошових витрат з метою надання користувачам зручного і ефективного інструменту для контролю, аналізу своїх витрат, зменшення часу, що витрачається на облік витрат, а також збереження їх історії.

Інструменти розробки: мова програмування Java, Spring Boot Framework, база даних PostgreSQL, СКБД pgAdmin 4, бібліотека для управління міграціями Flyway, інструменти роботи з даними Spring Data JPA та Spring JdbcTemplate, система контролю версій GitHub, платформа Digital Ocean, система збирання проекту Gradle, інтегроване середовище розробки IntelliJ IDEA Ultimate.

ЗМІСТ

СКОРОЧЕННЯ ТА УМОВНІ ПОЗНАЧЕННЯ

API — application programming interface, прикладний програмний інтерфейс;

CSV — comma-separated values, значення, розділені комою;

IDE — Integrated Development Environment, середовище для розробки програмного забезпечення;

JAR — Java ARchive, Java-архів;

JDBC — Java DataBase Connectivity, з'єднання з базами даних на Java

JPA — Java Persistence API, стандартизований інтерфейс для Java ORM фреймворків;

LTS — Long Term Support, підтримка протягом тривалого періоду;

ORM — Object-relational mapping, об'єктно-реляційна проекція;

SSH — Secure SHell, безпечна оболонка;

UML — Unified Modeling Language, уніфікована мова моделювання;

БД — база даних;

ПК — персональний комп’ютер.

СКБД — система керування базами даних

ВСТУП

Контроль над власними фінансами є актуальною задачею в будь-якому суспільстві. Користувачі все частіше шукають зручні та ефективні інструменти, що допоможуть їм контролювати свої гроші, планувати бюджет та вести облік витрат. Ведення обліку особистих витрат є важливим кроком досягнення фінансової грамотності для кожної людини. Сьогодні є багато джерел витрати коштів: готівкові витрати, купівлі у онлайн магазинах, банківські перекази, регулярні безготівкові платежі — усе це ускладнює облік витрат, саме тому ручний облік у паперовому вигляді давно відійшов на другорядний план і все популярнішими стають цифрові системи для обліку.

Метою даного проекту є створення системи, що надає можливість зручно зберігати інформацію про повсякденні витрати користувачів, допоможе отримати уявлення про мінімальну необхідну суму для прожитку, допоможе виявити категорії з найбільшою витратою коштів, а також більш ефективно розподіляти фінанси у майбутньому.

Завданням даної роботи є аналіз існуючих на ринку систем для обліку фінансів, проектування власної системи для збереження даних про витрати користувачів та розробка Telegram чат боту з використанням сучасних технологій розробки для зручної взаємодії з нею.

Об’єктом дослідження даної роботи є витрати користувачів, здійснені готівковим, безготівковим чи будь-яким іншим способом та процес їх обліку.

Предметом дослідження є програмні системи, призначені для обліку витрат, їх переваги та недоліки. Дослідження включає в себе визначення функціональних вимог, проектування архітектури системи, встановлення і налаштування необхідного середовища, тестування роботи та впровадження системи.

Інструментами розробки є мова програмування Java, Spring Boot Framework, база даних PostgreSQL, СКБД pgAdmin 4, бібліотека для управління міграціями Flyway, інструменти роботи з даними Spring Data JPA та Spring JdbcTemplate, система контролю версій GitHub, платформа Digital Ocean, система збирання проекту Gradle, інтегроване середовище розробки IntelliJ IDEA Ultimate.

РОЗДІЛ 1. ОГЛЯД НАЯВНИХ НА РИНКУ СИСТЕМ

1.1 Money Lover

Money Lover — мобільний застосунок, підтримує такі операційні системи як IOS та Android, також доступний на ПК. У додатку є облік витрат і доходів, імпорт даних в Excel, спільне ведення бюджету. Є безкоштовна версія, проте вона має обмежений функціонал: недоступна допомога техпідтримки, експорт транзакцій в Excel (для Android) або CSV (для iOS), а також присутня реклама. Преміум версія коштує $2,49/міс. Додаток доступний українською та англійською мовами.

Переваги:

– мультиплатформенність;

– широкий арсенал функцій;

– спільне ведення бюджету;

– можливість планування бюджету.

Недоліки:

– експорт доступний тільки у платній версії;

– процес запису транзакції довгий і вимагає кілька кроків (рис. 1.1);

– відсутність можливості додати власні категорії витрат;

– дані не зберігаються в разі видалення програми та їх неможливо відновити без попередньо створеної резервної копії, більш детально дана проблема висвітлена у статті [1].

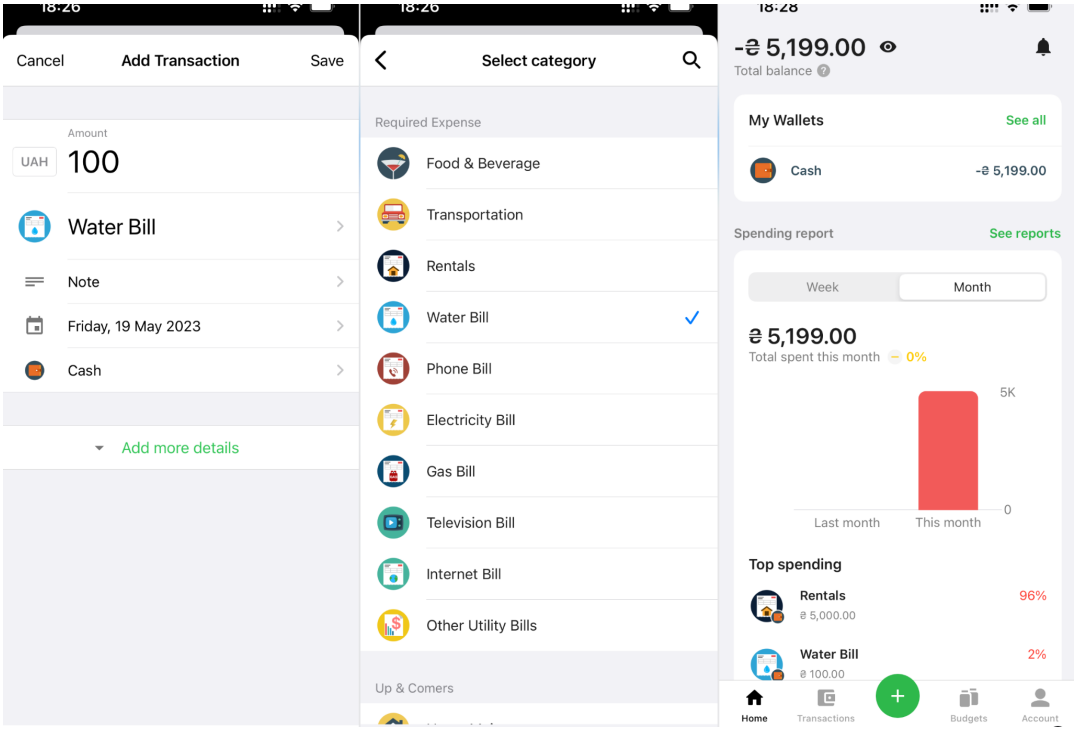


Рисунок 1.1 — Процес збереження транзакції у застосунку Money Lover

1.2 Goodbudget

Американський додаток, що дозволяє аналізувати витрати, завантажувати транзакції в CSV, а також планувати бюджет, для цього користувачу пропонується для кожної категорії створити "конверт" та вказати ліміт витрат на місяць. Дані про витрати автоматично підтягуються до десктопної версії Goodbudget.

Переваги:

– можливість встановлювати ліміти витрат на категорії та отримувати сповіщення про досягнення його;

– звіти доступні безпосередньо у застосунку у безкоштовній версії (рис. 1.2);

Недоліки:

– для користування системою обов’язкова реєстрація. Відсутня можливість реєстрації та входу через соціальні мережі;

– система має дещо застарілий та не інтуїтивний інтерфейс (див. рис. 1.3);

– система доступна виключно англійською мовою;

– щоб додати транзакцію до категорії, її потрібно попередньо створити, що ускладнює процес створення транзакцій (див. рис. 1.4);

– сервіс доступний тільки англійською мовою

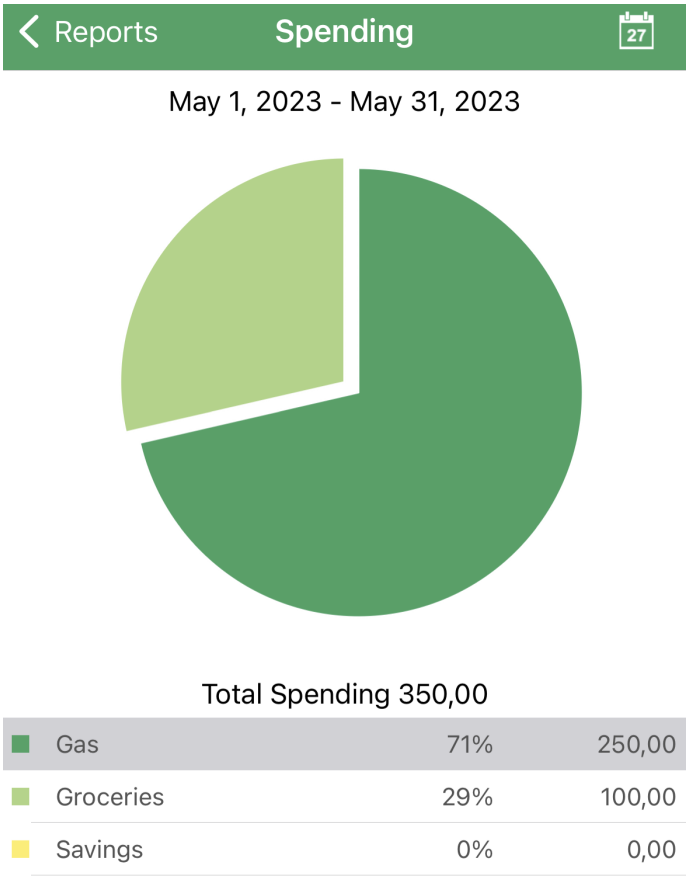


Рисунок 1.2 — Звіт по витратах за місяць

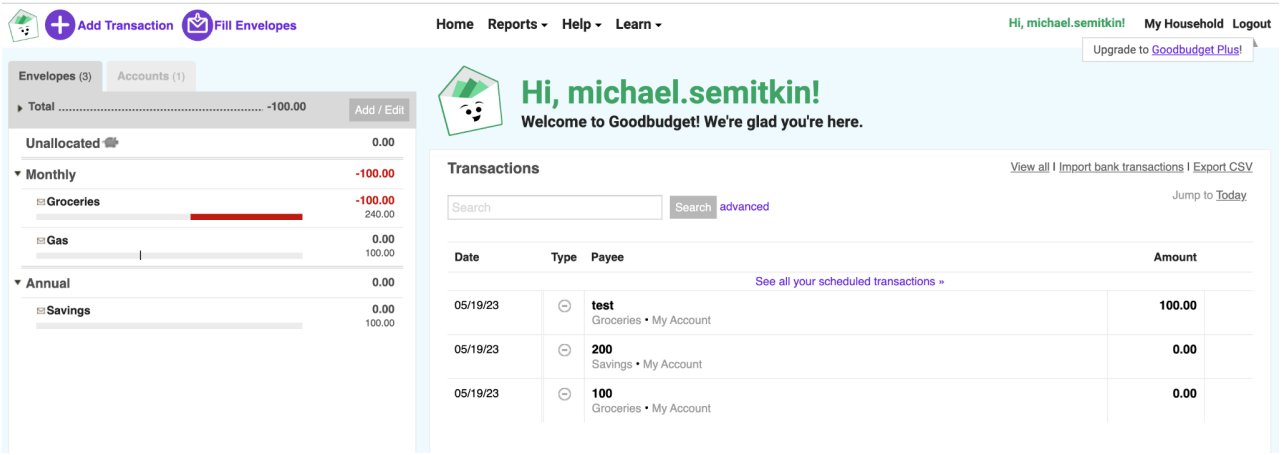


Рисунок 1.3 — Головна сторінка веб версії GoodBudget

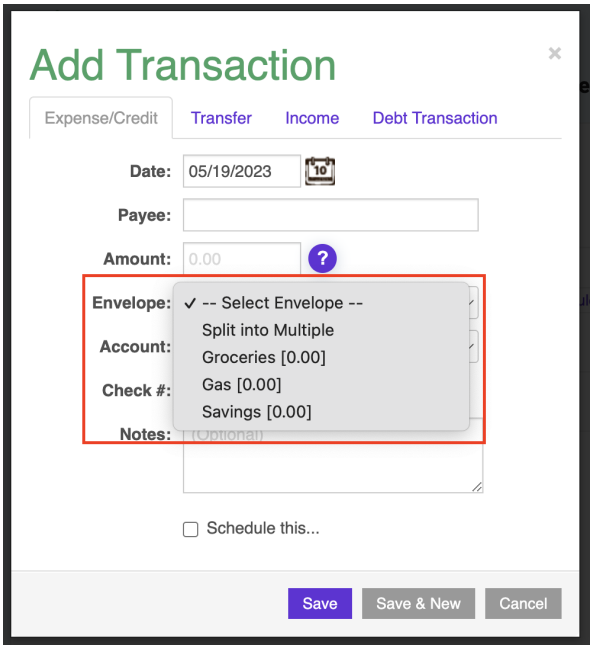


Рисунок 1.4 — Вікно створення транзакції

1.3 Monefy

Monefy — виключно мобільний застосунок, що доступний на операційних системах IOS та Android. Має безкоштовну та преміум версії ($70/рік)

Переваги:

– зручний інтерфейс (рис. 1.5);

– можливість переглядати транзакції за будь-який період.

Недоліки:

– відсутність версії WEB та ПК версій застосунку;

– створення власних категорій тільки у платній версії;

– синхронізація між девайсами тільки у платній версії;

– у безкоштовній версії дані про транзакції не зберігаються у разі видалення програми з телефону;

– відсутність безкоштовного пробного періоду преміум версії;

– відсутність функцій імпорту та експорту даних.

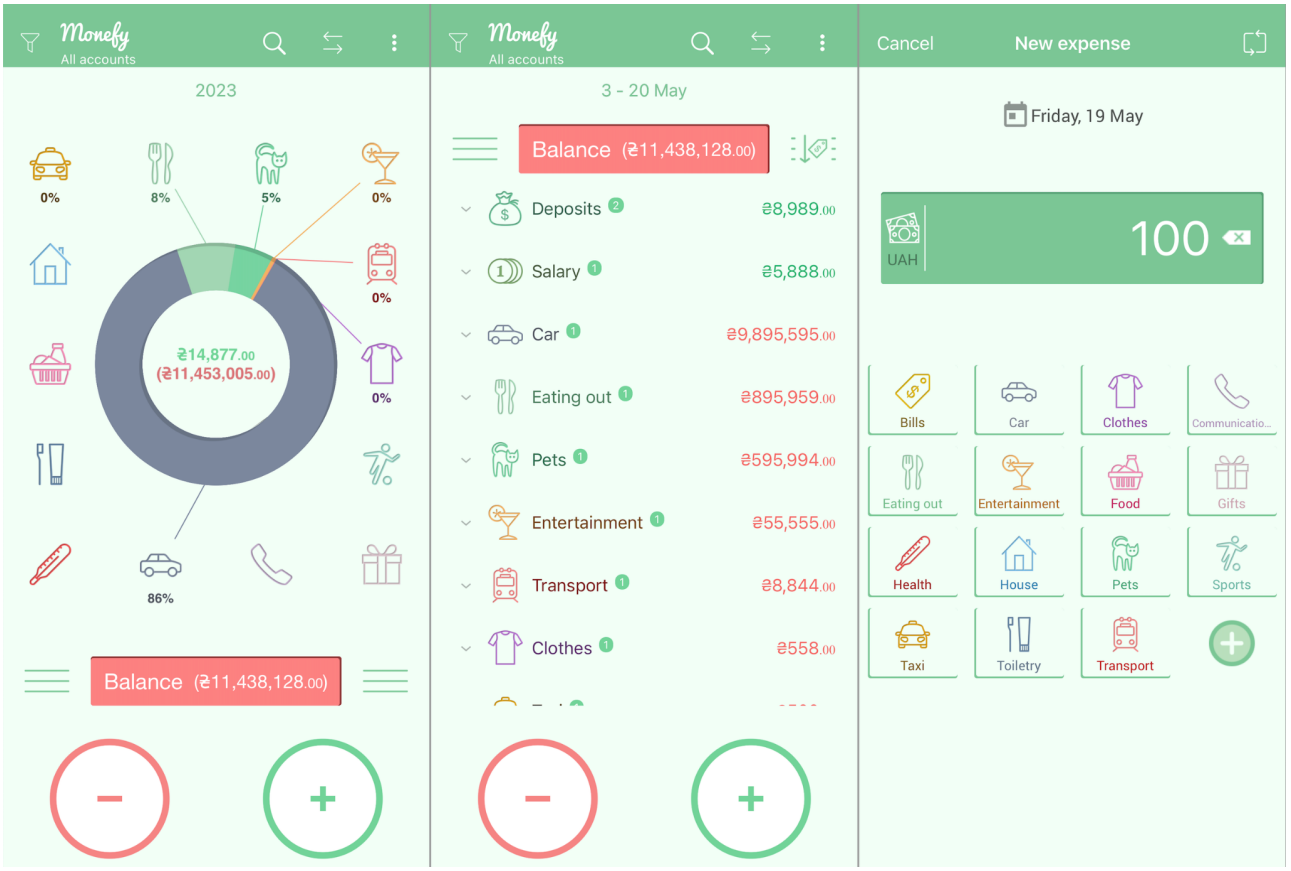


Рисунок 1.5 — Інтерфейс Monefy

1.4 ExpenseBot

ExpenseBot — альтернативний інструмент для відстеження витрат у вигляді Telegram чат-боту [2]. Взаємодія з ботом відбувається через команди, що мають спеціальний формат. Наприклад, щоб зафіксувати витрату у категорії “їжа”, потрібно ввести наступну команду: “/new 100 їжа” (рис. 1.6), де число означає розмір витрати. Бот дозволяє переглядати список витрат за вказаний місяць аа також фільтрувати їх по категоріях, експортувати транзакції за вказаний місяць.

Переваги:

– система з відкритим вихідним кодом [2];

– можливість запустити екземпляр чат-боту на власному сервері, що унеможливлює витік конфіденційних даних;

– можливість створення власних категорій витрат;

– можливість створення регулярний щомісячних транзакцій.

Недоліки:

– складний інтерфейс взаємодії (рис 1.7);

– нестабільна робота;

– відсутність функції імпорту даних;

– інтерфейс виключно англійською мовою;

– відсутність можливості експортувати дані за минулий рік.

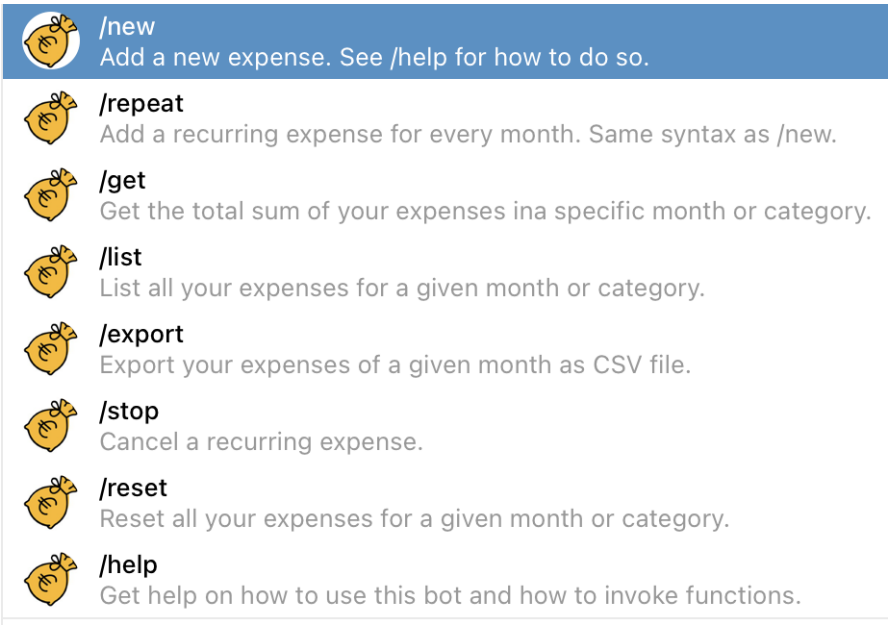


Рисунок 1.6 — Перелік доступних команд

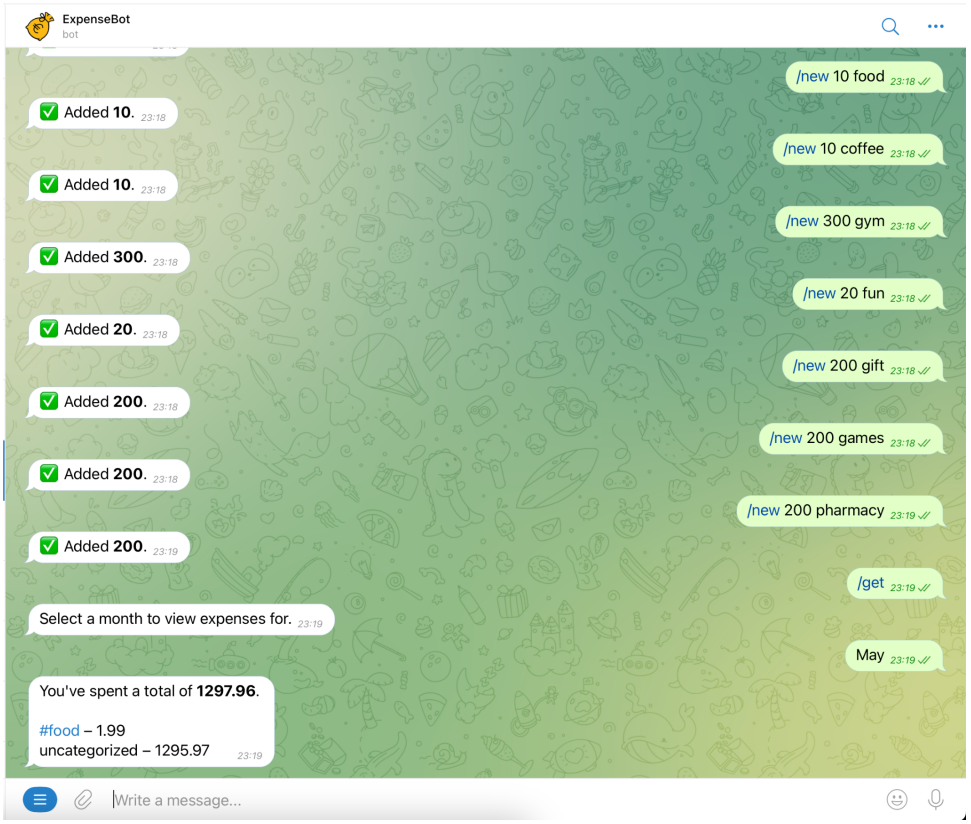


Рисунок 1.7 — Взаємодія з ExpenseBot

РОЗДІЛ 2. ПРИЗНАЧЕННЯ ТА ЦІЛІ СТВОРЕННЯ СИСТЕМИ

2.1 Призначення системи

Призначенням системи є автоматизація обліку витрат, реєстрації грошових витрат шляхом введення відповідних даних, таких як сума, дата та категорія витрати, створення детального журналу витрат і забезпечення точного обліку фінансових операцій, генерації звітів та статистики на основі введених даних про витрати, які дозволятимуть користувачам отримувати уявлення про загальну суму витрат, їх розподіл за категоріями, тренди, тощо, а також надаватимуть корисну інформацію для аналізу фінансового стану та прийняття обґрунтованих рішень.

2.2 Цілі створення системи

Цілями розробки системи є:

– забезпечити користувачам зручний інструмент для контролю та управління їх фінансами. Система повинна надати простий та інтуїтивно зрозумілий інтерфейс, який дозволить користувачам з легкістю реєструвати та відстежувати свої витрати;

– забезпечити облік витрат та можливість їх категоризації. Система повинна дозволяти користувачам встановлювати категорії для витрат, що допоможе у визначенні основних джерел витрат та аналізу їх розподілу;

– надати користувачам звіти та статистику щодо їх витрат. Система повинна здати генерувати звіти, які надають користувачам огляд загальної суми витрат, розподіл витрат за категоріями, тренди витрат тощо. Це допоможе користувачам аналізувати свої фінанси та приймати обґрунтовані рішення;

– допомогти користувачам планувати бюджет та керувати фінансовими ресурсами.

2.3 Вимоги до системи

Функціональні вимоги:

– можливість запису та категоризація витрат користувачів;

– можливість генерації звітів та статистики про витрати;

– підтримка кількох мов;

– дані повинні синхронізуватися одночасно між усіма девайсами, через які було авторизовано в систему.

Вимоги до інтерфейсу:

– інтуїтивність та легкість у використанні інтерфейс для взаємодії з системою;

– зручна навігація та можливість швидко здійснювати необхідні операції.

Вимоги до операційної системи:

– система повинна бути доступна на мобільних платформах та ПК.

Вимоги до продуктивності:

– система повинна забезпечувати швидкий відгук на дії користувачів;

– система повинна ефективно керувати ресурсами та запобігати витоку пам’яті

Інші нефункціональні вимоги:

– система повинно коректно реагувати на будь-які дії користувача та правильно опрацьовувати всі типи можливих виняткових ситуацій.

РОЗДІЛ 3. ОГЛЯД ВИКОРИСТАНИХ ТЕХНОЛОГІЙ

3.1 IntelliJ IDEA ULTIMATE

IntelliJ IDEA Ultimate — IDE від компанії JetBrains. Має інтуїтивний, зручний інтерфейс, вбудовану підсистему для взаємодії з БД, HTTP-клієнт, можливість редагування файлів більшості форматів та ін. Існує велика кількість плагінів [3], як безкоштовних, так і доступних за преміум підпискою, для розширення функціоналу IDE. Має безкоштовну, з дещо обмеженим функціоналом, та преміум версії. Існує 30-денний пробний період для преміум версії. Також, для студентів є можливість отримати безкоштовну підписку на період навчання. IntelliJ IDEA займає одну з лідерських позицій у загальному рейтингу популярності IDE станом на травень 2023 року за індексом PYPL (рис. 3.1) [4]. Також за інформацією онлайн-платформи Turing, IntelliJ IDEA займає перше місце у списку IDE для мови програмування Java [5]

Worldwide, May 2023 compared to a year ago:

Rank	Change	IDE	Share	Trend
1		Visual Studio	27.94 %	-1.4 %
2	↑	Visual Studio Code	13.59 %	+1.5 %
3	↓	Eclipse	11.73 %	-1.3 %
4	↑	pyCharm	8.92 %	+0.6 %
5	↓	Android Studio	8.91 %	+0.6 %
6		IntelliJ	6.83 %	+0.6 %
7		NetBeans	4.43 %	-0.7 %
8	↑↑↑	RStudio	3.53 %	+0.9 %
9	↑	Atom	3.31 %	+0.5 %
10	↓↓	Sublime Text	3.23 %	-0.4 %
11	↓↓	Xcode	2.75 %	-0.3 %
12		Code::Blocks	1.8 %	-0.4 %

Рисунок 3.1 — Рейтинг середовищ програмування

3.2 Мова програмування Java

Для реалізації програмної частини роботи було обрано мову програмування Java. Дана мова є кросплатформенною, суворо типізованою, об'єктно-орієнтованою мовою програмування із С-подібним синтаксисом, випущена у 1995 році. На сьогодні, її розробкою займається компанія “Oracle”. Перевагами цієї мови є:

– строга типізація;

– об'єктно-орієнтований підхід. Java базується на об'єктно-орієнтованому підході до програмування, що сприяє розподілу даних та функцій для роботи з ними у логічно об'єднані класи та спрощує керування кодом та розширення функціоналу в майбутньому

– кросплатформенність. Мається на увазі, що написана один раз програма на мові Java, буде працювати на будь-якій платформі без зміни в пофатковому коді;

– багатофункціональність. Java надає широкий набір інструментів та бібліотек для розробки різноманітних функцій. Це дозволяє ефективно реалізувати функціонал системи обліку витрат: робота з базами даних, обробка даних та інше.

– підтримка багатопоточності. Java підтримує багатопоточність, що дозволяє обробляти одночасно багато запитів і подій;

– автоматичне керування пам’яттю. Java використовує автоматичний збирач сміття (Garbage collector) для звільнення пам’яті від об’єктів, які уже не потрібні для програми;

– велика спільнота розробників. Java має велику та активну спільноту розробників, яка надає підтримку, документацію, різноманітні інструменти та сторонні бібліотеки. Це спрощує розробку та дозволяє швидше вирішувати проблеми

Станом на травень 2023-го року Java займає 3-тє місце у рейтингу мов програмування за індексом TIOBE (рис. 3.2) [6].

May 2023	May 2022	Change	Programming Language	Ratings	Change
1	1		Python	13.45%	+0.71%
2	2		C	13.35%	+1.76%
3	3		Java	12.22%	+1.22%
4	4		C++	11.96%	+3.13%
5	5		C#	7.43%	+1.04%
6	6		Visual Basic	3.84%	-2.02%
7	7		JavaScript	2.44%	+0.32%

Рисунок 3.2 — Рейтинг мов програмування

3.3 Spring Boot Framework

Spring Framework — один з найбільш популярних фреймворків для створення застосунків на Java. Spring Framework має модульну структуру, що у свою чергу дозволяє додавати до програми тільки необхідні модулі (рис 3.3) [7]. Перевагами даного фреймворку є:

– швидке налаштування: Spring Boot надає широкий спектр стандартних налаштувань та конфігурацій, що дозволяє швидко розпочати роботу над проектом. Він автоматично конфігурує багато аспектів додатку, що дозволяє зосередитись на розробці функціоналу;

– управління залежностями: Spring Boot надає зручний механізм управління залежностями за допомогою Maven або Gradle. Це спрощує використання сторонніх бібліотек та дозволяє легко оновлювати їх версії;

– моніторинг та адміністрація: Spring Boot надає зручний інтерфейс для моніторингу та керування додатком. За допомогою вбудованого актуатора (Actuator), можна отримати інформацію про стан додатку, метрики продуктивності та виконати різні адміністративні дії;

– розширюваність: Spring Boot дозволяє легко розширювати функціональність додатку шляхом додавання нових модулів та компонентів Spring.

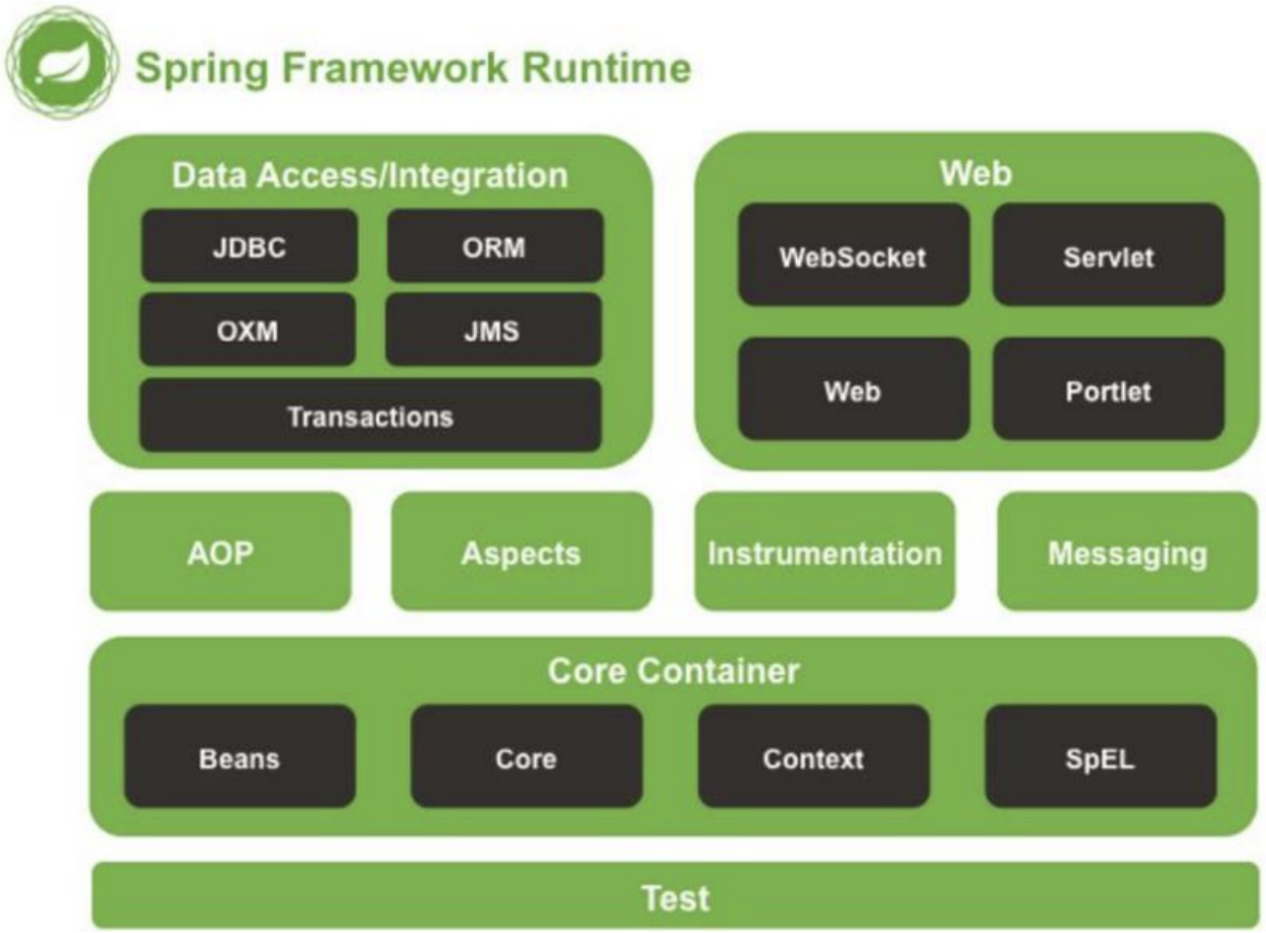


Рисунок 3.3 — Архітектура Spring Framework

Використання Spring Boot у розробці системи обліку витрат допомагає прискорити процес розробки, забезпечити стабільність та масштабованість додатку, а також використовувати потужну екосистему Spring для реалізації різноманітного функціоналу.

3.4 PostgreSQL

PostgreSQL — це потужна система керування базами даних (СКБД), яка пропонує надійне зберігання та обробку даних. Основні переваги використання PostgreSQL у розробці системи обліку витрат включають:

– надійність та цілісність даних: PostgreSQL забезпечує механізми транзакцій та відновлення після збоїв, що дозволяє зберігати дані в консистентному стані;

– можливість масштабування: PostgreSQL підтримує горизонтальне та вертикальне масштабування, що дозволяє збільшувати обсяг та продуктивність бази даних залежно від потреб проекту. Він також надає механізми реплікації для резервного копіювання та розподіленого доступу до даних;

– підтримка розширень та стандартів: PostgreSQL має велику кількість розширень та модулів, що дозволяють розширити функціональність бази даних згідно з потребами проекту. Він також підтримує стандарти SQL та деякі нереляційні функції, що спрощує розробку та інтеграцію з іншими системами;

– потужність та продуктивність: PostgreSQL пропонує різні оптимізації та індексування для покращення продуктивності запитів. Він також підтримує використання складних запитів, географічних даних та розширених типів даних.

– PostgreSQL не перший рік займає лідерські позиції в рейтингах популярності БД. За даними опитування розробників на порталі StackOverflow, 43.59% опитаних використовують PostgreSQL у повсякденній розробці або мають бажання використовувати у майбутньому (рис. 3.4) [8].

– PostgreSQL є проектом з відкритим кодом та будь-хто бажаючий може її використовувати безкоштовно.

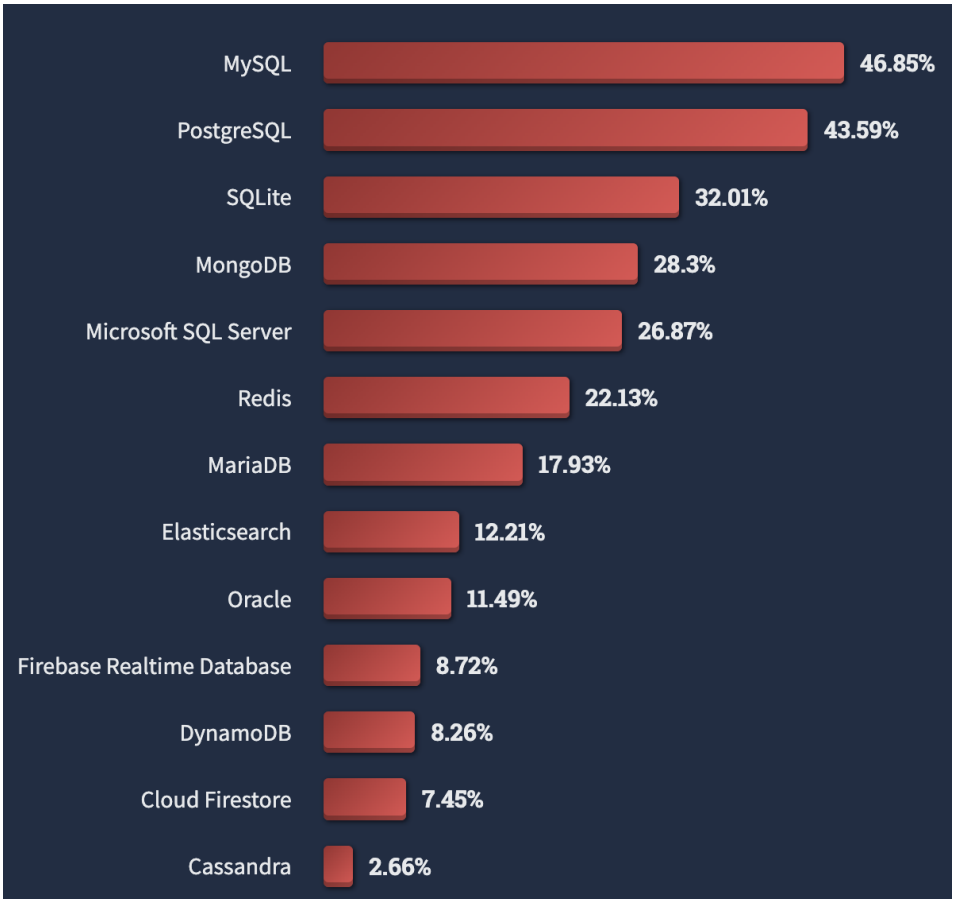


Рисунок 3.4 — Найпопулярніші бази даних

3.5 Flyway

Flyway — це відкрите програмне забезпечення для керування версіями бази даних. Воно дозволяє зручно вести міграцію бази даних при розробці програмного забезпечення.

Основні переваги використання Flyway у розробці системи обліку витрат включають:

– простий та зрозумілий синтаксис, що дозволяє легко створювати та керувати міграціями бази даних. Він інтегрується з SQL-скриптами та забезпечує автоматичне виконання міграцій при запуску додатку;

– Flyway дозволяє стежити за версіями бази даних та автоматично виконувати потрібні міграції для переходу від однієї версії до іншої. Це

забезпечує контроль за структурою бази даних та зручне впровадження змін;

– робота з командним рядком та інтеграція зі збірниками проектів: Flyway надає зручні інтерфейси командного рядка, що дозволяють виконувати міграції з командного рядка або включати їх у процес збирання проекту, що дає змогу автоматизувати процес міграції;

– підтримка різних баз даних: Flyway підтримує багато популярних баз даних, таких як PostgreSQL, MySQL, Oracle, Microsoft SQL Server та багато інших. Це дозволяє легко переносити систему обліку витрат на різні платформи та зберігати однорідність в управлінні базою даних.

Використання Flyway у розробці системи обліку витрат дозволяє зручно керувати версіями бази даних, забезпечуючи легку міграцію та стабільну роботу системи. Це сприяє збереженню цілісності даних, а також полегшує управління змінами та розвитком системи в майбутньому.

3.6 Google Timezone API

Google Timezone API приймає HTTP-запити координат у вигляді широти та довготи, а повертає дані про часовий пояс для місцезнаходження, включаючи зміщення для UTC і літнього часу. Використання даного сервісу дозволяє перевикористовувати робочі рішення без розробки власних. Перевагою цього сервісу у порівнянні із бібліотеками є актуальність — інформація про часові пояси у Google Timezone API регулярно оновлюється, що дає можливість користувачам завжди отримувати останню інформацію. Ця API є корисною коли є потреба показувати користувачам дані враховуючи їх часові пояси.

3.7 Telegram

Telegram — багатоплатформовий месенджер, обраний як інтерфейс для доступу до системи обліку через чат-бот. Telegram є одним з найпопулярніших

месенджерів у світі з великою кількістю активних користувачів. Багато людей вже мають встановлений Telegram на своїх пристроях (рис. 3.5). Телеграм боти надають можливість створювати інтерактивні елементи, такі як кнопки, меню, списки і т.д. Це робить інтерфейс бота зручним та інтуїтивно зрозумілим для користувачів. Вони можуть вибирати опції та взаємодіяти з системою за допомогою простих команд. Використання Telegram для доступу до системи обліку фінансів дозволяє прибрати необхідність завантаження додаткових програм для користувачів, а також, перевикористати існуючий інтерфейс та користувацький досвід при розробці інтерфейсу взаємодії.

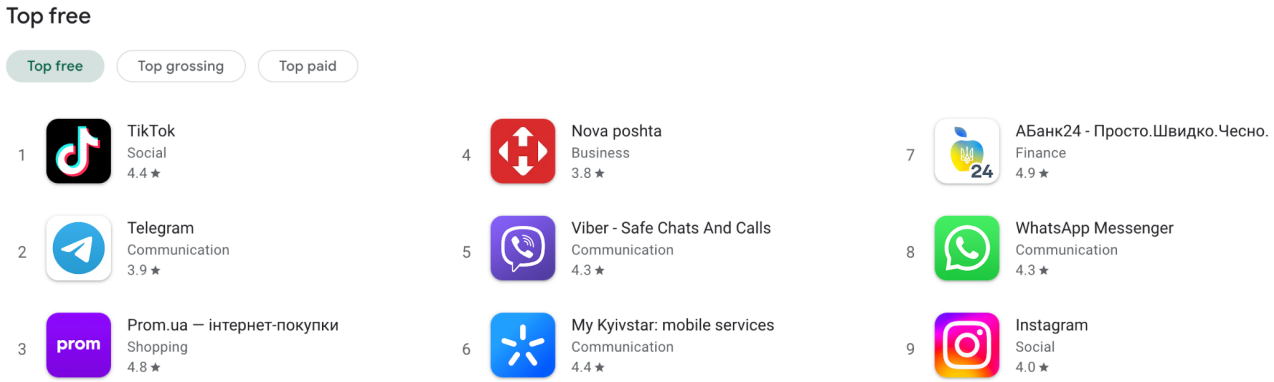


Рисунок 3.5 — Перелік популярних застосунків для Android

3.8 Бібліотека TelegramBots

TelegramBots — бібліотека з відкритим вихідним кодом, що надає зручний інтерфейс для створення та розробки Telegram-ботів з різноманітним функціоналом. Бібліотека забезпечує повну функціональність Telegram Bot API, що включає можливості взаємодії з користувачами, отримання та відправку повідомлень, роботу з клавіатурами, опрацювання команд, обробки подій та повідомлень від користувачів. Вона дозволяє створювати боти, які відповідають на повідомлення, команди та інші події у режимі реального часу, дозволяє створювати інтерактивні боти з кнопками, меню та іншими елементами управління, а також дозволяє використовувати додаткові функціональні

можливості Telegram Bot API, такі як робота з опитуваннями, опрацювання медіафайлів (фото, відео, аудіо), робота з груповими чатами, адміністрування та багато іншого. TelegramBots має добре документоване API та широку спільноту розробників, які активно діляться досвідом та надають підтримку. Це дозволяє швидко знайти відповіді на запитання та розв'язати проблеми, що виникають під час розробки бота, крім того, існує пакет для інтеграції зі Spring Boot Framework. Дана бібліотека є рекомендованою розробниками Telegram [9].

3.9 GitHub

GitHub — це веб-платформа для зберігання коду, управління версіями та спільної роботи над розробкою програмного забезпечення. Використання GitHub у розробці системи обліку витрат дозволяє ефективно керувати версіями коду та безпечно вносити в нього зміни впродовж усього циклу розробки.

3.10 diagrams.net

app.diagrams.net, раніше відомий як draw.io, є онлайн-інструментом для створення діаграм, схем, креслень та інших візуальних представлень. Він має зрозумілий інтерфейс, що дозволяє швидко створювати різноманітні діаграми та схеми без необхідності в глибоких знаннях графічного дизайну, надає велику кількість готових елементів та символів, які можна використовувати для створення різних видів діаграм, включаючи блок-схеми, UML-діаграми, потокові схеми та багато інших, дозволяє одночасно працювати над проектом кільком користувачам, обмінюватись діаграмами та спільно редагувати їх. Це спрощує командну роботу та спільне узгодження візуальних представлень, а також, app.diagrams.net підтримує імпорт та експорт діаграм у різних форматах, таких як PNG, PDF, SVG, XML, крім того, має зручну інтеграцію з GitHub, що дозволяє версіонувати діаграми та зручно зберігати їх поряд із проектом.

3.11 Digital Ocean

Digital Ocean — це хмарна платформа, яка надає інфраструктуру для розгортання та керування веб-додатками. Дана платформа дозволяє створювати віртуальні сервери (droplets), які можна швидко та легко створювати. DigitalOcean надає широкий вибір операційних систем (Linux дистрибутиви) та конфігураційних опцій для віртуальних серверів. Надається можливість обрати потрібну конфігурацію, оптимізовану під потреби користувача. DigitalOcean забезпечує високий рівень надійності та безпеки. Вони мають розташовані в різних регіонах центри обробки даних, що забезпечує резервне копіювання та відновлення даних. Крім того, DigitalOcean надає інструменти для мережевої безпеки, такі як мережеві правила та мережеві мережеві VPN. Також, DigitalOcean забезпечує можливість стежити за станом інфраструктури та додатків, розгорнутих на цій платформі. Є кілька інструментів моніторингу, зокрема:

– DigitalOcean Monitoring — вбудований сервіс, який надає основні метрики про ресурси, такі як використання CPU, пам'яті, мережі та дискового простору. Дані метрики можна відстежувати в реальному часі та налаштовувати сповіщення про події;

– DigitalOcean агент, який можна встановити на ваші віртуальні сервери для детальнішого моніторингу. Він збирає розширені метрики, такі як навантаження процесора, використання пам'яті, дискової активності та інші. Ці метрики дозволяють отримати більш повну картину про стан системи;

– DigitalOcean має інтеграції з різними зовнішніми сервісами моніторингу, такими як Prometheus, Grafana, Datadog та інші. Це дає вам можливість використовувати популярні інструменти для моніторингу та аналізу даних;

– можливість налаштування сповіщень про важливі події, такі як перевищення ліміту CPU або відсутність з'єднання з сервером.

Моніторинг на DigitalOcean надає можливість збирати та аналізувати дані протягом тривалого періоду часу. Це дозволяє відстежувати тенденції та планувати масштабування інфраструктури на основі зрозумілого контексту, а також спрощує виявлення причин непередбачуваних збоїв у роботі.

РОЗДІЛ 4. РЕАЛІЗАЦІЯ СИСТЕМИ

4.1 Проектування діаграми варіантів використання

Першим кроком у проектуванні системи для обліку фінансів є визначення варіантів використання. Взявши до уваги основні цілі створення системи, було визначено наступні варіанти використання:

– реєстрація користувача;

– створення транзакції із вказанням суми та категорії;

– імпорт та транзакцій;

– отримання списку категорій та сум витрачених у кожній з них за вказаний день або місяць;

– отримання списку транзакцій у категорії за вказаний день або місяць;

Для створення діаграми варіантів використання (Use-case) та всіх інших використовався сервіс app.diagrams.net через його простоту та зручну інтеграцію з GitHub, її наведено у додатку А.

4.2 Структура бази даних

Згідно з визначеними варіантами використання, було спроектовано структуру бази даних застосунку, що містить п’ять таблиць, діаграму зв’язків яких зображено на рис. 4.1. Перелік таблиць з їх описами наведено у таблиці 4.1. Детальний опис колонок кожної з таблиць наведено у таблицях 4.2-4.6

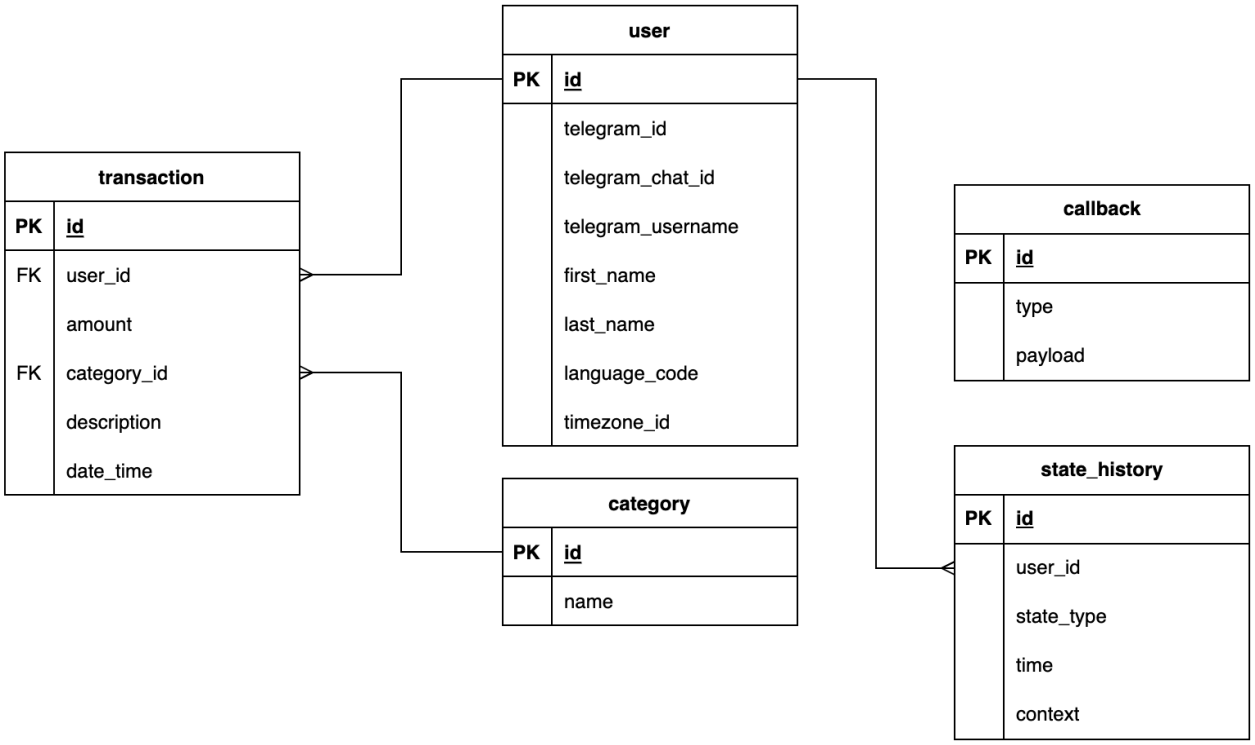


Рисунок 4.1 — Схема бази даних

Таблиця 4.1 — Перелік таблиць

Номер	Таблиця	Опис
1	user	Таблиця, що зберігає дані про користувачів
2	transaction	Таблиця для збереження інформації про транзакції
3	category	Таблиця для збереження категорій транзакцій
4	callback	Таблиця для збереження запитів зворотного виклику (callback query)
5	state_history	Таблиця для збереження історії стану користувача в системі впродовж часу

Таблиця 4.2 — Опис таблиці user

Назва колонки	Тип даних	Опис колонки
id	bigint	Унікальний ідентифікатор користувача
telegram_id	bigint	Унікальний ідентифікатор користувача у системі Telegram
telegram_chat_id	bigint	Унікальний ідентифікатор чату з користувачем у системі Telegram
telegram_username	varchar	Унікальне ім’я користувача у системі Telegram
first_name	varchar	Ім’я користувача
last_name	varchar	Прізвище користувача
language_code	varchar	Мова інтерфейсу користувача
timezone_id	varchar	Часовий пояс користувача

Таблиця 4.3 — Опис таблиці transaction

Назва колонки	Тип даних	Опис колонки
id	bigint	Унікальний ідентифікатор транзакції
user_id	bigint	Ідентифікатор користувача, що додав транзакцію, посилання на таблицю user
amount	double	Розмір транзакції
category_id	bigint	Ідентифікатор категорії транзакції, посилання на таблицю category
date_time	timestamp	Дата та час додавання транзакції в систему

Таблиця 4.4 — Опис таблиці category

Назва колонки	Тип даних	Опис колонки
id	bigint	Унікальний ідентифікатор категорії
name	varchar (64)	Назва категорії

Таблиця 4.5 — Опис таблиці callback

Назва колонки	Тип даних	Опис колонки
id	bigint	Унікальний ідентифікатор запиту зворотного виклику
type	varchar	Тип запиту
payload	json	Тіло запиту зворотного виклику

Таблиця 4.6 — Опис таблиці state_history

Назва колонки	Тип даних	Опис колонки
id	bigint	Унікальний ідентифікатор запису
user_id	bigint	Ідентифікатор користувача, до якого відноситься запис, посилання на таблицю user
state_type	varchar	Тип стану
time	timestamp	Дата та час переходу в даний стан
context	json	Корисна інформація про стан користувача в системі

4.3 Діаграма класів

Для реалізації системи було спроектовано діаграму класів [10], де наведено основні класи, що відповідають за обробку повідомлень, збереження та отримання транзакцій та інше (див. додаток Б).

4.4 Реалізація системи

Для створення додатку було використано Spring initializr — утиліту для швидкого старту проектів, що використовують Spring Framework, де було вказано мову програмування — Java, обрано останню актуальну версію Spring Boot, останню LTS версію мови Java та обрано базовий набір необхідних модулів: PostgreSQL Driver, Spring Data JPA, Spring Data JDBC, Flyway Migration. Процес ініціалізації зображено на рис. 4.1. Після того, як проект був успішно створений, можна приступати до розробки.

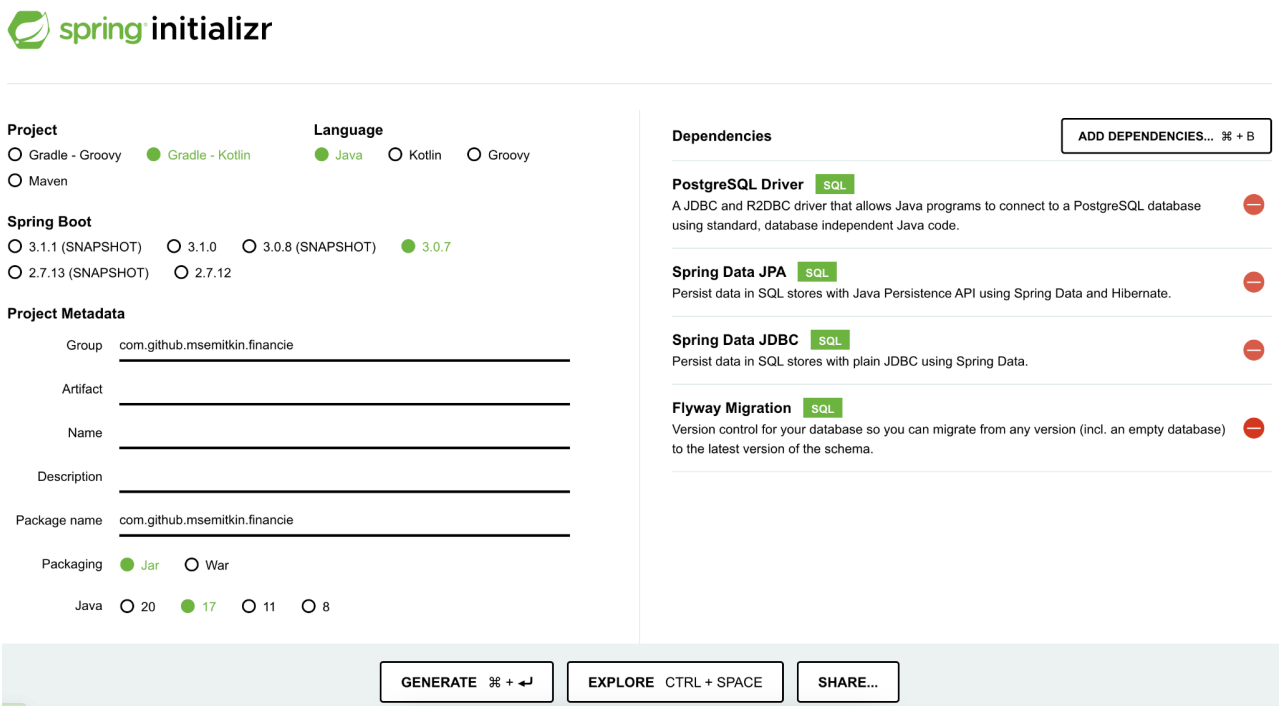


Рисунок 4.1 — Вікно ініціалізації проекту Spring initializr

Наступним кроком є створення реєстрація чат боту у BotFather — спеціальному боті в Telegram, який використовується для створення та налаштування інших ботів. Він є офіційним інструментом від Telegram і надає зручний і простий спосіб створення свого власного Telegram-бота (рис. 4.2). Для реєстрації необхідно вказати назву бот, а також його унікальне ім’я. У разі коректного введених усіх необхідних даних, BotFather надішле у відповідь

посилання, за яким буде доступний бот, а також ключ доступу до API, через яке можна буде отримувати інформацію повідомлення від користувачів та інші події, його необхідно зберігати у захищеному сховищі та уникати потрапляння його до посторонніх рук. У налаштуваннях боту також є можливість опціонально додати опис та аватар для для кращої ідентифікації боту користувачами.

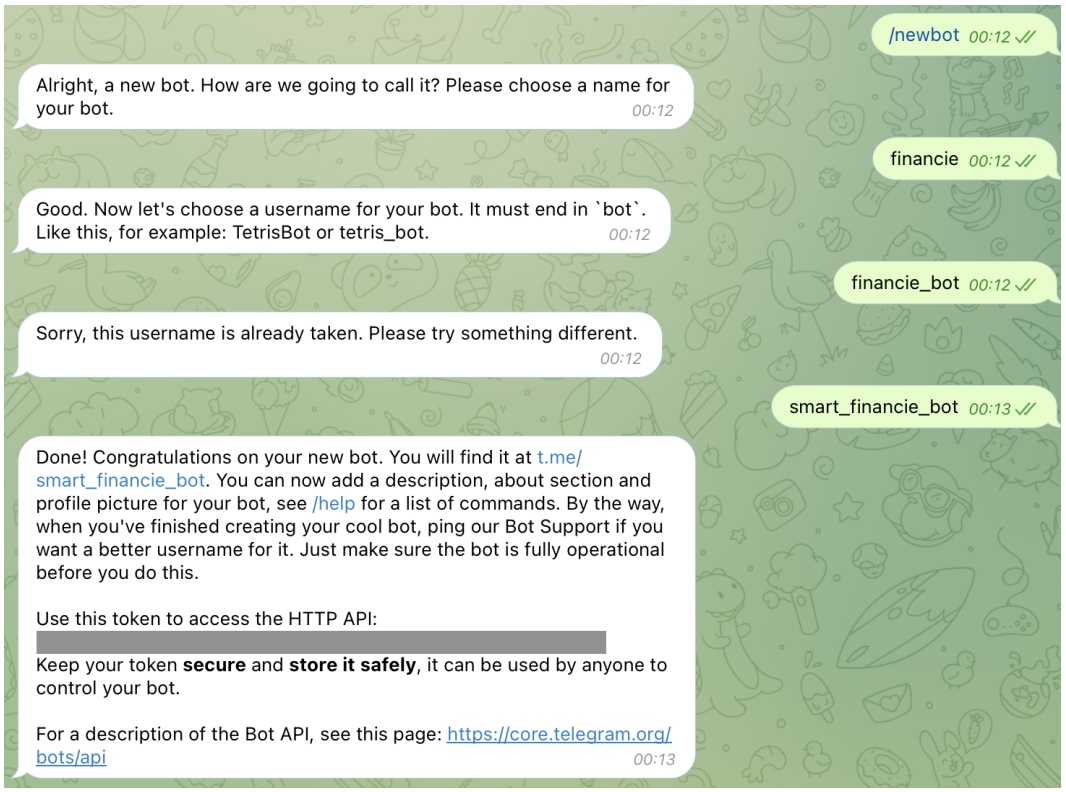


Рисунок 4.2 — Реєстрація боту у BotFather

Дякуючи інтеграції TelegramBots бібліотеки зі Spring Boot, для підключення боту, треба додати в контекст об’єкт класу, що реалізовує інтерфейс TelegramBot, для цього було описано клас FinancieTelegramBot (див. рис. 4.3). Даний клас використовуючи ім’я боту та ключ буде повідомлення від користувачів та інші події з серверу Telegram використовуючи тривале опитування (long polling) — клієнт (чат-бот) підключається до сервера який не закриває з'єднання доки не з'являться дані або мине час очікування. Після чого

клієнт підключається повторно. Даний підхід дозволяє отримувати інформацію про події та реагувати на них миттєво.

```
@Component
public class FinancieTelegramBot extends TelegramLongPollingBot {
    private static final Logger logger = LoggerFactory.getLogger(FinancieTelegramBot.class);

    private final String username;
    private final ApplicationEventPublisher applicationEventPublisher;

    public FinancieTelegramBot(
        @Value("${bot.telegram.username}") String username,
        @Value("${bot.telegram.token}") String botToken,
        ApplicationEventPublisher applicationEventPublisher
    ) {
        super(botToken);
        this.username = username;
        this.applicationEventPublisher = applicationEventPublisher;
    }

    @Override
    public String getBotUsername() {
        return username;
    }

    @Override
    public void onRegister() {
        logger.info("Bot successfully registered");
        super.onRegister();
    }

    @Override
    public void onUpdateReceived(Update update) {
        applicationEventPublisher.publishEvent(new UpdateReceivedEvent(update));
    }
}
```

Рисунок 4.3 — Реалізація інтерфейсу TelegramBot

Для того, щоб схема бази даних була завжди в актуальному стані та готова до роботи, було використано бібліотеку Flyway, що дозволяє описувати міграції бази даних мовою SQL та перевіряти актуальніть схеми перед кожним запуском застосунку. Існує три типи Flyway міграцій: версіоновані міграції — виконуються лише один раз та зазвичай використовуються для створення або модифікації схеми бази даних, а також модифікації даних, повторювані міграції — виконуються кожного разу, коли текст міграції змінився, міграції скасування — потрібні для скасування змін, зроблених у версіонованій міграції. Щоб Flyway мав змогу зрозуміти яка з міграцій належить до якого типу, потрібно

використовувати спеціальний формат для назв файлів, у яких описано міграції (див. рис. 4.3) [11].

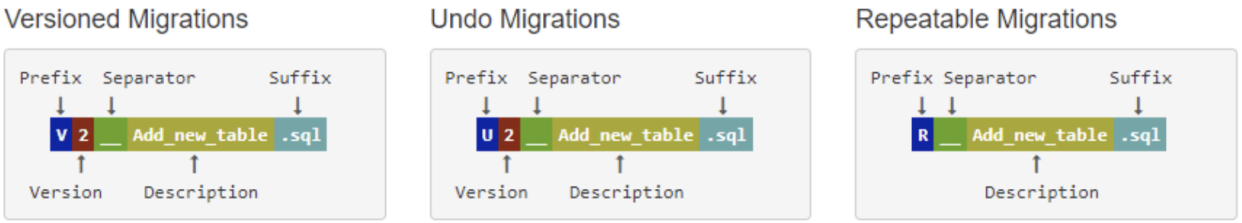


Рисунок 4.3 — Шаблони назв Flyway міграцій

Для створення схеми бази даних у системі для обліку фінансів було створенно окремі міграції для кожної таблиці. Всього було створено 11 міграцій, код яких наведено у додатку В.

Для отримання доступу до даних у проекті було використано репозиторії Spring Data JPA. Запити JPA автоматично генеруються на основі назви методів. Для складання запитів зі специфічною логікою використовується JPQL, що описується в анотацій @Query безпосередньо біля самого методу. JPQL — це мова запитів, яка дозволяє формувати запити до бази даних, використовуючи моделі сутностей. Її синтаксис і структура дуже схожі на SQL. Для створення репозиторію потрібно описати інтерфейс, що розширює інтерфейс JpaRepository. Реалізовувати власноруч його не потрібно, Spring Boot бере це на себе (див. рис. 4.4).

```
public interface TransactionRepository
    extends JpaRepository<TransactionEntity, Long> {
    @Query("""
        SELECT t FROM TransactionEntity t
        WHERE t.userId = :userId
        AND t.categoryId = :categoryId
        AND t.dateTime >= :start
        AND t.dateTime < :end""")
    List<TransactionEntity> findAllByUserIdAndCategoryIdAndDateTimeBetween(
        @Param("userId") Long userId,
        @Param("categoryId") Long categoryId,
        @Param("start") LocalDateTime start,
        @Param("end") LocalDateTime end
    );

    @Query("""
        SELECT t FROM TransactionEntity t
        WHERE t.userId = :userId
        AND t.dateTime >= :start
        AND t.dateTime < :end""")
    List<TransactionEntity> findAllByUserIdAndDateTimeBetween(
        @Param("userId") Long userId,
        @Param("start") LocalDateTime start,
        @Param("end") LocalDateTime end
    );

    List<TransactionEntity> findAllByUserIdOrderByDateTimeDesc(long userId);
}
```

Рисунок 4.4 — Опис інтерфейсу репозиторію для роботи з даними транзакцій

4.5 Обробка подій

4.5.1 Ланцюжок обов’язків

При розробці обробників подій, було використано ланцюжок обов’язків (chain of responsibility) [13] — поведінковий шаблон проектування, що дає змогу передавати запити послідовно ланцюжком обробників. Кожен наступний обробник вирішує, чи може він обробити запит сам і чи варто передавати запит далі ланцюжком. Використовуючи цей підхід, кожен з обробників має свою зону відповідальності та додавання нових обробників стає легкою операцією. Для цього було описано інтерфейс обробника UpdateHandler (рис. 4.5). Він має один публічний метод — processUpdate, який викликається у ланцюжку, всередині нього перевіряється чи може обробник обробити цей запит, якщо так — повідомленн обробляється та ланцюжок переривається, інакше — запит

передається наступному обробнику, якщо такого немає — створюється виняток, що повідомляє про неможливість обробити повідомлення. Приклади реалізації класу UpdateHandler наведено у додатку Г. Реалізація ланцюжка інкапсульована у класі UpdateHandlerChain, який включає в себе допоміжний клас для конфігурації ланцюжка (рис. 4.6)

```
public abstract class UpdateHandler {
    private UpdateHandler next;

    public void setNext(UpdateHandler next) { this.next = next; }

    public void processUpdate(Update update) {
        if (canHandle(update)) {
            handleUpdate(update);
        } else if (next != null) {
            next.processUpdate(update);
        } else {
            throw new UnhandledUpdateException();
        }
    }

    3 implementations
    protected abstract boolean canHandle(Update update);

    19 implementations
    protected abstract void handleUpdate(Update update);
}
```

Рисунок 4.5 — Інтерфейс обробника

```
public class UpdateHandlerChain {
    private final UpdateHandler rootHandler;

    public UpdateHandlerChain(UpdateHandler rootHandler) { this.rootHandler = rootHandler; }

    public void handleUpdate(Update update) { rootHandler.processUpdate(update); }

    public static UpdateHandlerChain.Builder builder() { return new Builder(); }

    public static class Builder {
        private UpdateHandler root;
        private UpdateHandler tail;

        public Builder addHandler(UpdateHandler updateHandler) {
            if (root == null) {
                root = updateHandler;
            }
            if (tail != null) {
                tail.setNext(updateHandler);
            }
            tail = updateHandler;
            return this;
        }

        public UpdateHandlerChain build() { return new UpdateHandlerChain(root); }
    }
}
```

Рисунок 4.6 — Реалізація класу UpdateHandlerChain

4.5.2 Обробка текстових повідомлень

Для створення нових витрат використовуються звичайні текстові повідомлення у форматі <сума> <категорія>, тому важливо, щоб система могла коректно обробляти їх. Для цього була створена окрема реалізація інтерфейсу UpdateHandler — SaveTransactionHandler. Реалізація canHandle методу наведена на рис 4.7. Натискання на кнопки меню надсилає повідомлення з відповідним текстом кнопки в чат, тому обробка натискання кнопок також зводиться до обробки повідомлень.

```
@Override
protected boolean canHandle(Update update) {
    return Optional.ofNullable(update.getMessage()) Optional<Message>
        .map(Message::getText) Optional<String>
        .map(transactionRecognizer::hasTransactionFormat) Optional<Boolean>
        .orElse( other: false);
}
```

Рисунок 4.7 — Реалізація методу canHandle для обробки витрат

Повідомлення, що мають необхідний формат витрати, валідуються та додаються в базу даних(див. рис. 4.8).

```
@Override
protected void handleUpdate(Update update) {
    UserContext userContext = UserContextHolder.getContext();
    Long chatId = getChatId(update);
    String text = update.getMessage().getText();
    Integer messageId = update.getMessage().getMessageId();
    try {
        transactionCommandValidator.validateTransaction(text);
        IncomingTransaction incomingTransaction = transactionParser.parseTransaction(text);
        long userId = userService.getUserByTelegramId(getSenderTelegramId(update)).id();
        var command = new SaveTransactionCommand(
            userId, incomingTransaction.amount(), incomingTransaction.category(), null, null);
        transactionService.saveTransaction(command);
        sendSuccessfullySavedTransaction(chatId, userId, messageId, incomingTransaction.category(),
            userContext.locale(), userContext.timeZone().toZoneId());
    } catch (MessageException e) {
        sendErrorMessage(chatId, messageId, e);
    }
}
```

Рисунок 4.8 — Процес обробки транзакції

4.5.3 Обробка inline кнопок

Обробка inline кнопок в Telegram боті дозволяє створити інтерактивні елементи у вигляді кнопок, які можуть бути відображені в повідомленнях. Коли користувач натискає на таку кнопку, бот отримує спеціальне повідомлення з даними про натиснуту кнопку і може відреагувати на нього відповідним чином. Для обробки inline кнопок у Telegram боті використовується InlineKeyboardMarkup. Це об'єкт, який містить одну або кілька рядків кнопок. Кожна кнопка має поле text та callback_data, яке представляє ідентифікатор

кнопки. При натисканні на кнопку, Telegram бот отримує повідомлення з інформацією про натиснуту кнопку. Ця інформація міститься у полі callback_query об'єкта Update. За допомогою цієї інформації бот може визначити, яка кнопка була натиснута та відповідно реагувати на цю дію. Для обробки таких кнопок, було реалізовано допоміжний клас, якому делегується робота по визначенні типу кнопки (див. рис. 4.9). Для прив’язки корисної інформації до кнопки, такої як: номер сторінки, ідентифікатор категорії та інший контекст, який може бути використаним при обробці, використовується клас CallbackService, який зберігає дану інформацію в базу даних з прив’язкою до ідентифікатора, а сам ідентифікатор додається у поле callback_data кнопки. Таким чином, при натисканні кнопки, дана інформація отримується з бази даних та дозволяє кастомізувати обробку події.

```
class CallbackQueryUpdateMatcher implements UpdateMatcher {
    private final CallbackService callbackService;
    private final Set<CallbackType> queryTypes;

    CallbackQueryUpdateMatcher(CallbackService callbackService, Set<CallbackType> queryTypes) {
        this.callbackService = callbackService;
        this.queryTypes = queryTypes;
    }

    @Override
    public boolean match(Update update) {
        return Optional.ofNullable(getCallbackId(update)).map(callbackService::getCallbackType)
            .map(queryTypes::contains).orElse( other: false);
    }

    @Nullable
    private UUID getCallbackId(Update update) {
        return Optional.ofNullable(update.getCallbackQuery()).map(CallbackQuery::getData)
            .map(UUID::fromString).orElse( other: null);
    }
}
```

Рисунок 4.9 — Реалізація допоміжного класу

4.5.4 Реалізація функції імпорту витрат

Для реалізації функції імпорту витрат у вигляді CSV файлу було реалізовано окремий обробник, який перевіряє чи містить вхідне повідомлення файл. Він перевіряє, що файл має текстовий формат та розширення .csv, також перевіряється правильність дат (рис. 4.10), числових значень та обов'язкових полів. У випадку успішного сценарію, усі витрати з файлу будуть збережені в базу даних та доступні для перегляду в боті

```
private boolean hasCsvFile(Update update) {
    return Optional
        .ofNullable(update.getMessage())
        .map(Message::getDocument)
        .map(Document::getMimeType)
        .map("text/csv"::equals)
        .orElse( other: false);
}
```

Рисунок 4.10 — Перевірка наявності CSV файлу в повідомленні

4.6 Надсилання повідомлень користувачу

Надсилання повідомлень користувачу реалізоване за допомогою бібліотеки TelegramBots. Для надсилання повідомлення необхідно викликати метод execute класу TelegramApi, передавши клас SendMessage із необхідниими полями: ідентифікатор чату користувача, текст повідомлення та, опціонально, додавши клавіатуру для відповіді, а також параметр форматування тексту. Відповідь користувачу на запит інформації про витрачені суми на категорії за вказаний день показано на рис. 4.11

```
public void handleUpdate(Update update) {
    var response = dailyCategoriesResponseService.prepareResponse(update, new
     GetDailyCategoriesCommand( offset: 0));

    telegramApi.execute(SendMessage.builder()
        .chatId(getChatId(update))
        .text(response.text())
        .parseMode(ParseMode.MARKDOWNV2)
        .replyMarkup(response.keyboardMarkup())
        .build());
}
```

Рисунок 4.11 — Обробка запиту на вичитку інформації по категоріях

4.7 Визначення часового поясу користувача

Для визначення часового поясу користувача було використано функцію Telegram API запиту поточного місця розташування. Ця робиться за допомогою кнопки, вказавши значення атрибуту requestLocation як true. Для зручності створення такої кнопки було винесене в окремий метод (рис. 4.12)

```
public static KeyboardButton requestLocationButton(String text) {
    return KeyboardButton.builder()
        .text(text)
        .requestLocation(true)
        .build();
}
```

Рисунок 4.12 — Створення кнопки для запиту локації користувача

При натисканні кнопки, користувача буде запитано чи погоджується він передати інформацію про його місце розташування, якщо він погоджується, бот отримає повідомлення з координатами користувача, після чого за допомогою Google TimeZone API визначається його часовий пояс та зберігається в базу даних. Запити до Google TimeZone API інкапсульовані у класі GetTimezoneByLocationSource (рис. 4.13). Для запиту до API потрібно вказати

довготу, широту та ключ доступу, який потрібно отримати попередньо створивши Google Cloud проект [12].

```
@Service
class GetTimezoneByLocationSource implements GetTimezoneByLocationPort {
    private final GeoApiContext context;

    GetTimezoneByLocationSource(
        @Value("${com.github.msemitkin.financie.googlemapsplatform.api-key}") String apiKey) {
        this.context = new GeoApiContext.Builder()
            .apiKey(apiKey)
            .build();
    }

    @Override
    public TimeZone getTimezoneByLocation(Location location) {
        return TimeZoneApi
            .getTimeZone(context, new LatLng(location.latitude(), location.longitude()))
            .awaitIgnoreError();
    }
}
```

Рисунок 4.13 — Інтеграція з Google TimeZone API

4.8 Реалізація підтримки кількох мов інтерфейсу

Для розширення кола користувачів було вирішено підтримувати дві мови інтерфейсу: англійську та українську. Якщо мова інтерфейсу Telegram не є однією з цих двох, буде використовуватися мова за замовчуванням — англійська. Реалізація включає декілька кроків:

– реалізовано автоматичне визначення мови користувача: бібліотека TelegramBots дозволяє отримати налаштування мови користувачаа у системі Telegram через клас Update, таким чином налаштування мови актуалізуються з кожним повідомленням, що надсилає користувач боту.

– збереження локалізованих ресурсів: усі ресурси, що потребують локалізації були винесені у пакети ресурсів (Resource bundle) [14] та для кожної мови створено окремий файл, де визначені відповідні ресурси для цієї мови (рис. 4.14).

Рисунок 4.14 — Пакет ресурсів із файлами для кожної мови

– використано відповідні локалізовані ресурси для відображення тексту повідомлень, кнопок, повідомлень про помилки та іншого інтерфейсу бота. Замість фіксованих текстів використовуються ключі, які відповідають відповідним локалізованим ресурсам. Для цього було реалізовано клас ResourceService, який дозволяє отримати ресурс за його ключем та мовою користувача (рис. 4.15).

```
@NonNull
public static String getValue(String key, Locale locale) {
    return ResourceBundle.getBundle( baseName: "texts", locale).getString(key);
}
```

Рисунок 4.15 — Реалізація методу отримання ресурсу за ключем та мовою

4.9 Збірка проекту

Для збірки проекту використано Gradle — потужний інструмент для збирання проектів, включаючи керування залежностями, компіляцію та ін. Проект упаковується у JAR архів — стандартний спосіб пакування Java додатків. Для збірки використовується команда “./gradlew build”. Gradle виконає збирання, включаючи компіляцію, тестування, пакування та ін. Результати

будуть збережені в директорії build проекту. Для запуску архіву використовується команда “java -jar <шлях до архіву>”, для запуску на сервері потрібно додатково вказати такі параметри як шлях та дані для доступу до бази даних, ключ Telegram бот, а також ключ для доступу до Google Timezone API. Додаткові аргументи вказуються в кінці команди у форматі
--<ім’я аргументу>=<значення аргументу> та розділяються пробілом.

4.10 Розгортання на сервері

Розгортання на платформі Digital Ocean включає в себе наступні кроки:

– реєстрація на платформі DigitalOcean;

– створення серверу (Droplet);

– підключення до сервера. Підключення до серверу відбувається використовуючи SSH тунель. Для цього на локальному комп’ютері необхідно ввести наступну команду : “ssh <ім’я користувача на сервері>@<IP адреса сервера>”. Попередньо необхідно в налаштування сервера вказати публічний ключ, що відповідає приватному ключу, збереженому на локальному комп’ютері, який використовується для доступу до серверу

– завантаження застосунку на сервер. Для завантаження JAR архіву використовується команда “scp <шлях до архіву на локальному комп’ютері> <ім’я користувача на сервері>@<IP адреса сервера>:<цільовий шлях до архіву на сервері>”

– запуск застосунку. Процес запуску описаний у пункті 4.9.

РОЗДІЛ 5. РОБОТА З СИСТЕМОЮ

5.1 Старт

Для запуску боту необхідно перейти за посиланням https://t.me/smart_financie_bot та натиснути кнопку “Розпочати”. Після цього бот надішле вітальне повідомлення та буде відображено головні кнопки управління:

– Сьогодні: кнопка для отримання інформації про витрачені кошти в день запиту по категоріях;

– Цього місяця: кнопка для отримання інформації про витрачені кошти по категоріях цього місяця;

– Меню: Кнопка для переходу на головне меню.

Також доступні команди після натискання кнопки Меню:

– /start: Комадна для перезавантаження боту. Дана команада бути корисною при збою в роботі боту;

– /help: Допомога з використанням бота, короткий опис функцій;

– /author: Інформація про автора боту.

5.2 Вибір часового поясу

Після старту боту користувачу рекомендується вказати його часовий пояс для більш коректного збереження витрат під час ручного введення та імпорту витрат, а також під час відображення та експорту витрат. Для цього користувачу необхідно перейти Налаштування (> Меню > Налаштування) та натиснути кнопку “Визначити часовий пояс” після цього погодитись на передачу поточного місця розташування. У відповідь бот надішле повідомлення про успішне встановлення часового поясу.

5.3 Імпорт витрат

Деякі користувачі перед початком роботи захочуть імпортувати CSV файл з даними про витрати у минулому, для цього їм необхідно натиснути кнопку “Імпортувати транзакції” (> Меню > Імпортувати транзакції), після чого користувач отримує відповідь про необхідні колонки, які повинен мати файл, також користувач отримує файл-шаблон, який уже має всі необхідні колонки та може бути використаний для наповнення. Після цього користувачу потрібно натиснути на кнопку з іконкою скріпки і обрати файл зі списку та натиснути “Надіслати”, після чого бот повідомляє про те, що файл було завантажено та почато обробку. Користувача буде додатково повідомлено про завершення обробки. Також користувач може відмінити операцію, натиснувши кнопку “Відміна”.

5.4 Додавання витрат

Щоб додати нову витрату, користувачу потрібно ввести повідомлення у форматі “<сума> <категорія витрати>”, після чого натиснути кнопку “Надіслати”. Користувачу буде надіслано у відповідь про успішне збереження та коротку інформацію про витрати: загальну суму, витрачену сьогодні, загальну суму, витрачену цього місяця та загальну суму, витрачену цього місяця у даній категорії. Мова відповіді буде залежати від мови інтерфейсу користувача у застосунку Telegram. Також є можливіть додати витрату за попередній день, у випадку, якщо користувач не зробив це відразу. Для цього потрібно натиснути кнопку ”Сьогодні” та за допомогою стрілок вліво та вправо вибрати необхідний день, після цього натиснувши кнопку “Додати транзакцію”, ввести дані про транзакцію.

5.5 Перегляд витрат за день

Щоб переглянути витрати за день, користувачу потрібно натиснути кнопку “Сьогодні”, після чого у відповідь буде надіслано список витрат за цей день. Натиснувши на транзакцію, користувачу буде покказане вікно з кнопкою “Видалити” — для видалення витрати. Вліво та вправо користувач може подивитися інформацію за попередні дні.

5.6 Перегляд витрат за місяць

Щоб переглянути витрати за місяць, користувачу потрібно натиснути кнопку “Цього місяця”, після чого у відповідь буде надіслано список категорій та сум витрат за цей місяць у кожній з них. Натиснувши на категорію, користувачу буде показано список витрат у даній категорії за обраний місяць та, натиснувши на транзакцію, користувач зможе її видалити, аналогічно з пунктом 5.5. Також доступна навігація по місяцях за допомогою кнопок вліво та вправо.

5.7 Експорт витрат

Для експорту витрат користувачу потрібно натиснути кнопку ”Експортувати транзакції у CSV” (> Меню > Експортувати транзакції у CSV), після чого користувачу у відповідь буде надіслано CSV файл з усіма транзакціями за період використання системи.

ВИСНОВКИ

Розроблено систему для обліку витрат з метою надання зручного і ефективного інструменту для контролю, аналізу витрат, оптимізації часу, що витрачається на облік витрат, збереження історії витрат, а також впровадження процесу обліку витрат в повсякденне життя людей з метою контролю витрат, їх оптимізації, а також зваженого планування бюджету в майбутньому. Для взаємодії з системою розроблено Telegram бот.

Досліджені та описані існуючі на ринку системи, визначені їх переваги та недоліки, здійснено моделювання системи, розроблено структуру бази даних, діаграму прецедентів, розроблено інтерфейс боту, реалізовано обробку повідомлень.

Перспективами подальшої розробки є можливість додавати витрати у різних валютах, удосконалення аналітики, інтеграції з платіжними шлюзами, банківськими API або іншими фінансовими сервісами, щоб автоматизувати облік транзакцій та отримувати більш точну інформацію про фінансовий стан користувача, оптимізація та вдосконалення алгоритмів для можливості обробляти великі обсяги даних та інше.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

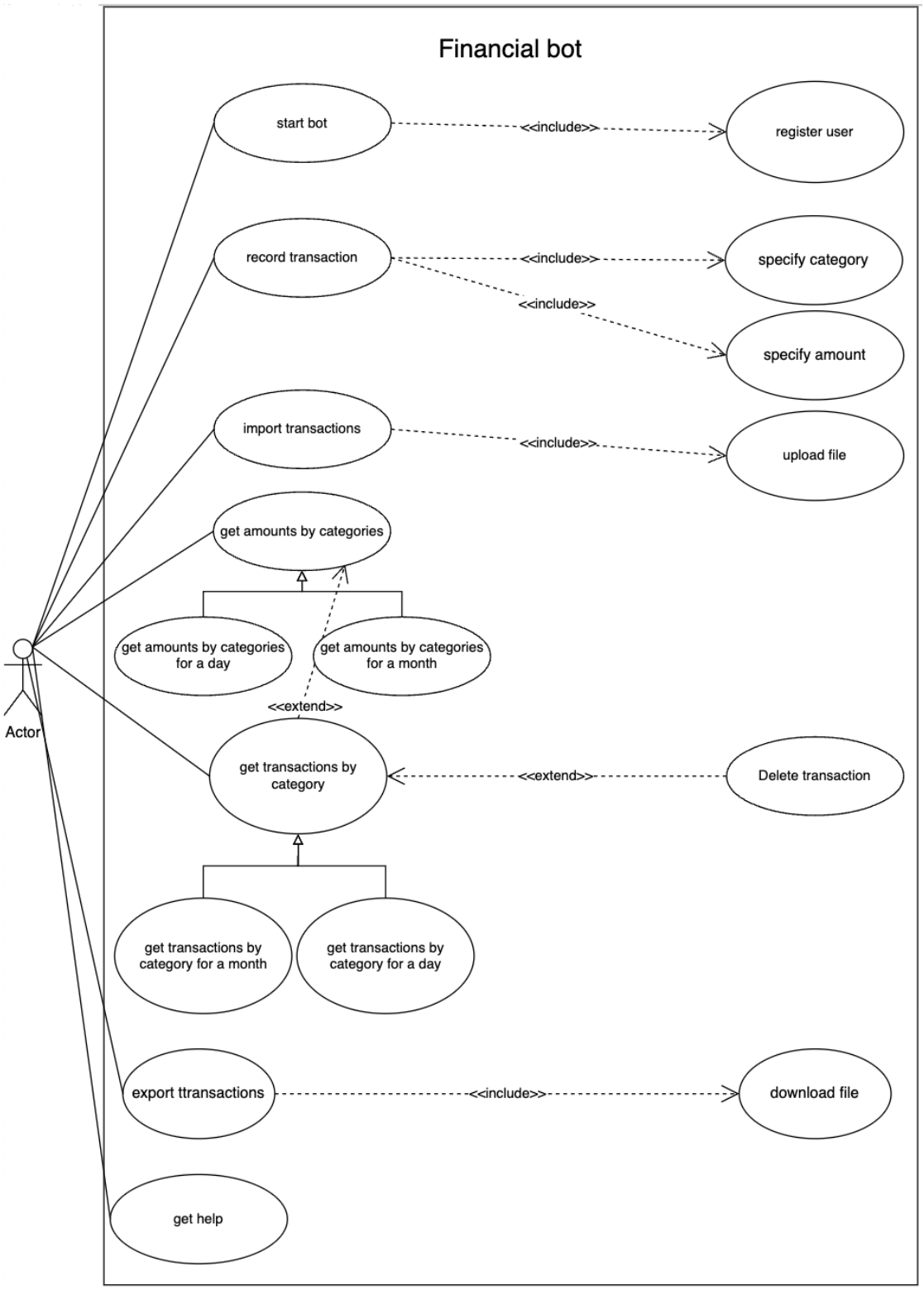
1. Sidharta Y. [Product Review] Money Lover: Great Expense Tracker [Електронний ресурс] / Yosef Sidharta // Medium. – Режим доступу: https://medium.com/@yosefsid/product-review-money-lover-great-expense-tracker-1dd00da18a41.
2. GitHub - muety/telegram-expense-bot: A bot that helps you manage and track your daily expenses. [Електронний ресурс] // GitHub. – Режим доступу: https://github.com/muety/telegram-expense-bot. – Назва з екрана.
3. JetBrains Marketplace [Електронний ресурс] // JetBrains Marketplace. – Режим доступу: https://plugins.jetbrains.com/idea. – Назва з екрана.
4. TOP IDE Top Integrated Development Environment index [Електронний ресурс] // Page Redirection. – Режим доступу: https://pypl.github.io/IDE.html. – Назва з екрана.
5. 10 Most Popular Java IDEs for Coding in 202 [Електронний ресурс] // Turing Blog. – Режим доступу: https://www.turing.com/blog/best-java-ides-and-editors. – Назва з екрана.
6. TIOBE Index - TIOBE [Електронний ресурс] // TIOBE. – Режим доступу: https://www.tiobe.com/tiobe-index/. – Назва з екрана.
7. Spring Framework [Електронний ресурс] // Spring Framework. – Режим доступу: https://spring.io/projects/spring-framework. – Назва з екрана.
8. Stack Overflow Developer Survey 2022 [Електронний ресурс] // Stack Overflow. – Режим доступу: https://survey.stackoverflow.co/2022. – Назва з екрана.
9. Bot API Library Examples [Електронний ресурс] // Telegram APIs. – Режим доступу: https://core.telegram.org/bots/samples. – Назва з екрана.
10. financie/class-diagram.drawio.svg at master · msemitkin/financie [Електронний ресурс] // GitHub. – Режим доступу:

https://github.com/msemitkin/financie/blob/master/class-diagram.drawio.svg. – Назва з екрана.

11. Flyway: Naming Patterns Matter | Redgate [Електронний ресурс] // Redgate. – Режим доступу: https://www.red-gate.com/blog/database-devops/flyway-naming-patterns-matter. – Назва з екрана.
12. Get Started | Time Zone API | Google for Developers [Електронний ресурс] // Google for Developers. – Режим доступу: https://developers.google.com/maps/documentation/timezone/get-started. – Назва з екрана.
13. Gamma E. Design patterns: Elements of reusable object orientated software / Erich Gamma. – Reading, USA : Addison-Wesley, 1995. – 395 с.
14. ResourceBundle (Java Platform SE 8) [Електронний ресурс] // Oracle. – Режим доступу: https://docs.oracle.com/javase/8/docs/api/java/util/ResourceBundle.html. – Назва з екрана.

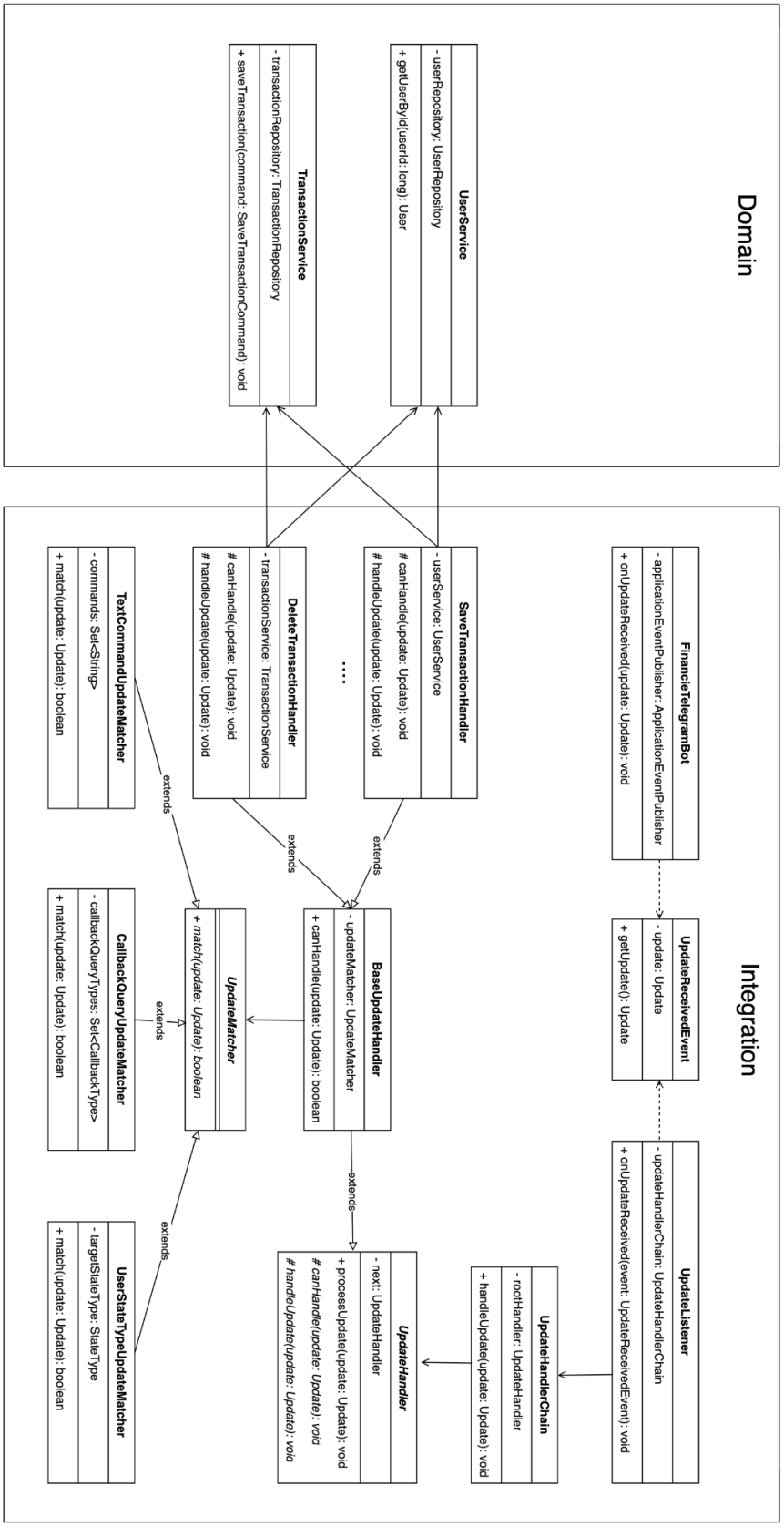
ДОДАТОК А

Діаграма варіантів використання (Use-case)



ДОДАТОК Б

Спрощена діаграма класів



ДОДАТОК В

Код SQL міграцій

Назва міграції	SQL код міграції

V20230216001610__create_category_table.sql	CREATE TABLE category
	(
	  id BIGSERIAL,
	  name VARCHAR(64)
	);

V20230216001811__create_user_table.sql	CREATE TABLE "user"
	(
	  id BIGSERIAL,
	  telegram_id BIGINT
	);

V20230216002121__create_transaction_table.sql	CREATE TABLE transaction
	(
	  id BIGSERIAL,
	  user_id BIGINT,
	  amount DOUBLE PRECISION,
	  category_id BIGINT,
	  description TEXT,
	  date_time TIMESTAMP
	);

V20230304144845__add_user_columns.sql	ALTER TABLE "user"
	  ADD COLUMN telegram_chat_id BIGINT,
	  ADD COLUMN telegram_username VARCHAR(64),
	  ADD COLUMN first_name VARCHAR(64),
	  ADD COLUMN last_name VARCHAR(64)

V20230304194857__create_callback_table.sql	CREATE TABLE callback
	(
	  id UUID NOT NULL PRIMARY KEY,
	  type VARCHAR NOT NULL,

	payload JSON NOT NULL);
V20230322223549__add_language_code_column.sql	ALTER TABLE "user" ADD COLUMN language_code VARCHAR NOT NULL DEFAULT 'en';
V20230516205422__add_unique_user_id_constraint.sql	ALTER TABLE "user" ADD PRIMARY KEY (id);
V20230516205610__create_state_history_table.sql	CREATE TABLE state_history (id BIGSERIAL NOT NULL PRIMARY KEY, user_id BIGINT NOT NULL REFERENCES "user" (id), state_type VARCHAR NOT NULL, time TIMESTAMP NOT NULL)
V20230516233104__add_timezone_id_column.sql	ALTER TABLE "user" ADD COLUMN timezone_id varchar NULL
V20230517000339__set_timezone_for_existing_users.sql	UPDATE "user" SET timezone_id = 'Europe/Kiev'
V20230518101259__add_context_column.sql	ALTER TABLE state_history ADD COLUMN context json NULL

ДОДАТОК Г

Приклади реалізації класу UpdateHandler

```
@Component
public class DefaultUpdateHandler extends UpdateHandler {
    private static final Logger logger = LoggerFactory
        .getLogger(DefaultUpdateHandler.class);
    private final TelegramApi telegramApi;

    public DefaultUpdateHandler(TelegramApi telegramApi) {
        this.telegramApi = telegramApi;
    }

    @Override
    protected boolean canHandle(Update update) {
        return true;
    }

    protected void handleUpdate(Update update) {
        logger.warn("Unrecognized update: {}", update);

        Integer messageId = Optional.ofNullable(update.getMessage()) Optional<Message>
            .map(Message::getMessageId) Optional<Integer>
            .orElse( other: null);

        String sorryMessage = ResourceService
            .getValue( key: "sorry-message", UserContextHolder.getContext().locale());
        long chatId = requireNonNull(getChatId(update));
        SendMessage sendMessage = SendMessage.builder()
            .chatId(chatId)
            .replyToMessageId(messageId)
            .text(sorryMessage)
            .build();
        telegramApi.execute(sendMessage);
    }
}
```