Міністерство освіти і науки України
Національний університет «Одеська політехніка»
Навчально-науковий інститут комп’ютерних систем
Кафедра інформаційних систем

Шидлюх Костянтин Олександрович,
студент групи AI-232

КУРСОВА РОБОТА

Дисципліна:
«ОБ’ЄКТНО-ОРІЄНТОВАНЕ ПРОГРАМУВАННЯ»

Укладачі:
М.А. Годовиченко

Одеса – 2025

ЗМІСТ

ВСТУП

У сучасному інформаційному суспільстві питання особистих фінансів посідає надзвичайно важливе місце. Постійне зростання кількості щоденних фінансових операцій, збільшення доступу до електронної комерції, поширення онлайн-банкінгу та зменшення використання готівкових коштів призводить до того, що люди все частіше стикаються з необхідністю планування та контролю власного бюджету. Раціональне управління доходами та витратами не лише сприяє покращенню фінансового стану домогосподарств, а й підвищує загальний рівень фінансової грамотності населення.

Одним із способів забезпечити ефективний контроль фінансів є створення автоматизованих систем обліку. Сучасні програмні засоби дозволяють користувачам вести облік витрат та доходів, аналізувати структуру витрат, формувати звіти за періодами, категоріями, перевіряти дотримання бюджету тощо. Зокрема, особливу популярність мають веб-застосунки, які надають доступ до фінансової інформації з будь-якого пристрою, що має доступ до мережі Інтернет.

Актуальність розробки такого веб-застосунку зумовлена кількома чинниками:

- зростанням кількості активних користувачів мобільних і веб-додатків для особистого обліку;
- потребою в централізованому, зручному та гнучкому інструменті ведення фінансів;
- браком локалізованих рішень, адаптованих до потреб українських користувачів;
- необхідністю вивчення сучасних технологій програмування, зокрема Java, Spring Boot, REST API, JPA/Hibernate, що є актуальними на ринку IT-праці.

Метою цієї курсової роботи є розробка програмного забезпечення у вигляді веб-системи, яка дозволяє користувачеві створювати, переглядати, оновлювати та видаляти записи про свої витрати, доходи, категорії та бюджети, а також отримувати розгорнуті аналітичні звіти. Для реалізації поставленої мети використовуються такі інструменти: мова програмування Java, фреймворк Spring Boot, модуль Spring Data JPA для взаємодії з базою даних, PostgreSQL як система керування базами даних, а також бібліотека Lombok для спрощення моделі даних.

Серед основних завдань, що були поставлені в межах роботи, можна виділити наступні:

- проєктування структури даних для зберігання інформації про фінансові операції;
- реалізація CRUD-функціоналу для основних сутностей;
- створення API для взаємодії з клієнтською частиною;
- реалізація логіки аналітичних запитів;
- перевірка коректності роботи застосунку за допомогою тестування.

Таким чином, курсова робота спрямована не лише на розв’язання практичної задачі − створення системи обліку витрат, а й на набуття глибоких знань і навичок у розробці серверних застосунків на Java з використанням сучасного технологічного стеку.

АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

Предметною областю дослідження в даній курсовій роботі є особисті фінанси користувачів − себто процеси обліку витрат, доходів, бюджетування та аналізу фінансових операцій в межах одного домогосподарства або окремої особи.

Фінансова поведінка користувача зазвичай включає ведення бюджету, регулярну фіксацію доходів та витрат, оцінку ефективності витрат за певними категоріями та контроль перевищення лімітів. Усі ці процеси вимагають систематизації та автоматизації.

Розробка системи обліку витрат передбачає створення структури, яка дозволяє виконувати такі дії:

- фіксувати витрати із зазначенням суми, дати, опису та категорії;
- фіксувати доходи, які мають джерело, суму та дату;
- групувати витрати за категоріями, щоб розуміти фінансову структуру;
- встановлювати місячний бюджет, який обмежує загальні витрати;
- формувати аналітичні звіти, які дозволяють користувачу робити висновки щодо ефективності управління фінансами.

2.1 Основні бізнес-об’єкти

1) Користувач (User)

Це основний суб’єкт системи. Кожен користувач має обліковий запис та взаємодіє із системою через створення витрат, доходів, встановлення бюджетів тощо. Користувачі мають унікальний ідентифікатор (id), ім’я (username) та електронну пошту (email).

Зв’язки:

- один користувач може мати багато витрат (OneToMany);
- один користувач може мати багато доходів;

- один користувач може мати індивідуальні бюджети по місяцях.

2) Витрата (Expense)

Це запис, що відображає факт витрати коштів користувачем. Кожна витрата має:

- суму (amount);
- дату (date);
- опис (description);
- категорію (прив’язку до неї) (category);
- прив’язку до користувача (user).

Витрати є ключовим аналітичним інструментом − на їх основі генерується звітність, визначається структура витрат та ефективність бюджету.

3) Категорія (Category)

Категорії дозволяють групувати витрати за тематичними напрямками (їжа, транспорт, освіта тощо). Кожна категорія має:

- назву (name);
- тип або підтип (type), який може використовуватись для додаткової класифікації.

Це дозволяє користувачеві бачити, на які цілі витрачається найбільше коштів, і, за потреби, оптимізувати свої витрати. Категорії є спільними для всіх користувачів (або можуть бути налаштовані як індивідуальні).

4) Дохід (Income)

Це запис про надходження коштів. Має такі атрибути:

- суму (amount);
- дату (date);
- джерело (source);
- користувача (user).

Аналіз доходів дає змогу не лише бачити баланс доходів/витрат, а й прогнозувати бюджет на майбутні місяці.

5) Бюджет (Budget)

Бюджет − це фінансовий ліміт, який користувач встановлює на конкретний місяць. Вміст інформації про бюджет:

- місяць у вигляді рядка (month, наприклад 2025-06);
- ліміт (totalAmount, у грошовому еквіваленті);
- користувача.

Система порівнює фактичні витрати за місяць із лімітом і може сигналізувати про перевищення або залишок бюджету.

2.2 Бізнес-сценарії використання

1) Додавання витрати.

Користувач відкриває мобільний або веб-застосунок, вводить дані про витрату (сума, дата, опис, категорія), система зберігає запис і автоматично враховує її в аналітиці.

2) Формування бюджету

На початку місяця користувач встановлює бюджет. Усі витрати цього місяця порівнюються із цим лімітом.

3) Отримання звіту

Користувач вибирає місяць, і система показує:

- суму витрат;
- витрати по категоріях;
- перевищення бюджету;
- топ найбільших витрат.

4) Аналіз поведінки

Користувач бачить, у яких категоріях найчастіше відбуваються витрати, у які дні відбувається пікове витрачання коштів, чи дотримується він фінансової дисципліни.

2.3 Обґрунтування структури моделі

Чітке розділення сутностей дозволяє:

- уникнути дублювання інформації;
- забезпечити правильну нормалізацію БД;
- легко додати нові поля (наприклад, валюта витрати, періодичність доходу тощо);
- масштабувати систему для багатьох користувачів.

Використання зв’язків типу OneToMany, ManyToOne між сутностями дозволяє забезпечити логічну прив’язаність кожного запису до його власника (користувача), а також до категорії, що дає агрегувати дані для аналітики.

2.4 Особливості предметної області

І все ж таки, треба перерахувати основні особливості предметної області:

- дата є критично важливим параметром майже для всіх об'єктів: вона визначає, до якого місяця відносити витрату, чи потрапляє вона в поточний бюджет тощо;
- категорії можуть бути системними (спільними) або призначатись окремо для користувача;
- дохід і витрата можуть бути джерелом аналітики у вигляді графіків, співвідношень, таблиць;

- бюджет є доволі гнучким параметром: користувач може змінювати його за потреби (редагування бюджету реалізовано окремо).

ПРОЄКТУВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

У цьому розділі описано, як було спроєктовано програмне забезпечення системи обліку витрат домогосподарства. Враховано вимоги предметної області, завдання користувача, принципи модульності та відповідність сучасним підходам до архітектури Java-застосунків. Система побудована за шаблоном трирівневої архітектури, що включає рівні доступу до даних, бізнес-логіки та контролю взаємодії з користувачем (REST API).

3.1 Структура даних (модель сутностей)

Основу проєкту становить модель даних, що відображає ключові об’єкти предметної області. Було виділено 5 основних сутностей:

1) User

Містить інформацію про користувача: ідентифікатор, ім’я (username) та електронну пошту. Кожен користувач має власні витрати, доходи та бюджети.

- id: Long − унікальний ідентифікатор;
- username: String − ім’я користувача;
- email: String − електронна пошта.

2) Expense

Описує витрату користувача. Має зв’язки з User та Category.

- id: Long − ідентифікатор витрати;
- amount: double/BigDecimal − сума витрати;
- date: LocalDate − дата витрати;
- description: String – опис;
- category: CategoryId − категорія витрати;
- user: UserId − користувач, до якого прив’язана витрата.

3) Category

Категорія витрати (наприклад, їжа, транспорт).

- id: Long − ідентифікатор категорії;
- name: String – назва/тип категорії;
- type: String (основна, допоміжна, інше).

4) Income

Запис про дохід користувача.

- id: Long − ідентифікатор запису про дохід;
- amount: double / BigDecimal – сума доходу;
- date: LocalDate – дата доходу;
- source: String – джерело надходження;
- user: UserId – користувач, який отримав дохід.

5) Budget

Встановлює ліміт витрат на конкретний місяць.

- id: Long – ідентифікатор бюджета/ліміта;
- month: String (формат yyyy-MM) – місяць, на який його встановлено;
- limit: BigDecimal – сума обмеження;
- user: UserId – користувач, до якого його прив’язано.

Усі зв’язки між сутностями реалізовані через JPA-анотації (ManyToOne, OneToMany), що дозволяє формувати правильну реляційну модель у базі даних (PostgreSQL).

3.2 Архітектура застосунку

Система реалізована відповідно до багаторівневої архітектури типу MVC (Model-View-Controller), у модифікованому вигляді, який більше підходить для RESTful веб-додатків. Вона включає такі логічні шари:

1) Model (Entity/Model + DTO)

- Entity (у моєму випадку Model): Java-класи, що представляють таблиці в базі даних.
- DTO: об’єкти для передачі даних між клієнтом і сервером, спрощують структуру JSON-відповідей та захищають внутрішню логіку.

2) Repository

Інтерфейси, що розширюють JpaRepository, забезпечують доступ до бази даних (CRUD-операції). За потреби – додаткові запити через @Query.

3) Service

Містить бізнес-логіку. Сервіси відповідають за:

- обробку запитів від контролера;
- валідацію вхідних даних;
- взаємодію з репозиторіями;
- агрегацію та обробку даних (наприклад, розрахунок бюджету, аналітика).

4) Controller

REST-контролери приймають HTTP-запити, викликають методи сервісів і повертають відповіді у форматі JSON. Для кожної сутності створено окремий контролер (UserController, ExpenseController тощо).

3.3 Організація пакетів

У проєкті дотримано чіткої структури пакетів:

ua.opnu.pract1shydl

- controller
- service
- repository
- model
- dto

Це дозволяє легко орієнтуватися в коді, розділяти відповідальності й дотримуватись принципу Single Responsibility.

3.4 REST API

Усі операції взаємодії із системою реалізовано через REST API. Приклади основних ендпоїнтів:

- POST /users – створення нового користувача;
- POST /expenses – додавання витрати;
- GET /expenses/user/{userId} – отримання всіх витрат користувача;
- PUT /expenses/{id} – оновлення витрати;
- DELETE /expenses/{id} – видалення витрати;
- POST /categories — створення категорії;
- GET /analytics/budget-exceeded – перевірка перевищення бюджету;
- GET /analytics/category-report – звіт по категоріях;
- GET /analytics/total-month – загальна сума витрат за місяць;
- GET /analytics/by-date – витрати за конкретну дату.

Запити реалізовано згідно з принципами REST:

- GET – для отримання інформації;
- POST – для створення записів;

- PUT – для оновлення;
- DELETE – для видалення.

Усі запити повертають JSON-об’єкти. Відповіді адаптовані до клієнта через DTO.

3.5 Універсальність архітектури

Побудована архітектура дозволяє:

- легко додавати нові сутності та функції (наприклад, регулярні витрати, підписки);
- розширити систему до багатокористувацької (з автентифікацією);
- інтегрувати UI-фронтенд (React, Angular тощо);
- масштабувати та переносити логіку в мікросервіси;
- зручно реалізувати безпеку (Spring Security, JWT).

РЕАЛІЗАЦІЯ ПРОГРАМНОГО ПРОДУКТУ

У цьому розділі описано практичну реалізацію програмного забезпечення відповідно до спроєктованої архітектури. Було реалізовано повний стек CRUD-операцій для всіх бізнес-сутностей, а також аналітичні функції, які дозволяють користувачам здійснювати глибокий фінансовий аналіз. Система реалізована засобами мови Java з використанням Spring Boot, що забезпечує інверсію керування, розділення відповідальностей, а також автоматизоване підключення компонентів за допомогою анотацій.

4.1 Моделі (Model-класи)

Сутності створено з використанням бібліотеки Lombok для скорочення коду, а також JPA-анотацій, які визначають структуру таблиць у базі даних. Усі сутності мають унікальний ідентифікатор типу Long, що генерується автоматично.

Приклад: Expense.java

```
package ua.opnu.pract1shydl.model;

import jakarta.persistence.*;
import lombok.*;

import java.time.LocalDate;

@Entity
@Data
@NoArgsConstructor
@AllArgsConstructor
public class Expense {
  @Id
  @GeneratedValue(strategy = GenerationType.IDENTITY)
  private Long id;

  private double amount;
  private String description;
```

```
  private LocalDate date;

  @ManyToOne
  @JoinColumn(name = "user_id")
  private User user;

  @ManyToOne
  @JoinColumn(name = "category_id")
  private Category category;
}
```

Аналогічно реалізовані класи User, Income, Budget, Category.

У зв'язках між сутностями використовуються анотації @ManyToOne, @OneToMany, що дозволяє будувати правильні реляційні структури в базі даних.

4.2 DTO (Data Transfer Objects)

DTO використовуються для відокремлення внутрішньої структури Model від зовнішнього API. Це дозволяє уникнути надмірної серіалізації, знизити ризики циклічних залежностей і зменшити розмір відповідей API.

Приклад: ExpenseDTO.java

```
package ua.opnu.pract1shydl.dto;

import lombok.Data;

import java.time.LocalDate;

@Data
public class ExpenseDTO {
  private double amount;
  private String description;
  private LocalDate date;
  private Long userId;
  private Long categoryId;
}
```

У DTO-підході:

- у запитах передаються лише ідентифікатори пов’язаних об’єктів (userId, categoryId);
- у відповідях – лише релевантні поля без вкладених списків (expenses, budgets тощо).

Це дає змогу гнучко працювати з API без рекурсивної серіалізації.

4.3 Репозиторії

Кожна сутність має свій інтерфейс репозиторію, який розширює JpaRepository. Це забезпечує доступ до CRUD-операцій без необхідності ручної реалізації запитів.

Приклад: ExpenseRepository.java (імпорти прибрано, щоб код займав менше місця в пояснювальній записці)

```
package ua.opnu.pract1shydl.repository;

import ***

public interface ExpenseRepository extends JpaRepository<Expense,
Long> {
  List<Expense> findByUserId(Long userId);
  List<Expense> findByUserIdAndDateBetween(Long userId, LocalDate
startDate, LocalDate endDate);
  @Query("SELECT e FROM Expense e WHERE e.user.id = :userId AND
FUNCTION('to_char', e.date, 'YYYY-MM') = :month")
  List<Expense> findByUserIdAndMonth(@Param("userId") Long userId,
@Param("month") String month);

  List<Expense> findByUserIdAndDate(Long userId, LocalDate date);
}
```

Для аналітичних запитів додано методи з фільтрацією за датою, місяцем, категоріями тощо.

4.4 Сервіси

Сервіси реалізують логіку обробки запитів. Вони:

- приймають DTO від контролерів;
- перевіряють існування користувача/категорії;
- маплять DTO → Model і навпаки;
- викликають методи репозиторіїв;
- обробляють результати.

Приклад: IncomeService.java

```
package ua.opnu.pract1shydl.service;

import ***

@Service
@RequiredArgsConstructor
public class IncomeService {
  @Autowired
  private IncomeRepository incomeRepository;

  @Autowired
  private UserRepository userRepository;

  public Income createIncome(IncomeDTO dto) {
    User user =
userRepository.findById(dto.getUserId()).orElseThrow();

    Income income = new Income();
    income.setAmount(dto.getAmount());
    income.setSource(dto.getSource());
    income.setDate(dto.getDate());
    income.setUser(user);

    return incomeRepository.save(income);
  }
```

```
  public List<Income> getIncomes(Long userId) { return
incomeRepository.findByUserId(userId); }
  public void deleteIncome(Long id) {
incomeRepository.deleteById(id); }
}
```

Також у сервісах реалізовано аналітичну логіку: підрахунок бюджету, перевищення ліміту, групування по категоріях тощо.

4.5 Контролери

Контролери приймають HTTP-запити, викликають відповідні сервіси та повертають відповіді у форматі JSON. Для кожної сутності реалізовано повний CRUD.

Приклад: ExpenseController.java

```
package ua.opnu.pract1shydl.controller;

import ***

@RestController
@RequestMapping("/expenses")
@RequiredArgsConstructor
public class ExpenseController {

  private ExpenseRepository expenseRepository;
  private CategoryRepository categoryRepository;

  @Autowired
  private ExpenseService expenseService;

  @PostMapping
  @PreAuthorize("hasRole('USER')")
  public Expense create(@RequestBody ExpenseDTO dto) {
    return expenseService.createExpense(dto);
  }

  @GetMapping("/user/{userId}")
  @PreAuthorize("hasRole('USER')")
  public List<Expense> getExpenses(@PathVariable Long userId) {
    return expenseService.getExpenses(userId);
```

```
  }

  @PutMapping("/{id}")
  @PreAuthorize("hasRole('USER')")
  public Expense update(@PathVariable Long id, @RequestBody
ExpenseDTO dto) {
    return expenseService.updateExpense(id, dto);
  }

  @DeleteMapping("/{id}")
  @PreAuthorize("hasRole('USER')")
  public void deleteExpense(@PathVariable Long id) {
expenseService.deleteExpense(id); }
}
```

Усі запити дотримуються REST-конвенцій. Для аналітики використовується окремий AnalyticsController.

4.6 Реалізація реєстрації та автентифікації користувача

Друга частина курсової роботи з реалізації системи передбачала захист REST API, реєстрацію та автентифікацію користувачів. Це дозволяє забезпечити контрольований доступ до персональних фінансових даних, зберігати конфіденційність користувачів, а також розмежовувати права доступу.

4.6.1 Реєстрація та автентифікація користувачів

Реєстрація:

Користувач може зареєструватися, передавши в POST /api/auth/register поля: username, email та password.

Після перевірки унікальності імені користувача та електронної пошти, система зберігає новий обліковий запис у таблиці users, хешуючи пароль за допомогою BCrypt.

Авторизація:

Користувач входить у систему через POST /api/auth/login, передаючи логін і пароль. У відповідь сервер генерує JWT-токен, який передається клієнту та зберігається в браузері або додатку.

```
{
  "accessToken": "eyJhbGciOiJIUzUxMiIsInR5cCI6...",
  "tokenType": "Bearer",
  "expiresIn": 3600,
  "user": {
    "id": 1,
    "username": "demo_user",
    "email": "demo@example.com",
    "role": "USER"
  }
}
```

4.6.2 Реалізація JWT (JSON Web Token)

Створення токена:

Після успішного входу, сервер формує токен із даними користувача (id, роль, email) і підписує його секретним ключем.

Перевірка токена:

Усі захищені ендпоїнти перевіряють наявність токена в заголовку Authorization. Токен обробляється у фільтрі JwtAuthenticationFilter, який:

- виділяє токен із запиту;
- перевіряє його справжність;
- витягує id користувача;
- встановлює користувача в контекст безпеки Spring.

4.6.3 Авторизація через OAuth2 (Google)

Реалізована підтримка входу через Google OAuth2. Коли користувач авторизується через Google:

- система отримує email та ім’я користувача;
- створює нового користувача в базі, якщо такого ще немає;
- видає JWT-токен із цими даними.

Це реалізовано в класі OAuth2SuccessHandler, який активується після успішної авторизації:

```
String token = jwtTokenProvider.generateToken(user);
response.getWriter().write(mapper.writeValueAsString(Map.of("token",
token, "type", "Bearer")));
```

4.6.4 Конфігурація безпеки (Spring Security)

У конфігурації SecurityConfig визначено:

- які ендпоїнти відкриті (/api/auth/**, /oauth2/**);
- які вимагають роль USER (/expenses/**, /analytics/**, тощо);
- як обробляються винятки;
- як обробляються сесії (stateless через токен);
- як підключається обробник OAuth2-успіху.

Також інтегровано:

- JwtAuthenticationEntryPoint – обробка помилок доступу;
- DaoAuthenticationProvider – логін і пароль;
- JwtAuthenticationFilter – JWT фільтрація.

4.6.5 Глобальна обробка винятків

Усі помилки обробляються централізовано у GlobalExceptionHandler:

- помилки валідації (MethodArgumentNotValidException);

- неправильний логін або пароль;
- доступ до неавторизованих ресурсів;
- помилки пошуку (ResourceNotFoundException).

Усі відповіді мають уніфікований формат ApiResponse, наприклад:

```
{
  "success": false,
  "message": "Username is already taken!",
  "error": "Bad Request"
}
```

4.7 Конфігурація, підключення до бази даних та хостинг

Усі параметри зберігаються у файлі application.properties. Його вміст:

```
spring.application.name=Kurs_Shydl
spring.datasource.url=jdbc:postgresql://dpg-d14asanfte5s73clkmsg-
a.frankfurt-postgres.render.com/kurs_shydl
spring.datasource.username=kurs_shydl_user
spring.datasource.password=ygH6nw5cEvHPL6J9VJdhUO4aD1kGbVbF
spring.datasource.driver-class-name=org.postgresql.Driver
spring.jpa.hibernate.ddl-auto=update
spring.jpa.show-sql=true
spring.jpa.properties.hibernate.dialect=org.hibernate.dialect.Postgr
eSQLDialect

//JWT-конфігурація
app.jwt.secret=z8P9vA6eR3y7L0qB5f2nG6uXwM3dS9cH1pL4vJ7rZ2uB8tV5yE3hK
0xN6mT9jC1GFRTS
app.jwt.expiration=86400000

spring.security.oauth2.client.registration.google.client-
id=232673372869-
qt3g23b09v8mq03eljosvejn5bg20p2g.apps.googleusercontent.com
spring.security.oauth2.client.registration.google.client-
secret=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
spring.security.oauth2.client.registration.google.scope=email,profil
e

spring.security.oauth2.client.provider.google.authorization-
uri=https://accounts.google.com/o/oauth2/v2/auth
```

```
spring.security.oauth2.client.provider.google.token-
uri=https://oauth2.googleapis.com/token
spring.security.oauth2.client.provider.google.user-info-
uri=https://www.googleapis.com/oauth2/v3/userinfo
spring.security.oauth2.client.provider.google.user-name-
attribute=sub
```

Також підключено:

- автоматичну генерацію БД (ddl-auto=update);
- відображення SQL-запитів у консолі;
- коректну обробку дат (через Jackson).

ТЕСТУВАННЯ ТА НАЛАГОДЖЕННЯ

Якість програмного забезпечення значною мірою залежить від ефективності його тестування. У процесі розробки системи обліку витрат домогосподарства велика увага приділялась перевірці коректної роботи функціональності, точності аналітичних обчислень, захисту доступу до даних, обробці помилок та поведінки системи у граничних ситуаціях.

Тестування проводилось мануально (ручне тестування) за допомогою Postman, а також шляхом локального запуску програми з логуванням результатів. У перспективі можливе додавання модульних тестів (JUnit, Mockito), проте в межах цієї курсової акцент був зроблений на інтеграційне тестування поведінки системи в реальних сценаріях.

5.1 Тестування функціональності REST API

CRUD-операції для сутностей:

Було зроблено та перевірено виконання основних запитів:

- створення (POST);
- отримання (GET);
- оновлення (PUT);
- видалення (DELETE).

До прикладу успішних запитів – створення витрати (рис. 5.1):

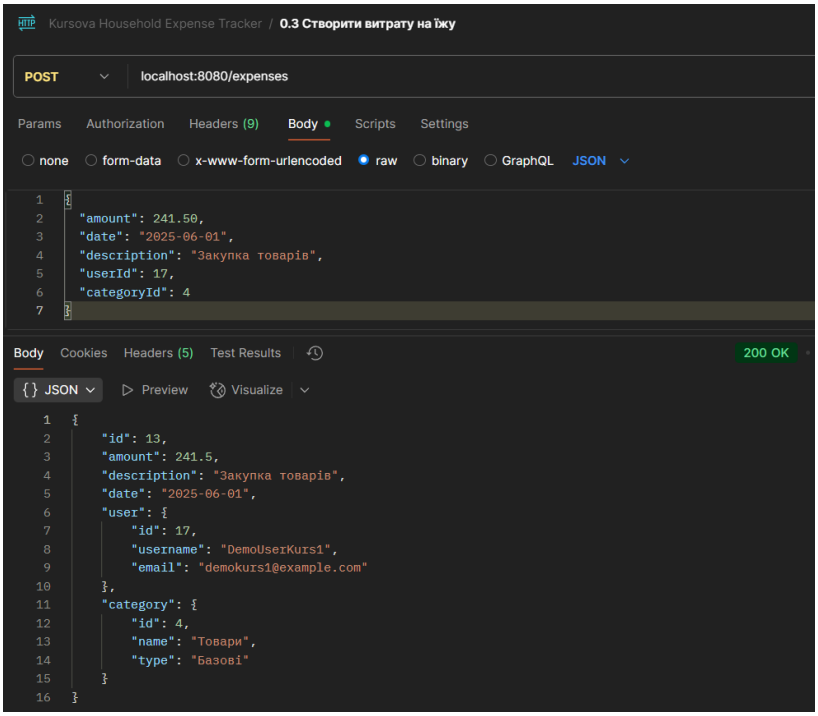

Рис. 5.1 – Створення витрати

5.2 Тестування аналітичної логіки (пункти 19–20)

Тестування запитів у Postman (для прикладу наведено запити з п. 19 – 20):

5.2.1 Отримати перевищення бюджету (пункт 19)

Задана «стеля» бюджету в прикладі: 12000 (рис 5.2)

Рис 5.2 – Значення потрібного для нас бюджету

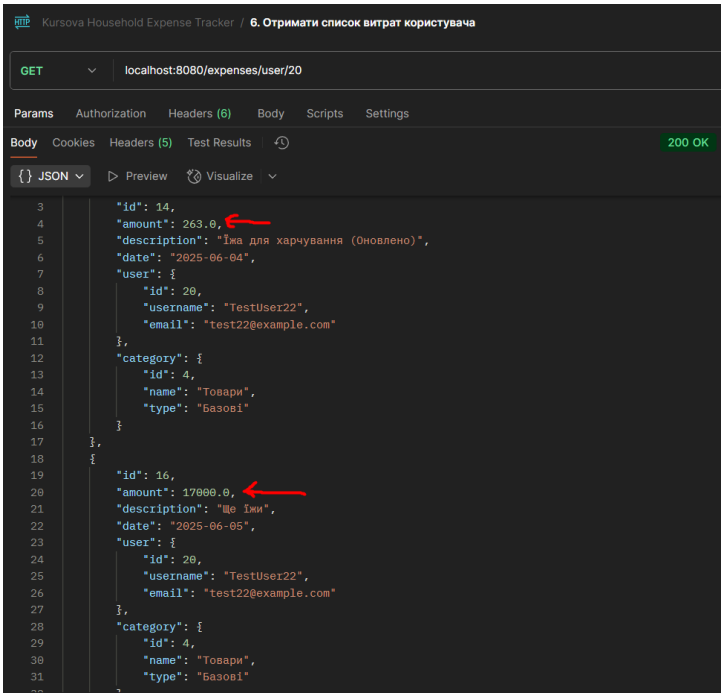

Рис. 5.3 – Виведення всіх наявних витрат одного користувача

Тобто, ліміт – 12000, а сума всіх витрат – 17263. Вводимо запит за визначення того, чи перевищує загальна сума витрат на місяць сам бюджет. Результат показано на рис. 5.4. Як ми бачимо, вивело значення «true», себто код впорався зі своєю задачею.

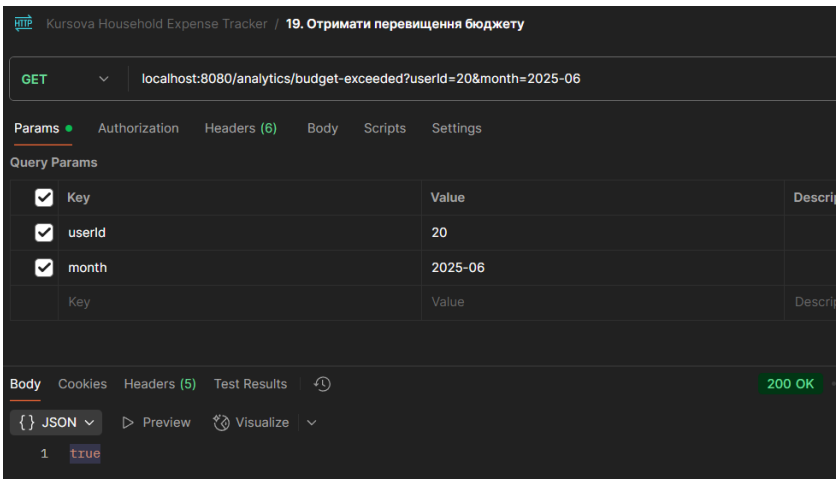

Рис 5.4 – Результат виведення запиту з пункту 19

Для наглядності роботи коду, зробимо умови, при яких сума витрат за певний місяць буде меншою за визначений ліміт бюджету. Оновимо витрату з сумою 17000, змінивши на 11500 (рис. 5.5).

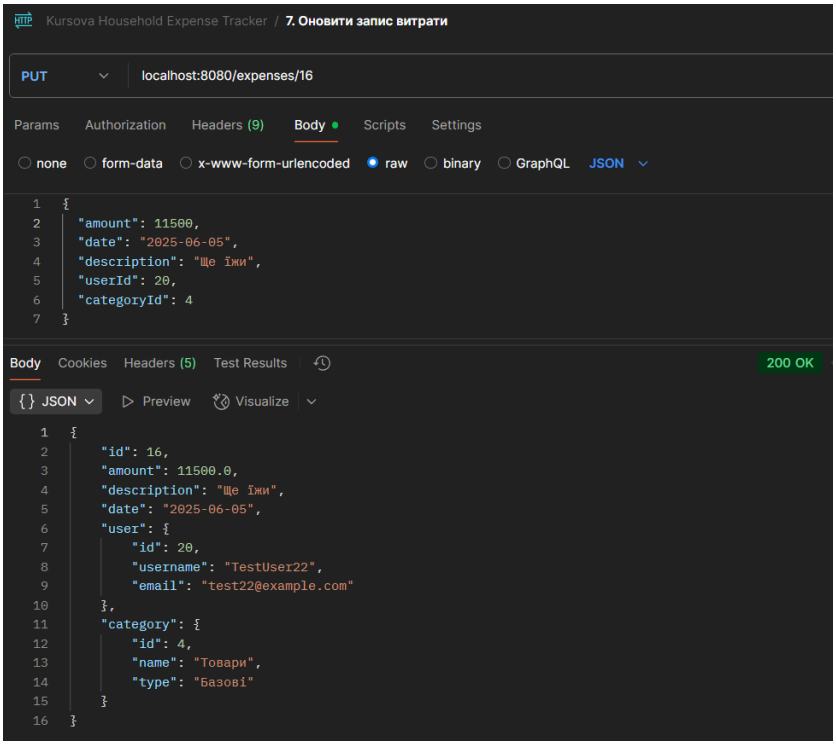

Рис 5.5 – Оновлення витрати (суму змінено з 17000 на 11500)

Тепер сума витрат за обраний нами місяць буде 11763 (менше, ніж ліміт бюджету). Повторимо запит перевірки на перевищення бюджету. Як і було очікувано, він показав «false».

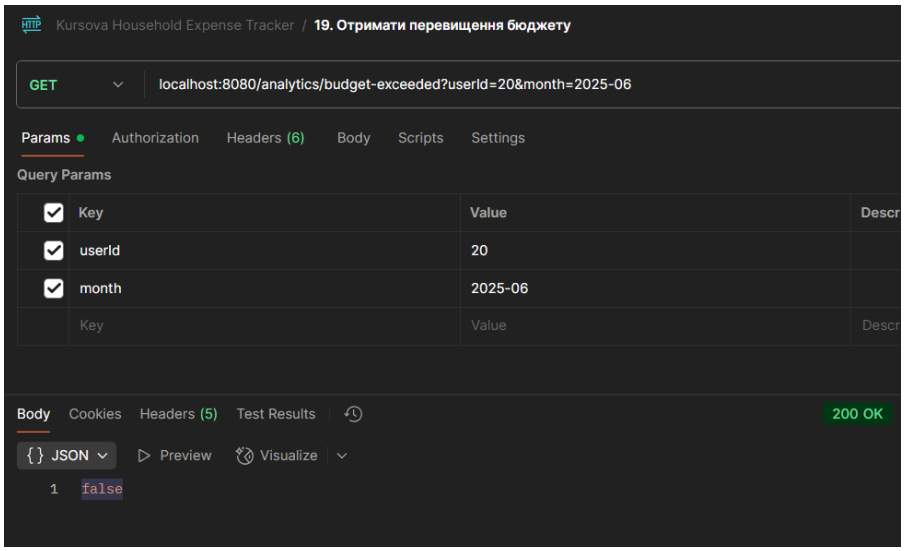

Рис 5.6 – Повторення запиту на перевірку на перевищення бюджету

5.2.2 Згенерування звіту по категоріях

Зробімо запит на генерування звіту по категоріях (сума всіх витрат, прив’язаних до тієї чи іншої категорії та користувача) (рис. 5.7)

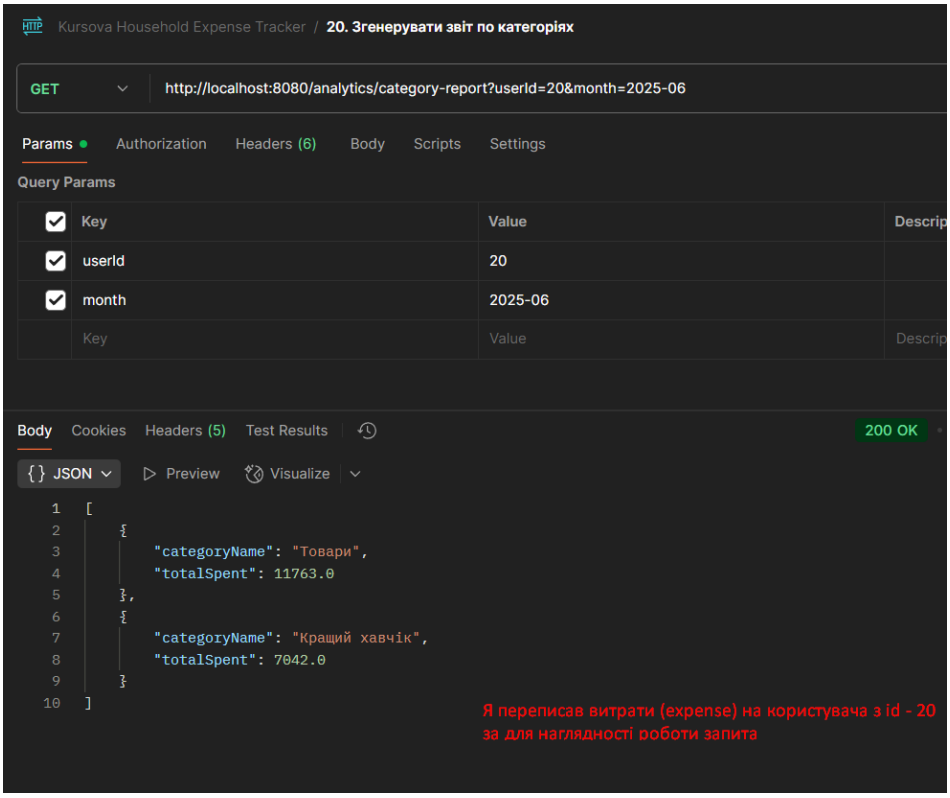

Рис 5.7 – Результат виконання запиту на генерацію

5.3 Тестування автентифікації та безпеки

У другій частині курсової роботи було реалізовано:

- реєстрацію через POST /api/auth/register;
- логін з генерацією JWT-токена;
- захист усіх приватних маршрутів (тільки авторизовані користувачі можуть отримати дані);
- логін через Google OAuth2.

Перевірка:

1) Реєстрування нового користувача

Заходимо в Postman, робимо POST-запит «/api/auth/register». Також у raw пишемо дані нового користувача і запускаємо (рис. 5.8).

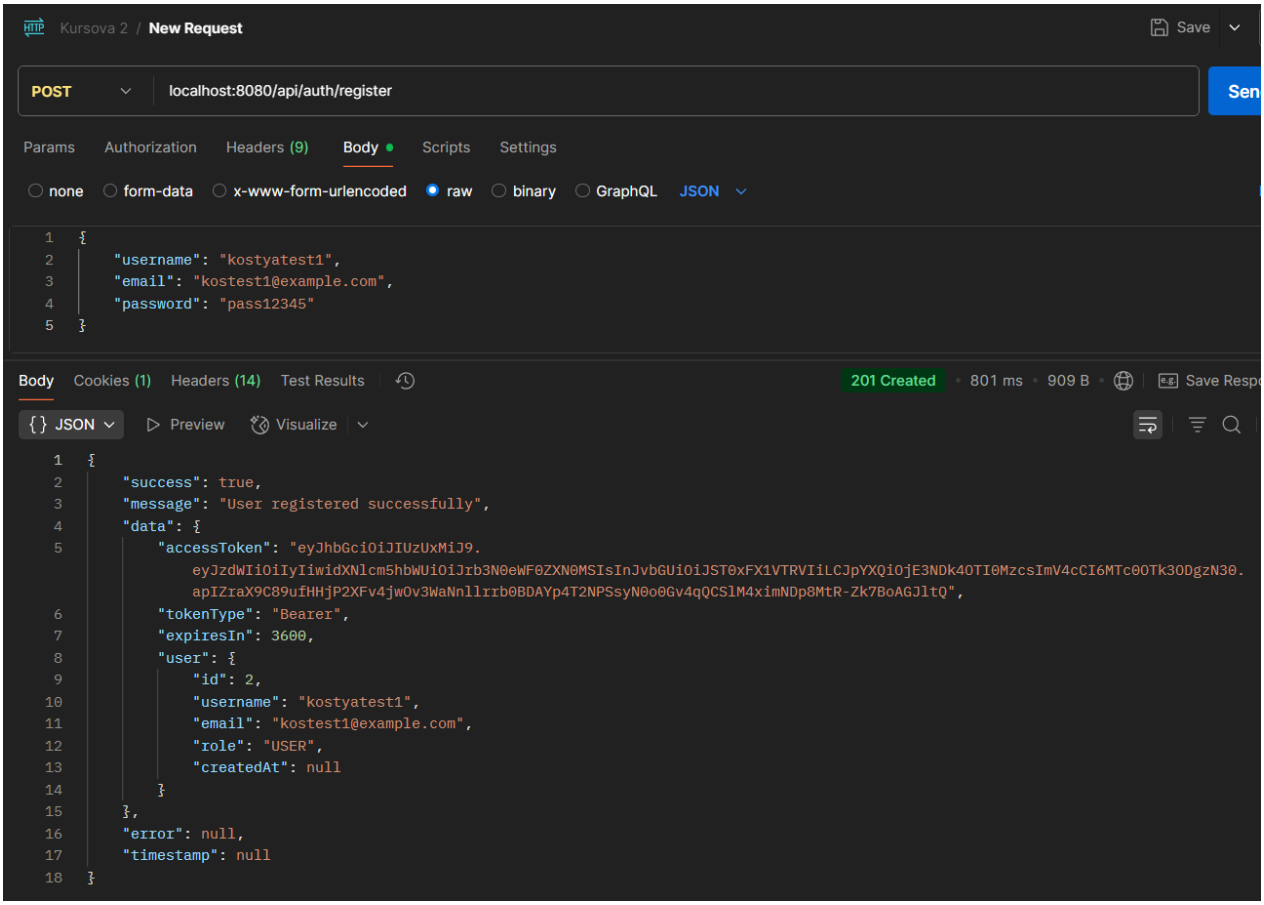

Рис 5.8 – Реєстрація користувача

2) Вхід/Автентифікація

Робимо POST-запит «/api/auth/login». Також у raw пишемо дані зареєстрованого користувача і запускаємо. Для наглядності роботи коду я спочатку ввів неправильні дані користувача, а потім правильні (рис 5.9 – 5.10).

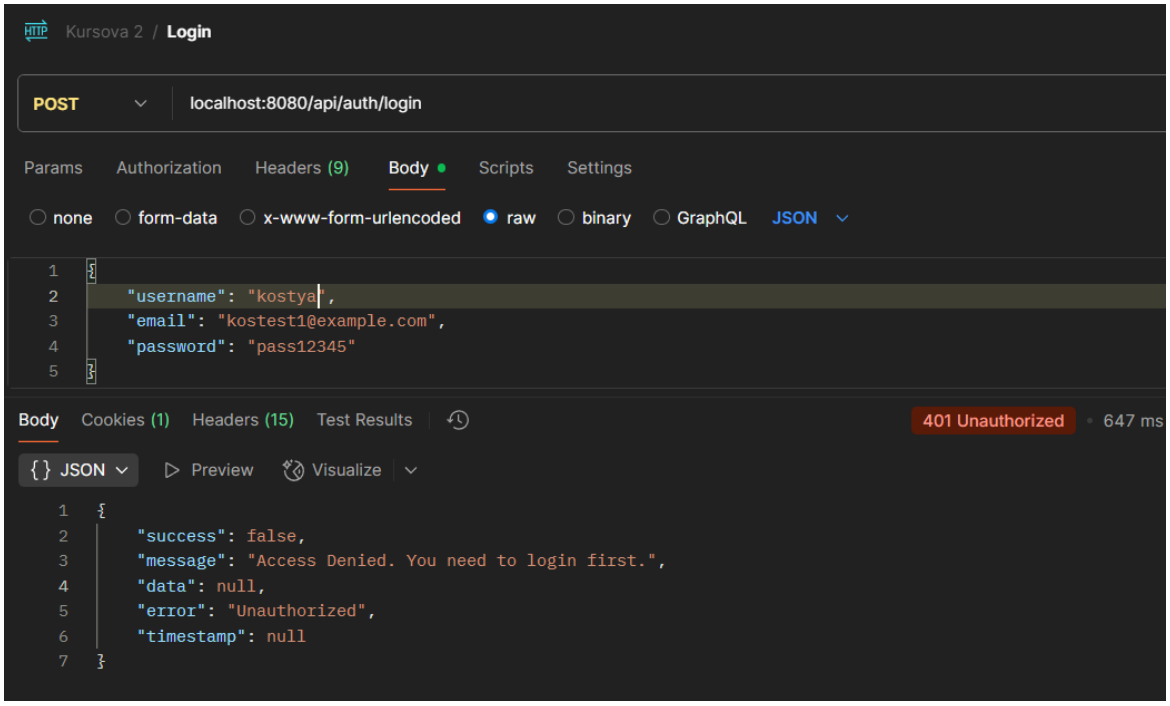

Рис 5.9 – Спроба входу (невдала, бо ми ввели неправильний username)

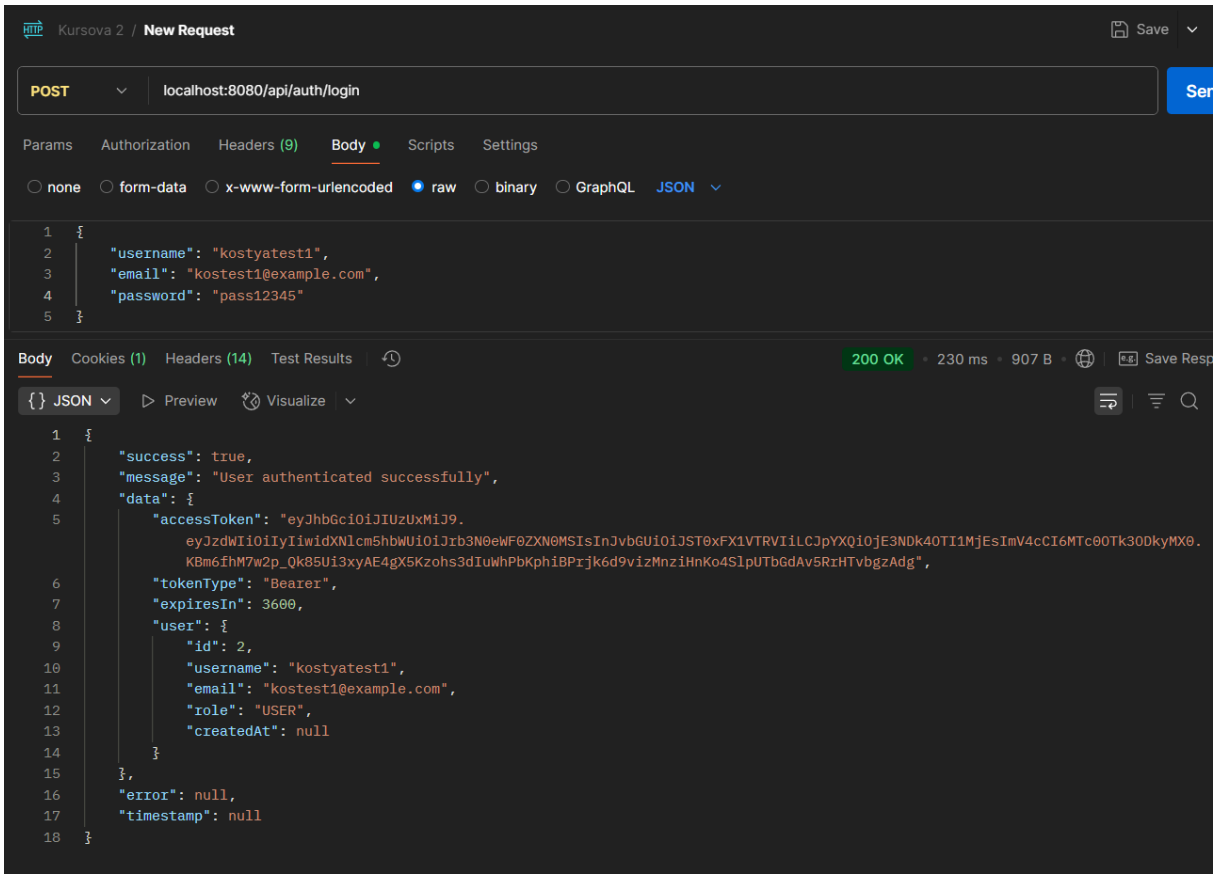

Рис 5.10 – Спроба автентифікації (цього разу вдала)

3) Автентифікація за допомогою JWT-токена

Заходимо на будь-який запит (у нашому випадку отримання даних конкретного користувача). Якщо ми спробуємо це зробити без попередньої автентифікації, то нам видасть помилку. Коли ми востаннє автентифіковувалися, нам видало дані користувача, зокрема й «accesstoken». Скопіюємо та використаємо його. Якщо ми введемо неправильний токен, то нам видасть помилку (рис. 5.11), але якщо правильний, то нам вдасться виконати обраний нами запит (рис. 5.12).

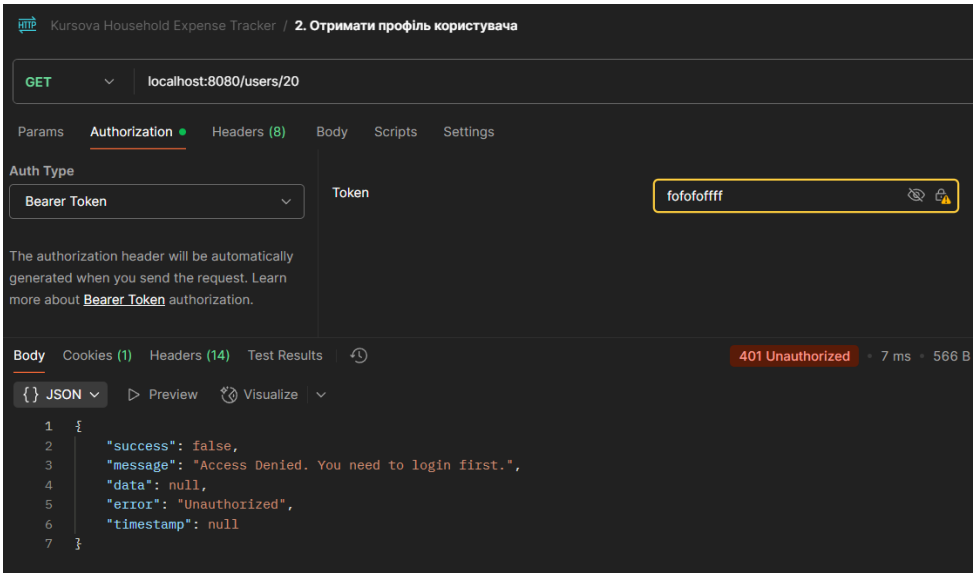

Рис 5.11 – Помилка при автентифікації з неправильним токеном

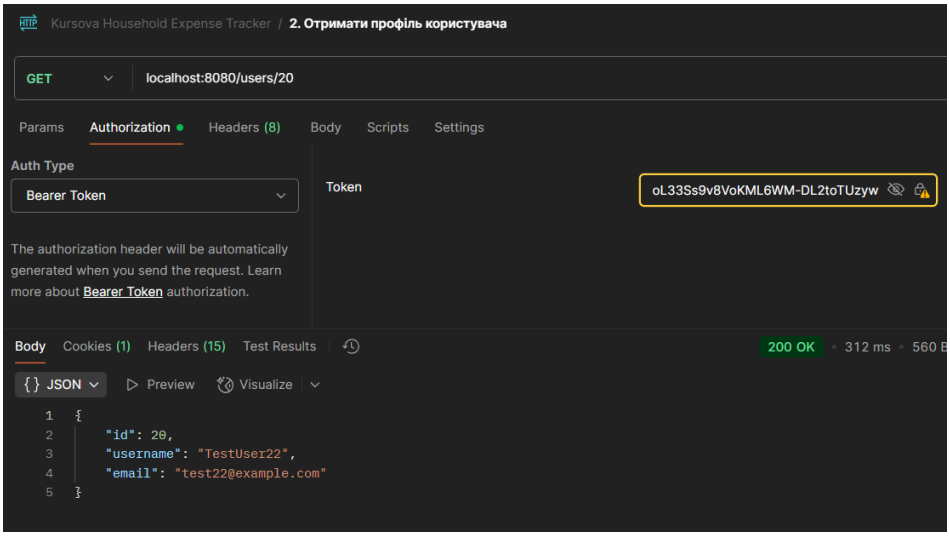

Рис 5.12 – Вдала автентифікація за допомогою токена

4) OAuth2

Заходимо до браузера «Google» і заходимо за посиланням: http://localhost:8080/oauth2/authorization/google

Обираємо акаунт, і нам виводить токен акаунту, копіюємо його (рис 5.13 – 5.15). Заходимо знову до Postman, вводимо цей токен і нам вдається автентифікуватися (рис 5.16).

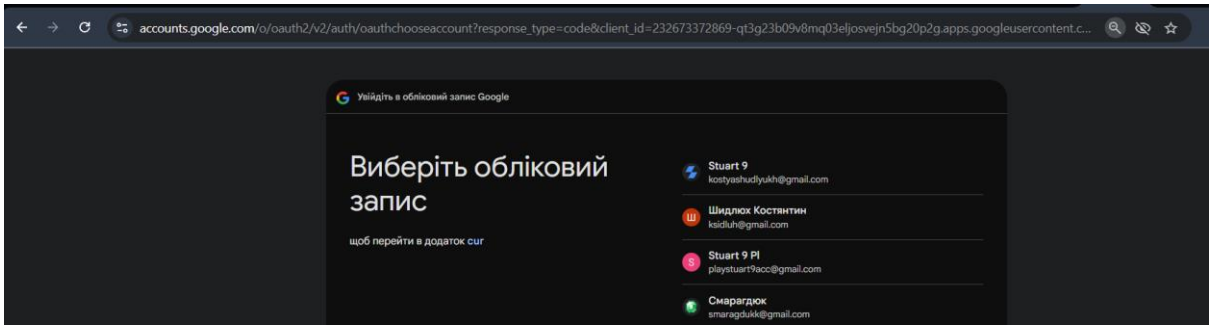

Рис 5.13 – Вибір акаунта

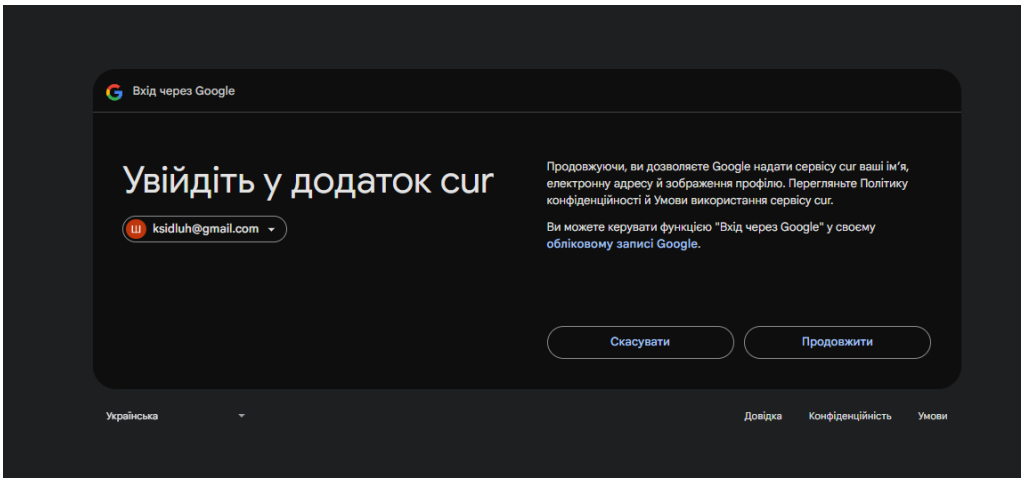

Рис 5.14 – Пряме продовження рис. 5.13

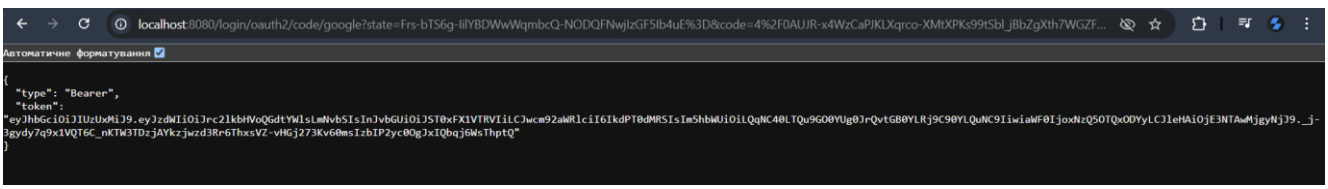

Рис 5.15 – Вдала автентифікація за допомогою токена

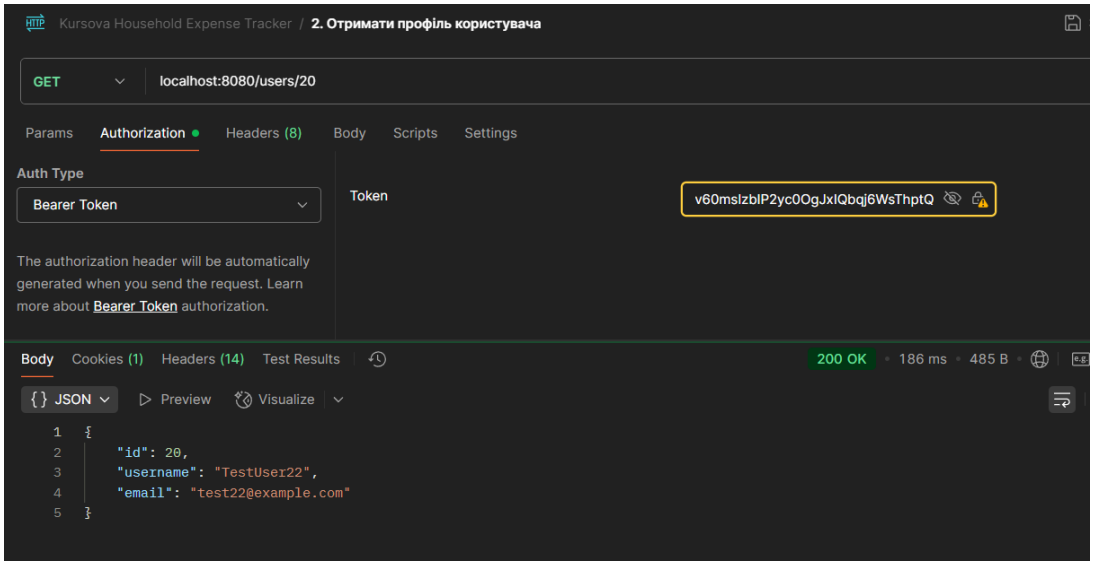

Рис 5.16 – Вдала автентифікація за допомогою токена, отриманого за допомогою OAuth2

ЗАГАЛЬНІ ВИСНОВКИ

У процесі виконання курсової роботи було реалізовано повнофункціональний веб-застосунок для обліку витрат домогосподарства. Цей застосунок дозволяє користувачам вести облік особистих фінансів, зберігати та переглядати витрати й доходи, класифікувати витрати за категоріями, планувати бюджет, отримувати аналітичні звіти, а також забезпечує захищений доступ до системи з використанням сучасних методів автентифікації.

У межах першої частини курсової роботи було спроєктовано модель предметної області, яка охоплює п’ять ключових сутностей: користувачі (User), витрати (Expense), доходи (Income), категорії (Category) та бюджети (Budget). Між цими сутностями встановлено логічні зв’язки за допомогою анотацій JPA, що дозволяє реалізувати реляційну структуру даних у базі PostgreSQL. Було створено окремі DTO для кожної сутності з метою контролю серіалізації та забезпечення чистоти REST API.

Архітектура застосунку реалізована за принципом багаторівневої моделі MVC з чітким поділом на шари: контролери (обробка HTTP-запитів), сервіси (бізнес-логіка), репозиторії (доступ до БД), сутності та DTO. В рамках проєкту реалізовано повний набір CRUD-операцій для всіх сутностей, а також аналітичні функції: визначення перевищення бюджету, отримання звіту по категоріях, підрахунок суми витрат за місяць, середньої витрати на день, виведення найбільших витрат місяця, визначення залишку бюджету та перегляд витрат за датою. Всі ці функції протестовані за допомогою Postman, що дозволило переконатися у правильності логіки та відповідності результатів поставленим завданням.

У другій частині курсової роботи було реалізовано реєстрацію та автентифікацію користувачів. Система підтримує реєстрацію з перевіркою унікальності логіна та емейлу, а також авторизацію через логін і пароль із генерацією JWT-токена. Додатково реалізовано підтримку авторизації через

Google за допомогою протоколу OAuth2, що дозволяє користувачам входити до системи без створення окремого облікового запису. Для захисту API використано Spring Security, де всі маршрути, крім автентифікації, закриті та вимагають JWT-токена. Права доступу обмежені за роллю користувача. Крім того, додано централізовану обробку помилок, валідацію запитів та обробку ситуацій типу “не знайдено ресурс” або “невалідні дані”.

Реалізація другої частини забезпечила безпечний і зручний механізм доступу до системи, унеможлививши несанкціоноване втручання в персональні фінансові дані. Усі запити протестовані на успішні та граничні кейси, підтверджуючи стабільну роботу застосунку, точність аналітичних обчислень і правильну поведінку системи в разі помилок.

У результаті виконання курсової роботи було досягнуто головної мети – створення стабільної, безпечної, розширюваної веб-системи, що вирішує практичну задачу обліку витрат і доходів домогосподарства. Одержаний результат має як навчальну, так і прикладну цінність. Застосунок може використовуватись як реальний продукт або як фундамент для подальшого розвитку.

Під час розробки проєкту було значно поглиблено практичні знання з Java, Spring Boot, REST API, JPA, JWT, OAuth2, Postgres, а також сформовано розуміння архітектури сучасного бекенд-застосунку та принципів безпечного зберігання та обробки даних.

Отже, виконана курсова робота демонструє не лише успішну реалізацію поставленого завдання, а й вказує на готовність автора працювати із сучасними фреймворками, створювати реальні проєкти та вдосконалювати систему в майбутньому.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Методичні вказівки до виконання курсової роботи з дисципліни «Об’єктно-орієнтоване програмування» для здобувачів інституту комп’ютерних систем за спеціальністю 122 Комп'ютерні науки, освітня програма «Комп’ютерні науки» / Укладачі: М.А. Годовиченко. – Одеса: Національний університет «Одеська Політехніка», 2024. – 59 с.

2. Spring Boot Documentation Site. URL: https://docs.spring.io/spring-boot/docs/current/reference/html/ (дата звернення: 25.04.2025).

3. Spring Security Reference. URL: https://docs.spring.io/spring-security/reference/index.html (дата звернення: 21.04.2025).

4. Spring Data JPA Reference Documentation. URL: https://docs.spring.io/spring-data/jpa/docs/current/reference/html/ (дата звернення: 21.04.2025).

5. JWT.io – Introduction to JSON Web Tokens. URL: https://jwt.io/introduction/ (дата звернення: 02.06.2025).

6. Official OAuth 2.0 Authorization Framework. URL: https://oauth.net/2/ (дата звернення: 06.06.2025).

7. Lombok Project Documentation. URL: https://projectlombok.org/ (дата звернення: 20.04.2025).

8. Spring MVC. Створення веб додатків на Java. URL: https://youtu.be/dsGq6noTUs0?si=-QoSwdyohtCaJ7sg (дата звернення: 14.04.2025).

ДОДАТКИ

Тут буде продемонстровано роботу всіх запитів із першої частини курсової роботи (пунктів 19 – 20 тут немає, тому що їх роботу вже було показано раніше):

1) Зареєстрування користувача (user) (рис. 6.1)

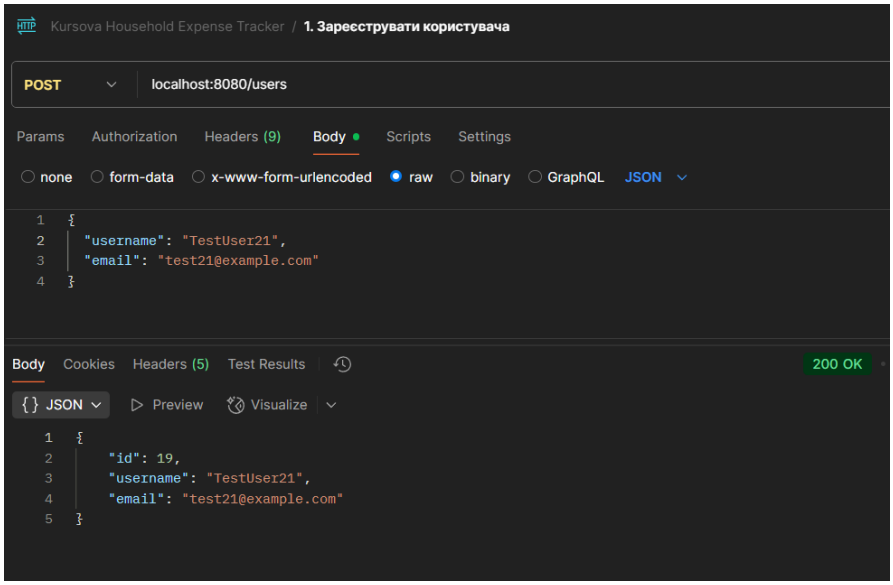

Рис 6.1 – Результат виконання запиту з пункту 1

2) Отримування профілю користувача (рис. 6.2)

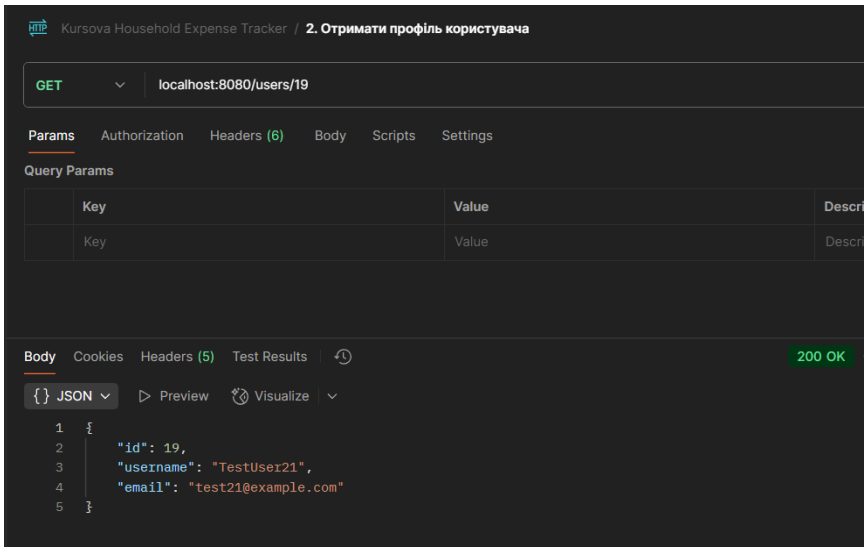

Рис 6.2 – Результат виконання запиту з пункту 2

3) Оновлення профілю користувача (рис. 6.3)

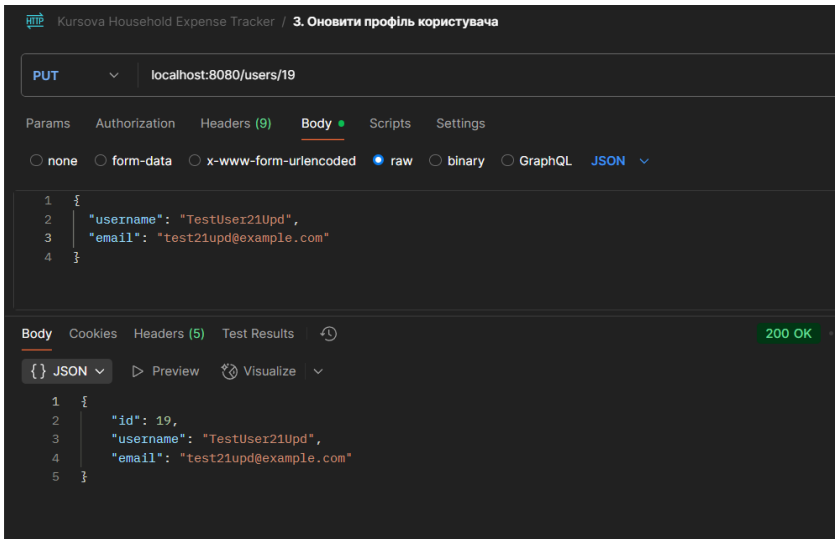

Рис 6.3 – Результат виконання запиту з пункту 3

4) Видалення користувача (рис. 6.4 – 6.5)

На рисунку 6.5 показано, що при спробі вивести дані цього користувача нічого не виводиться, бо ми його видалили.

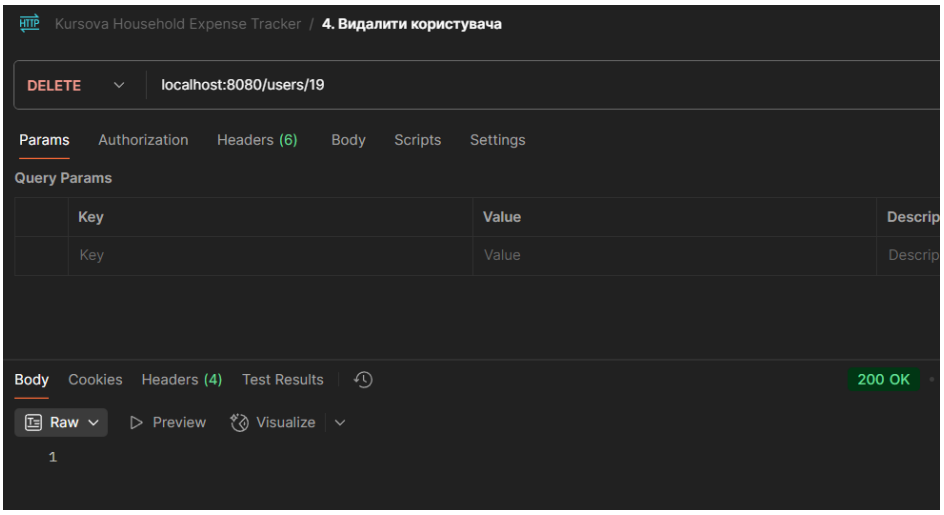

Рис 6.4 – Результат виконання запиту з пункту 4

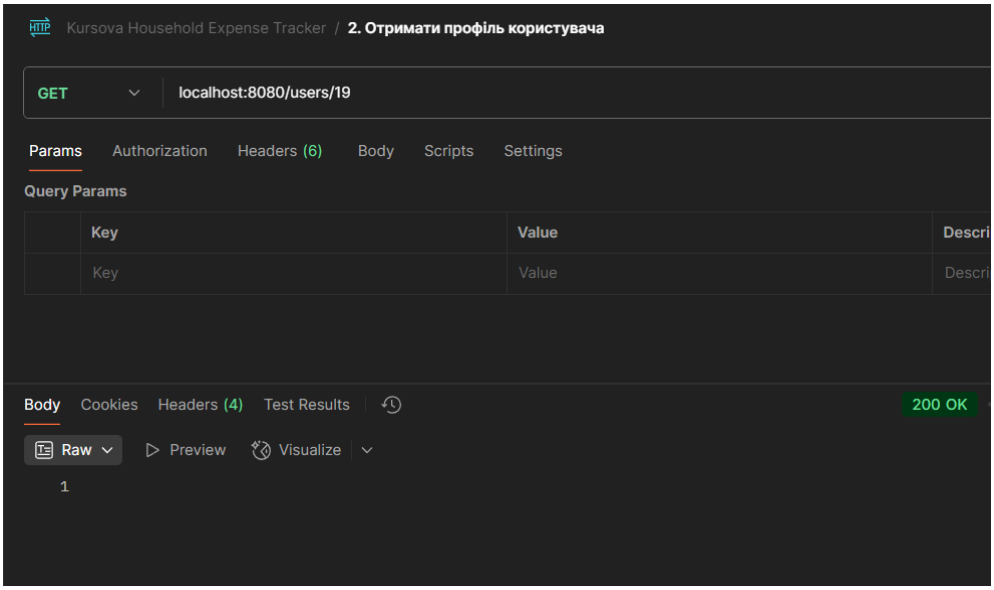

Рис 6.5 – Результат виконання запиту з пункту 4

5) Додавання витрати (expense) (рис. 6.6)

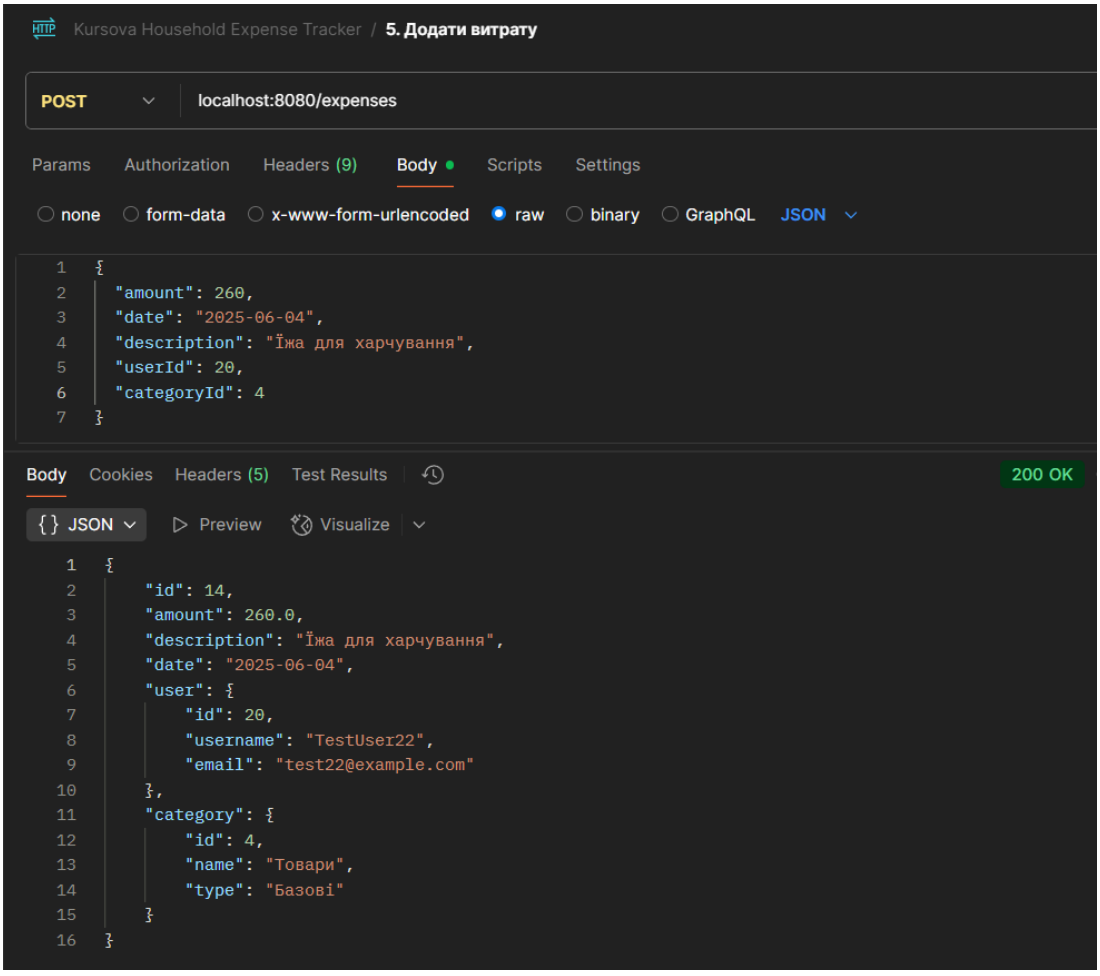

Рис 6.6 – Результат виконання запиту з пункту 5

6) Отримання списку витрат користувача (рис. 6.7 – 6.8)

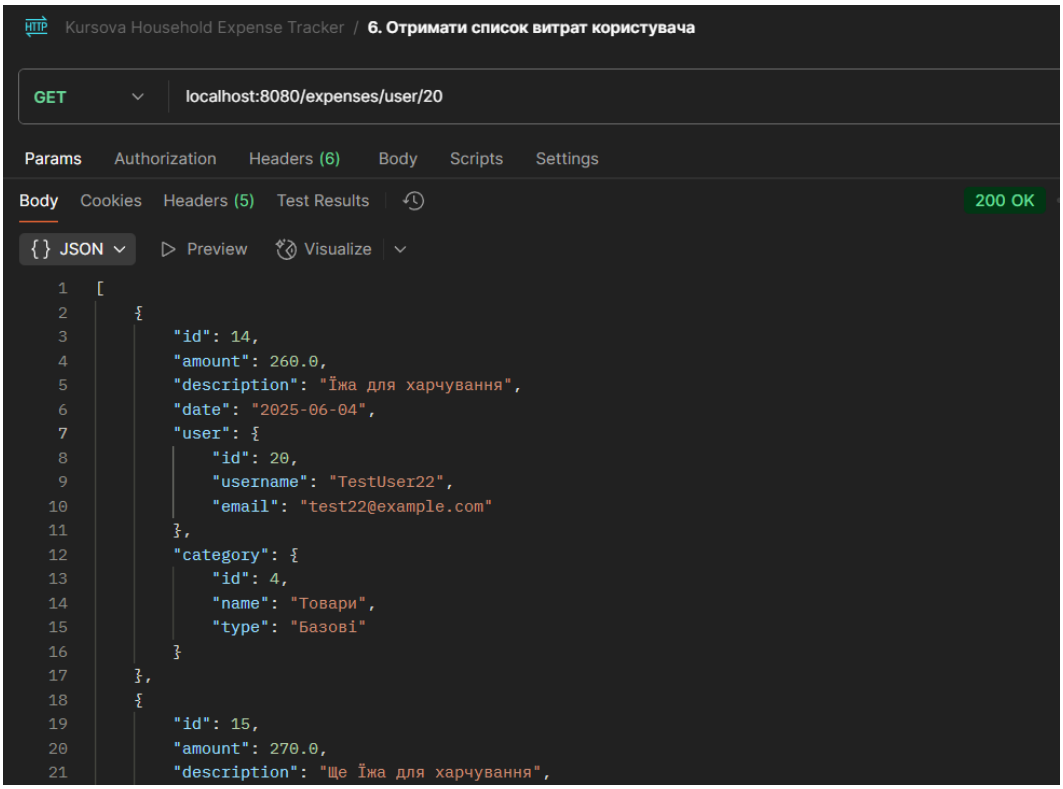

Рис 6.7 – Результат виконання запиту з пункту 6

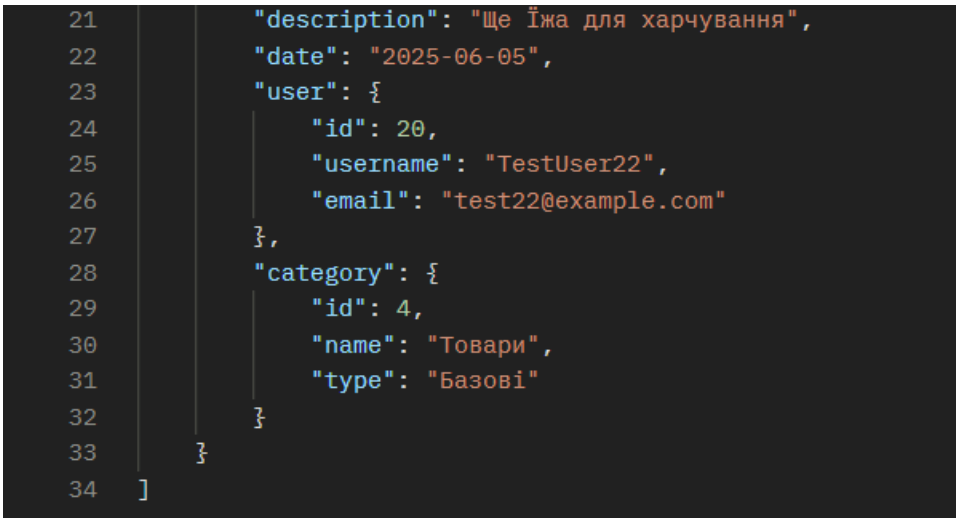

Рис 6.8 – Результат виконання запиту з пункту 6 (пряме продовження рисунка 6.7)

7) Оновлення запису витрати (рис. 6.9)

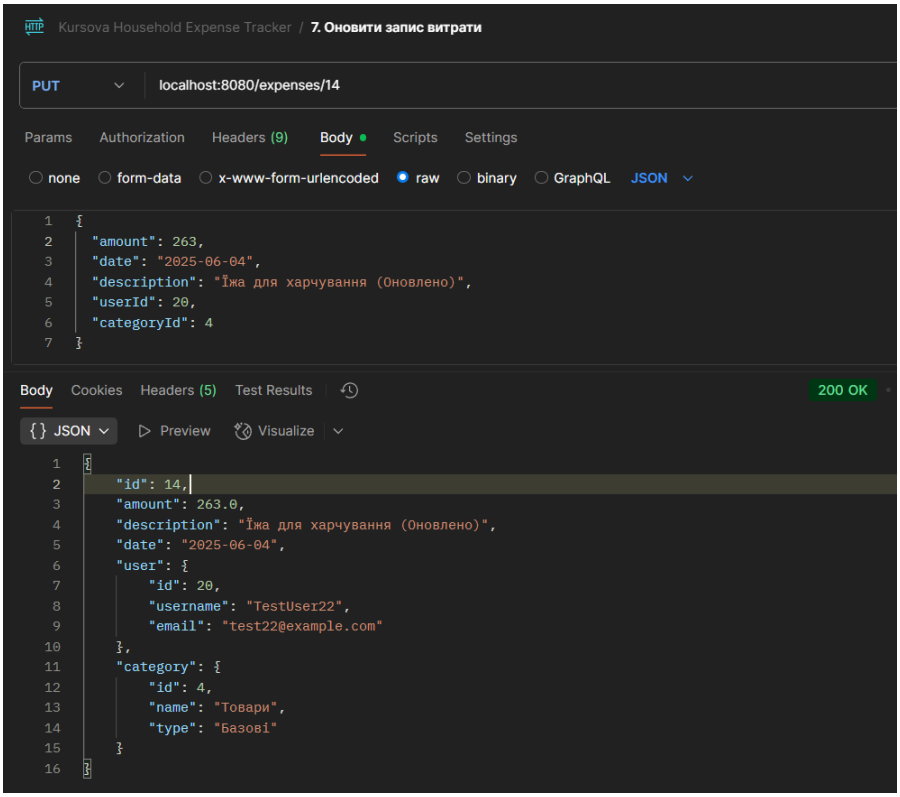

Рис 6.9 – Результат виконання запиту з пункту 7

8) Видалення витрати (рис. 6.10 – 6.11)

На рисунку 6.11 показано, що при спробі вивести дані цієї витрати нічого не виводиться, бо ми її видалили (спочатку були витрати з id – 14 та 15, тепер лише з id – 14).

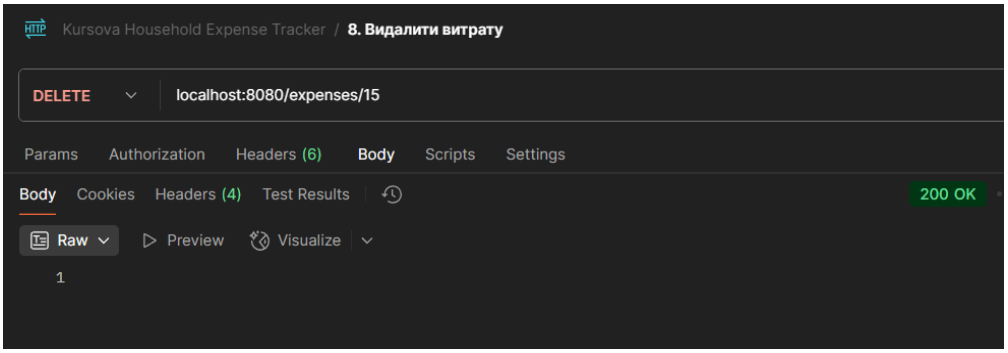

Рис 6.10 – Результат виконання запиту з пункту 8

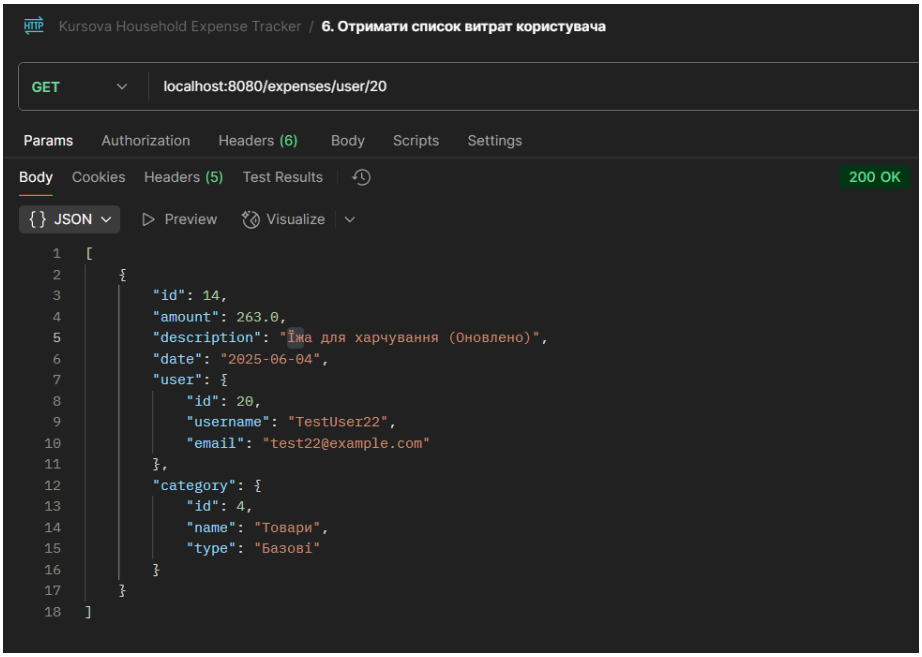

Рис 6.11 – Результат виконання запиту з пункту 8

9) Додавання категорії витрат (category) (рис. 6.12)

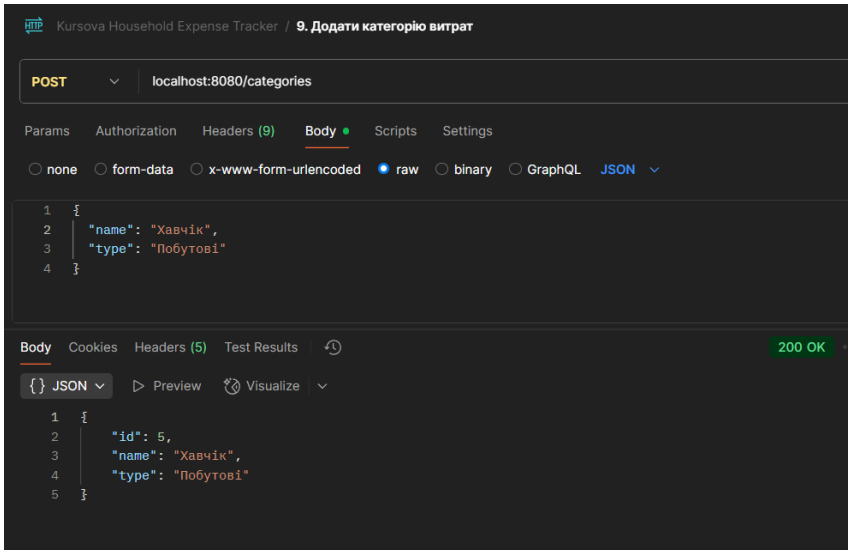

Рис 6.12 – Результат виконання запиту з пункту 9

10) Отримання категорій (рис. 6.13)

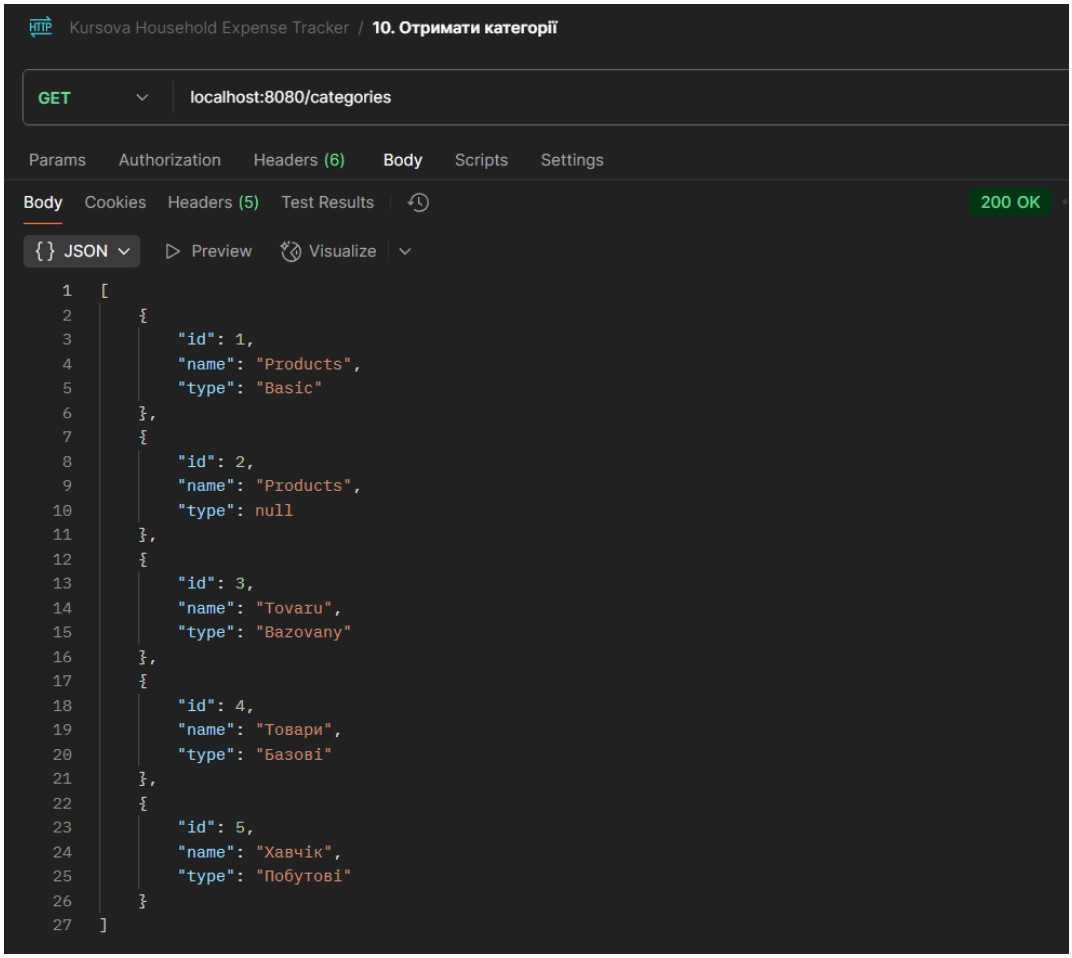

Рис 6.13 – Результат виконання запиту з пункту 10

11) Оновлення категорії (рис. 6.14 – 6.15)

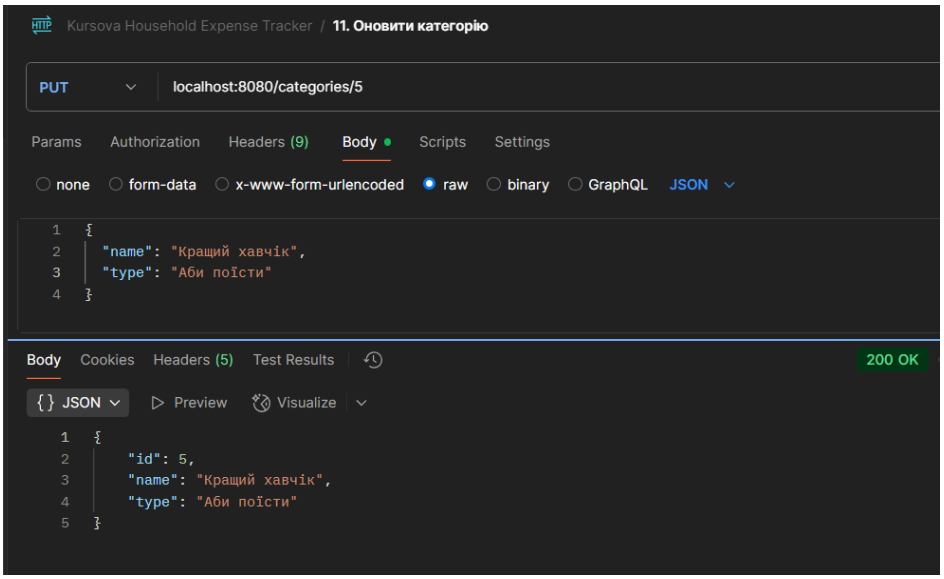

Рис 6.14 – Результат виконання запиту з пункту 11

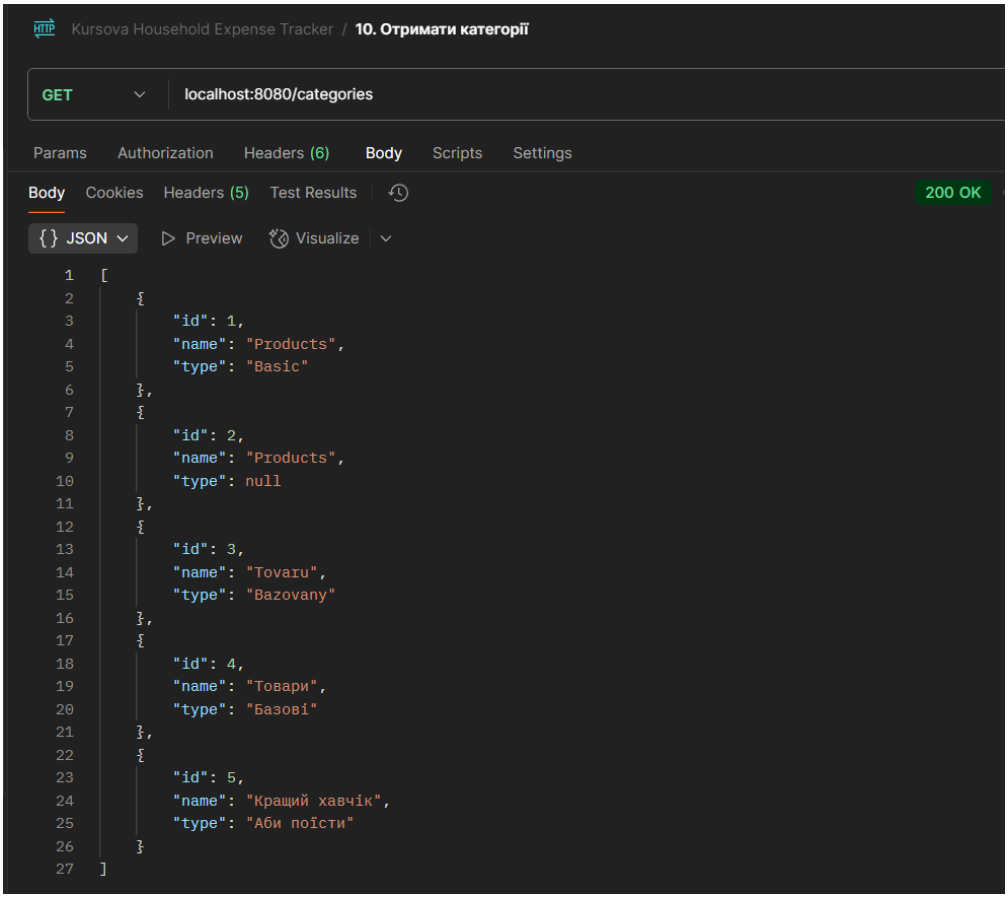

Рис 6.15 – Демонстрація результату виконання запиту з пункту 11

12) Видалення категорії (рис. 6.16 – 6.17)

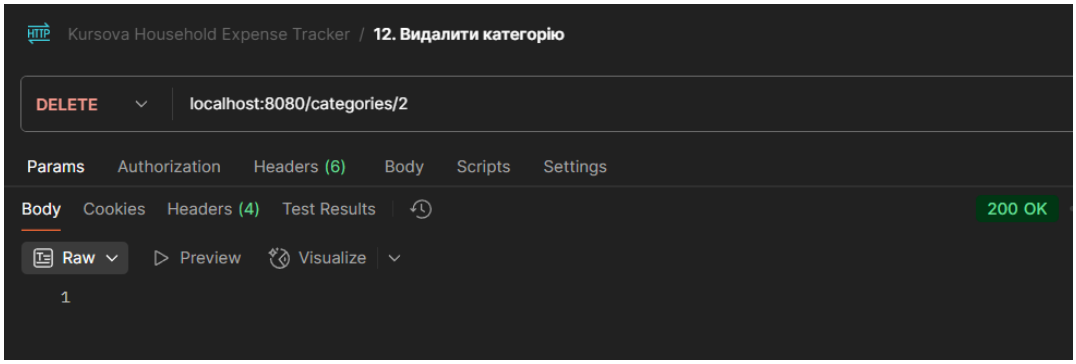

Рис 6.16 – Результат виконання запиту з пункту 12

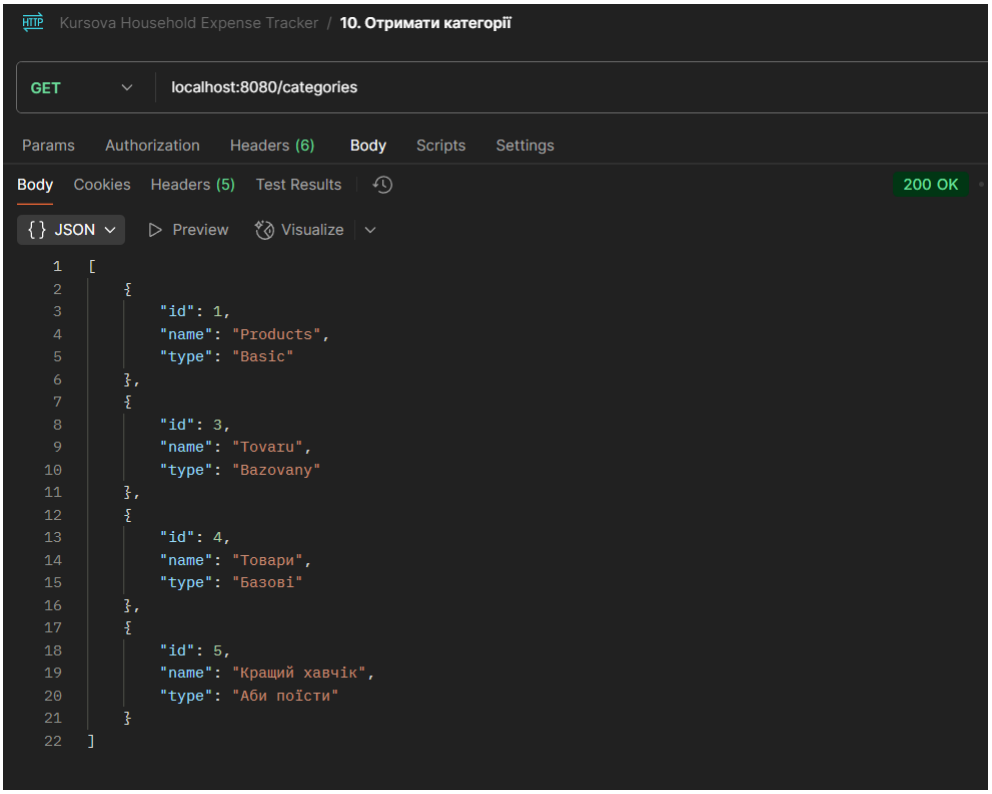

Рис 6.17 – Демонстрація результату виконання запиту з пункту 12

13) Додавання запису доходу (income) (рис. 6.18)

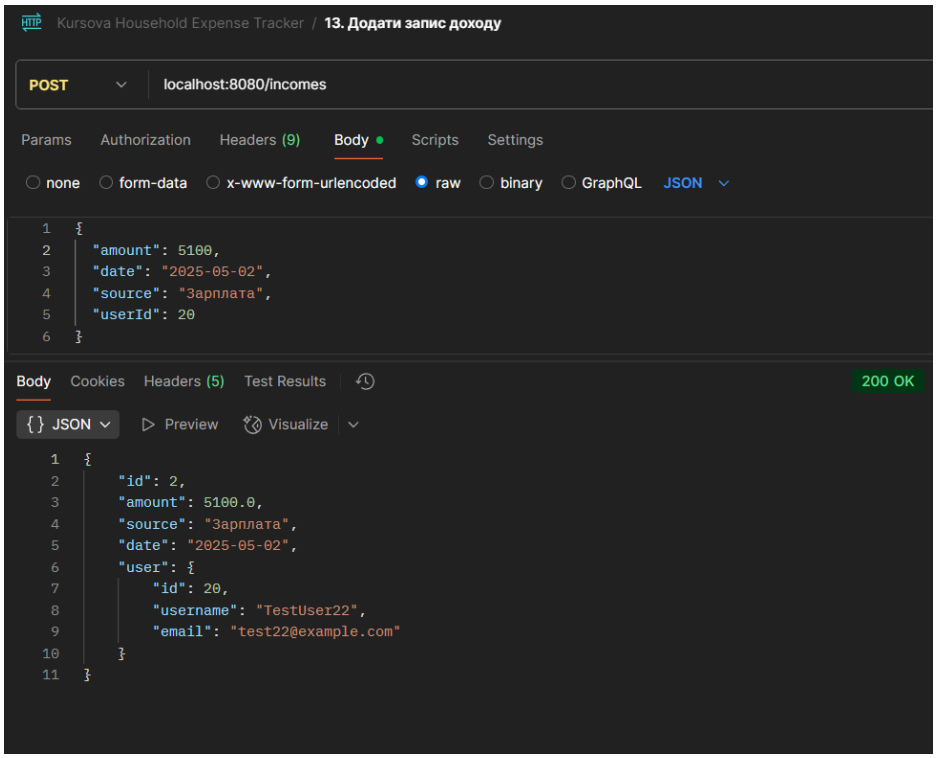

Рис 6.18 – Результат виконання запиту з пункту 13

14) Отримання доходу користувача (рис. 6.19)

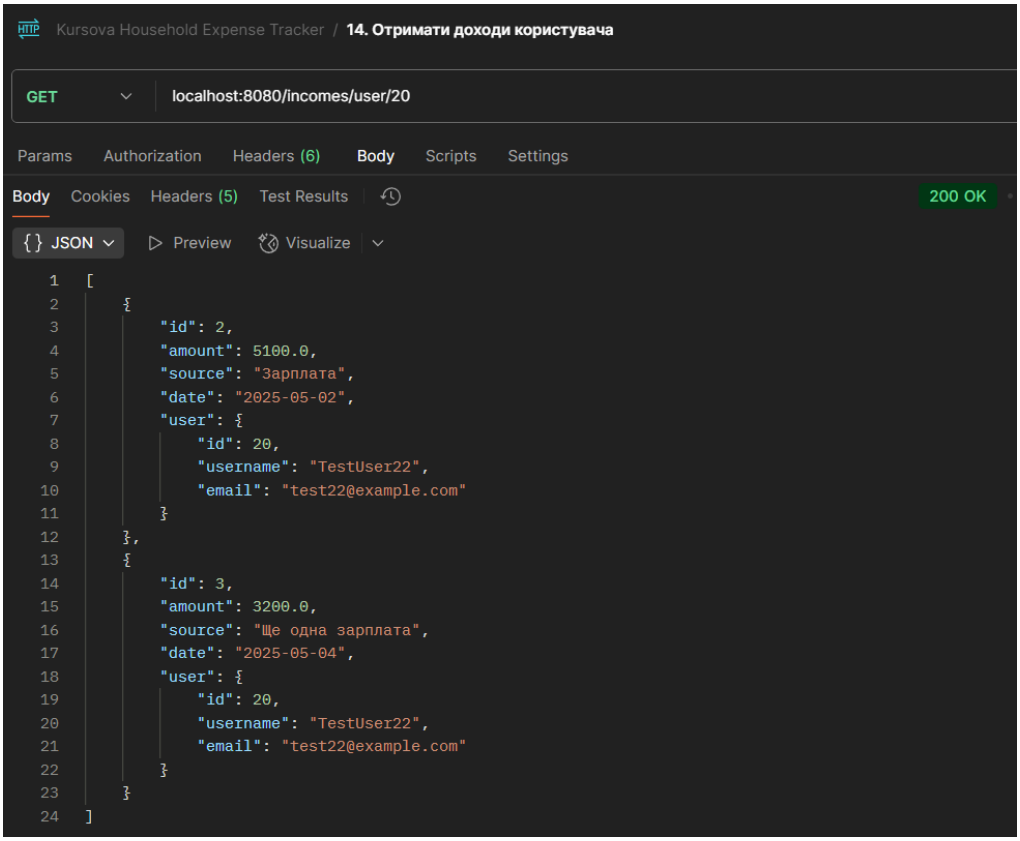

Рис 6.19 – Результат виконання запиту з пункту 14

15) Видалення доходу (рис. 6.20 – 6.21)

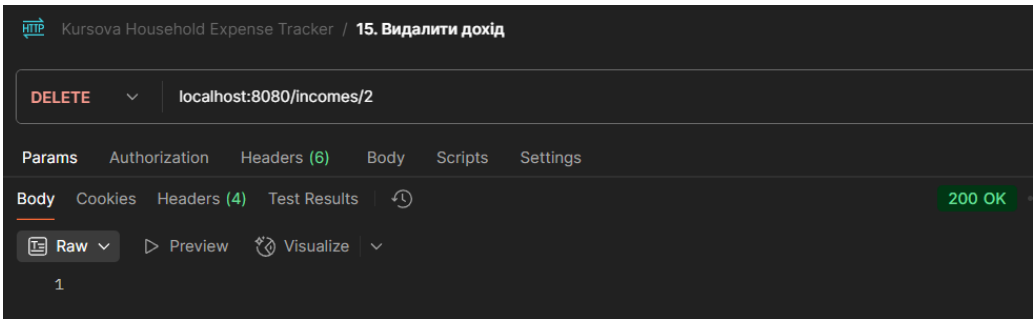

Рис 6.20 – Результат виконання запиту з пункту 15

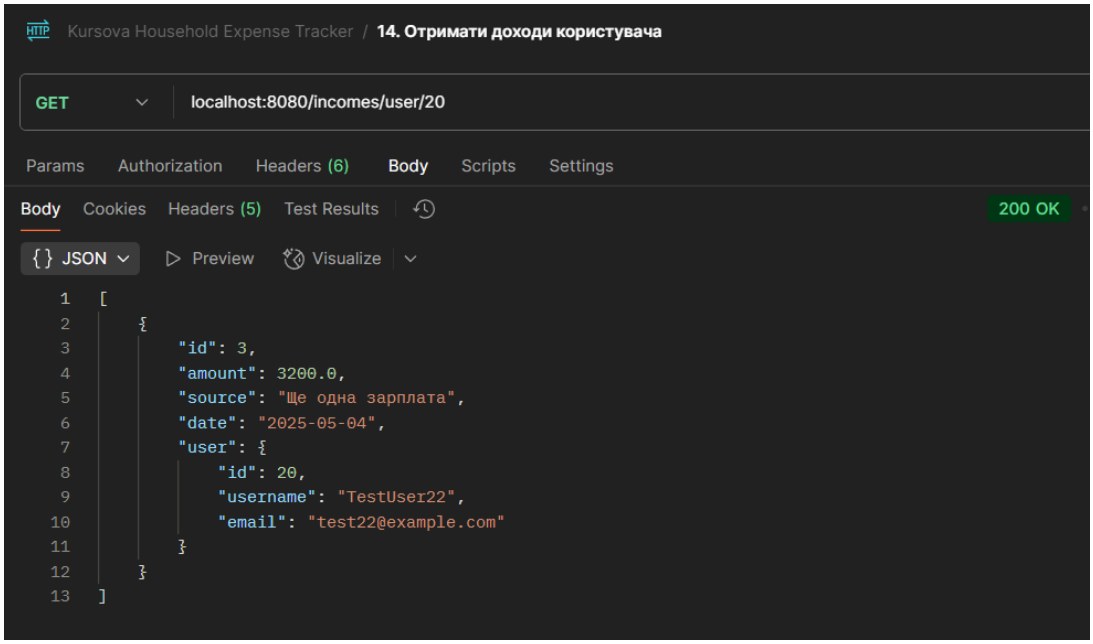

Рис 6.21 – Демонстрація результату виконання запиту з пункту 15

16) Додавання місячного бюджету (budget) (рис. 6.22)

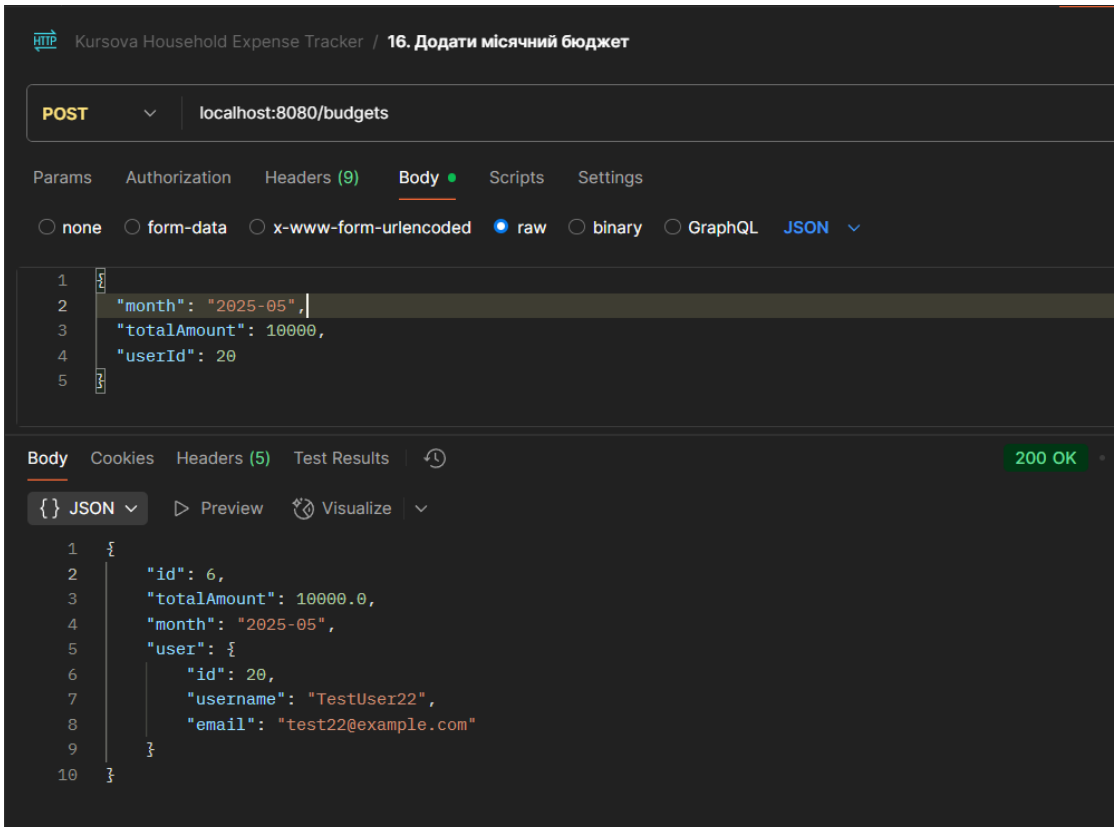

Рис 6.22 – Результат виконання запиту з пункту 16

17) Отримання бюджетів користувача (рис. 6.23)

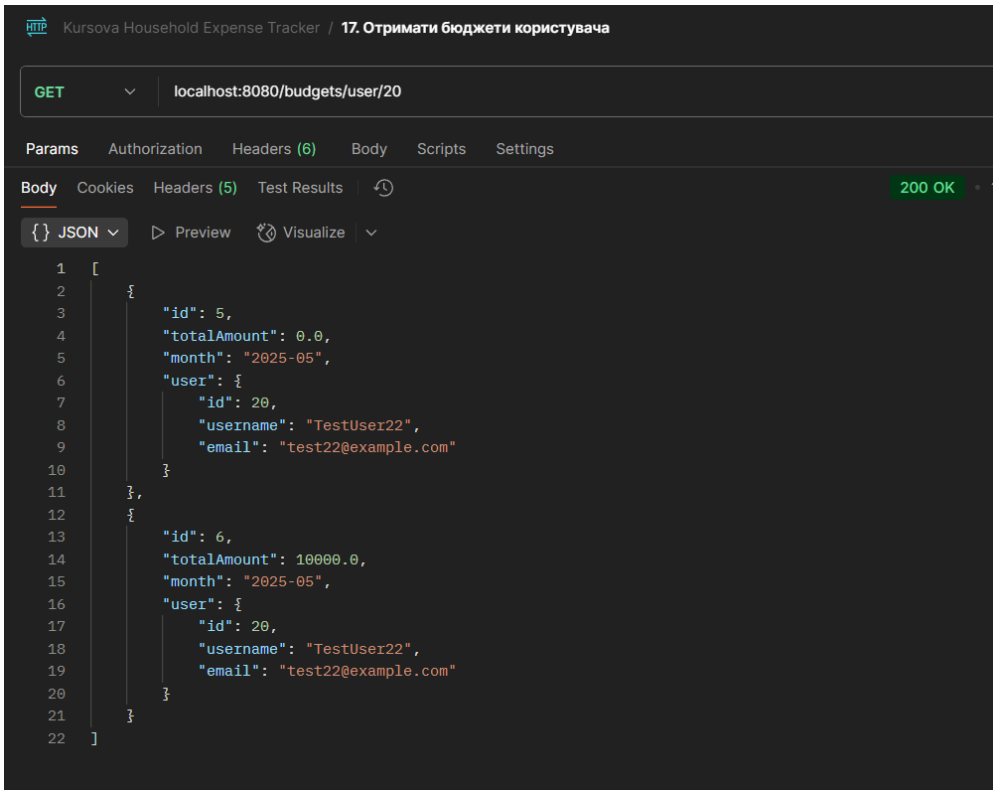

Рис 6.23 – Результат виконання запиту з пункту 17

18) Оновлення бюджету (рис. 6.24 – 6.25)

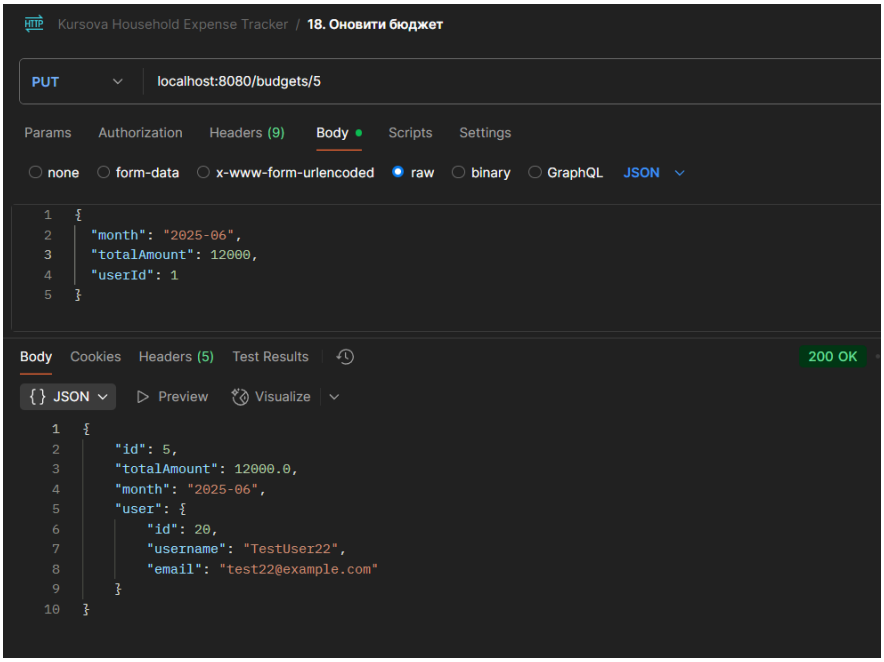

Рис 6.24 – Результат виконання запиту з пункту 18

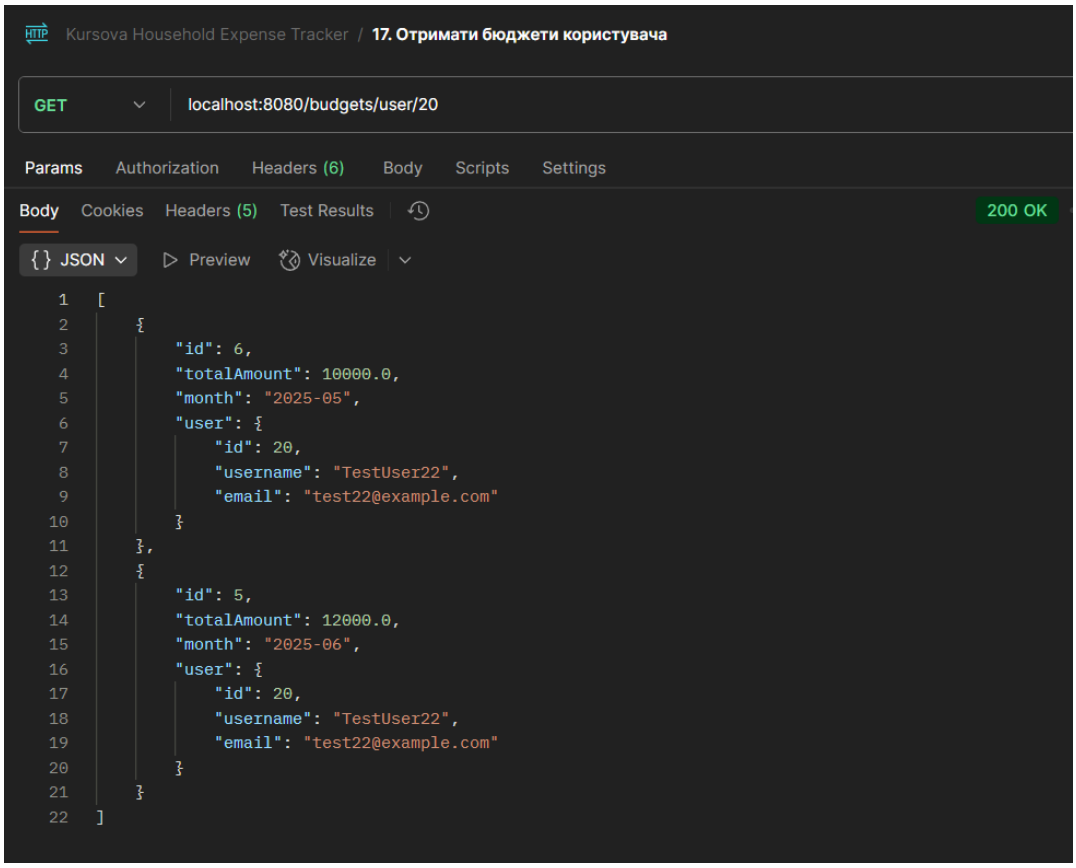

Рис 6.25 – Результат виконання запиту з пункту 18

21) Отримання суми витрат за місяць (рис. 6.26 – 6.30)

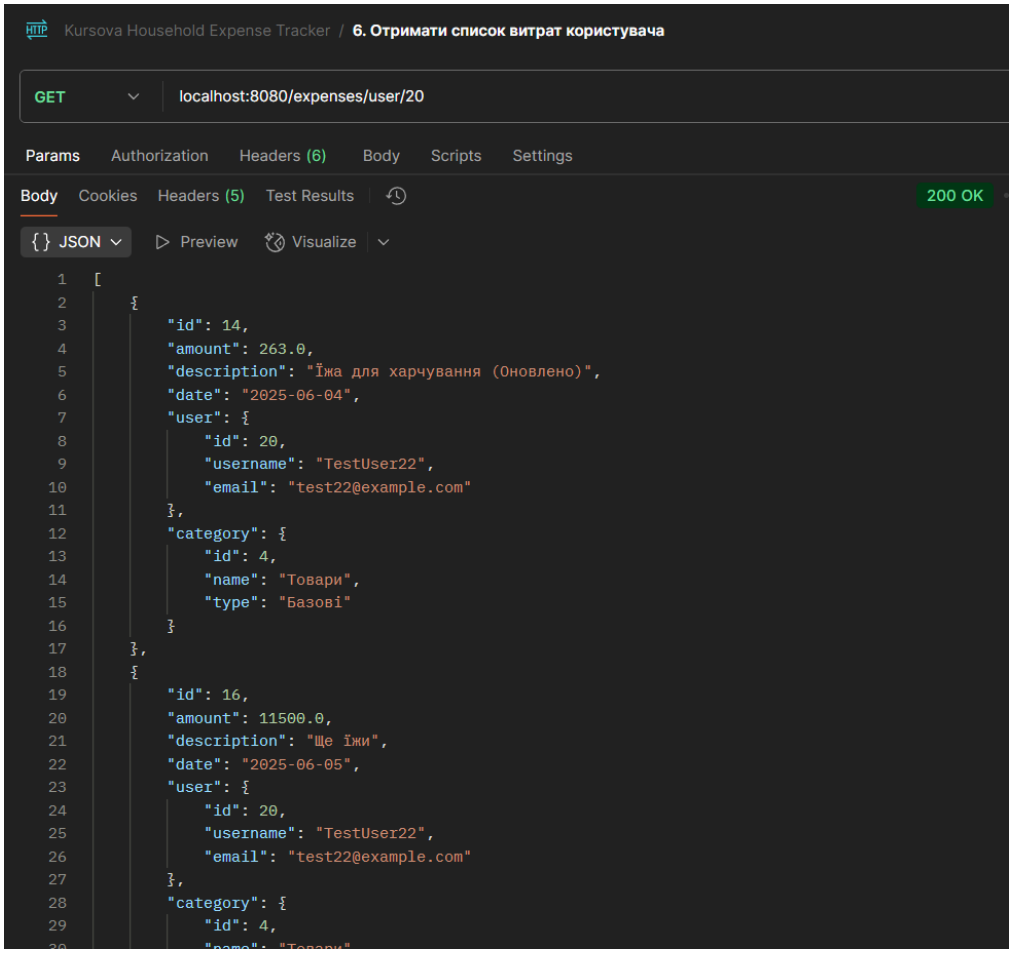

Рис 6.26 – Виведення інформ. про всі витрати, прив’язані до юзера з id=20

Рис 6.27 – Пряме продовження рис. 6.26

```
        },
        "category": {
            "id": 5,
            "name": "Кращий хавчік",
            "type": "Аби поїсти"
        }
    },
    {
        "id": 18,
        "amount": 575.4,
        "description": "Хавчік 2-ий",
        "date": "2025-05-06",
        "user": {
            "id": 20,
            "username": "TestUser22",
            "email": "test22@example.com"
        },
        "category": {
            "id": 5,
            "name": "Кращий хавчік",
            "type": "Аби поїсти"
        }
    }
]
```


Рис 6.28 – Пряме продовження рис. 6.27

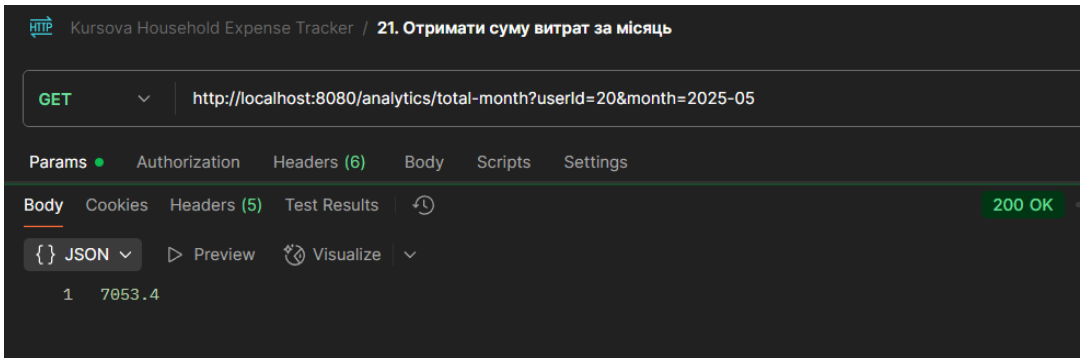

Рис 6.29 – Результат виконання запиту з пункту 21 (місяць – 2025-05)

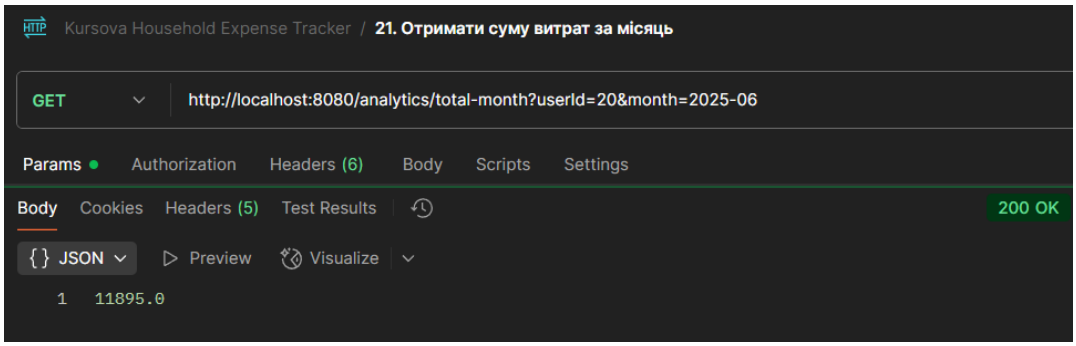

Рис 6.30 – Результат виконання запиту з пункту 21 (місяць – 2025-06)

22) Отримання середньої витрати на день (рис. 6.31 – 6.32)

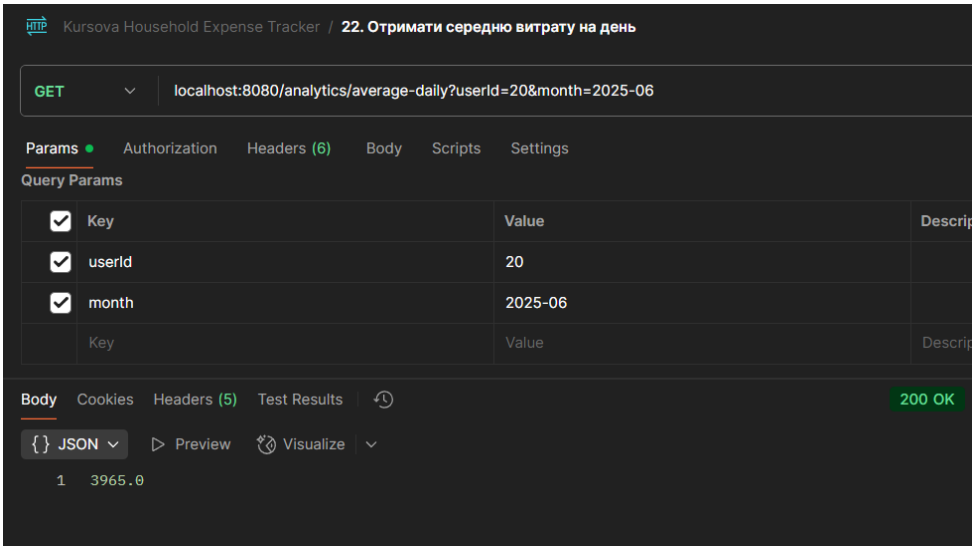

Рис 6.31 – Результат виконання запиту з пункту 22 (місяць – 2025-06)

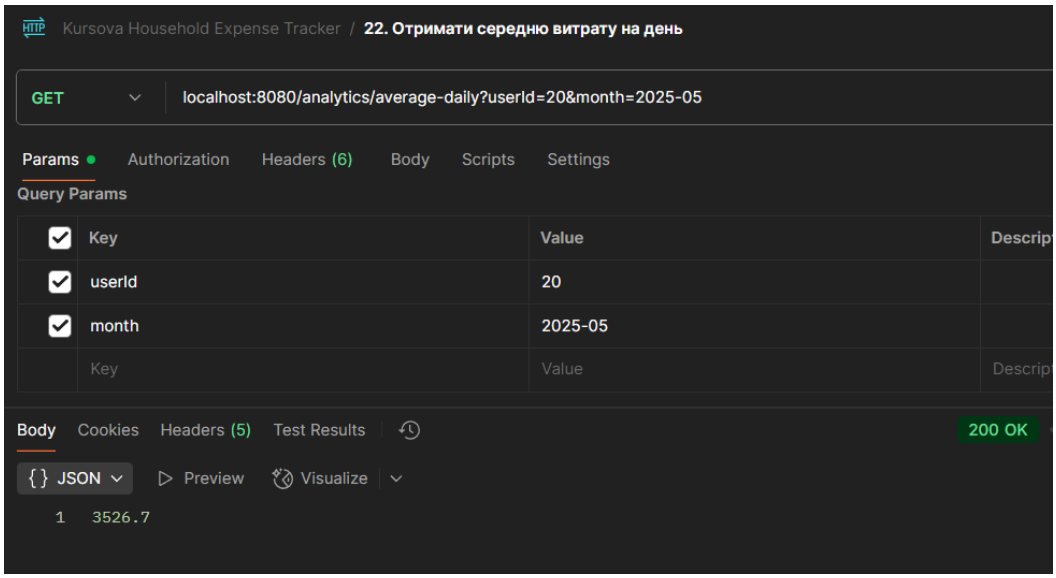

Рис 6.32 – Результат виконання запиту з пункту 22 (місяць – 2025-05)

23) Отримання найбільших витрат місяця (рис. 6.33 – 6.34)

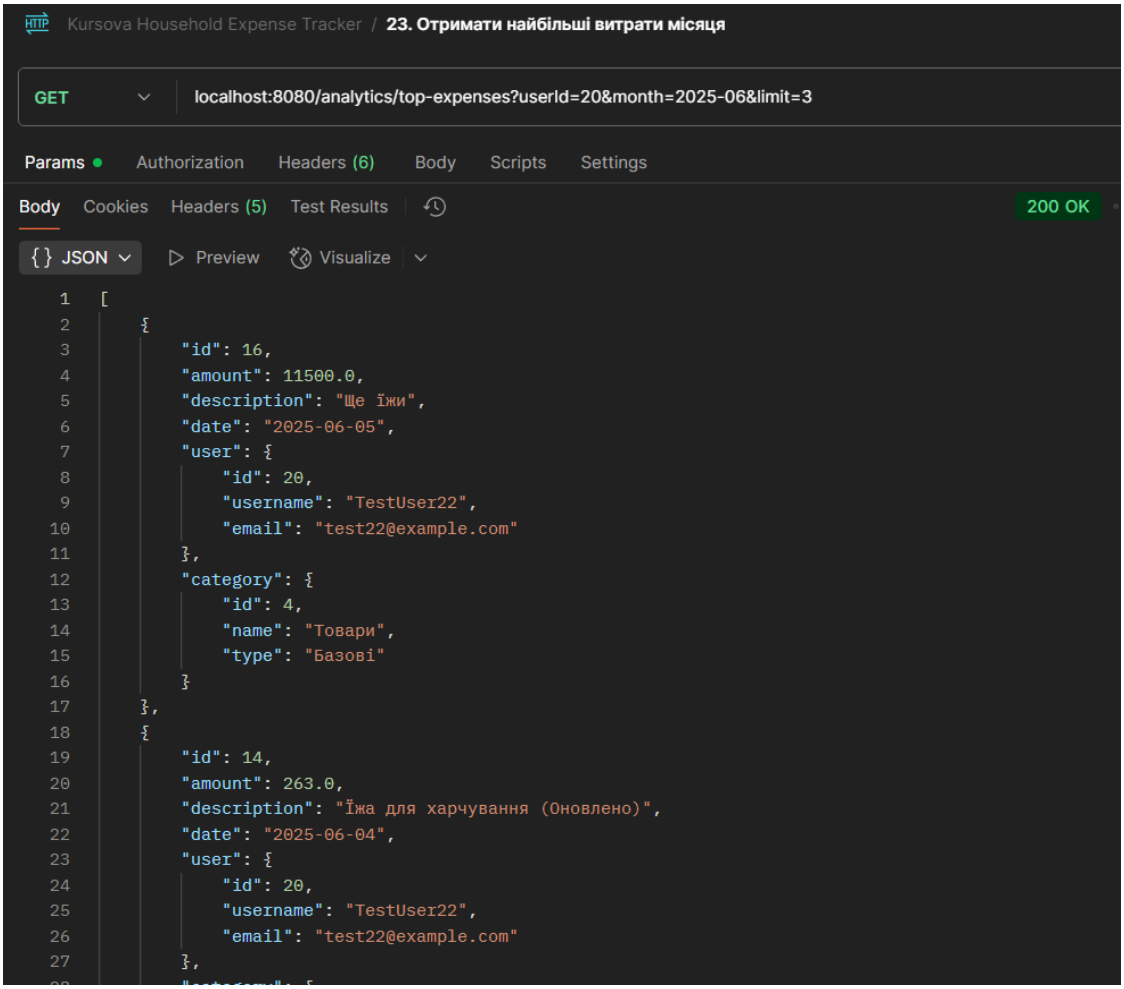

Рис 6.33 – Результат виконання запиту з пункту 23

Рис 6.34 – Пряме продовження рис. 6.33

24) Отримання залишку бюджету (рис. 6.35 – 6.36)

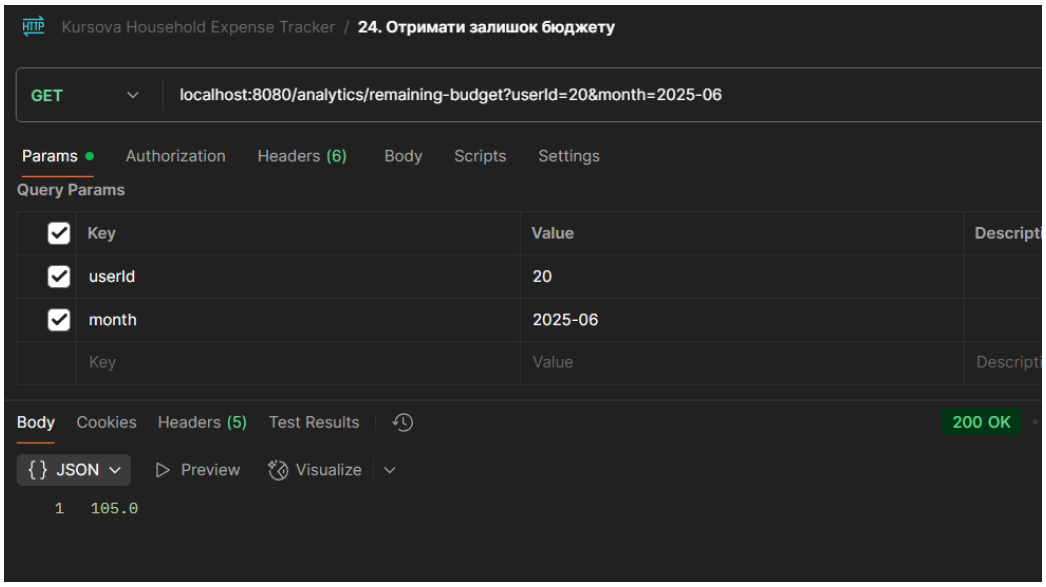

Рис 6.35 – Результат виконання запиту з пункту 24 (місяць – 2025-06)

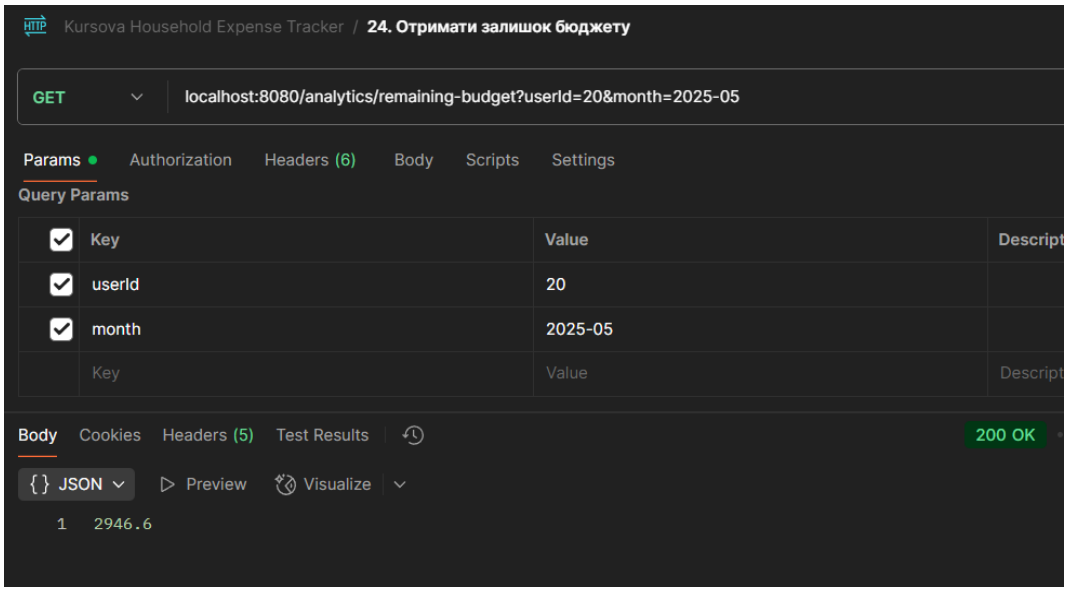

Рис 6.36 – Результат виконання запиту з пункту 24 (місяць – 2025-05)

25) Отримання витрат по даті (рис. 6.37 – 6.40)

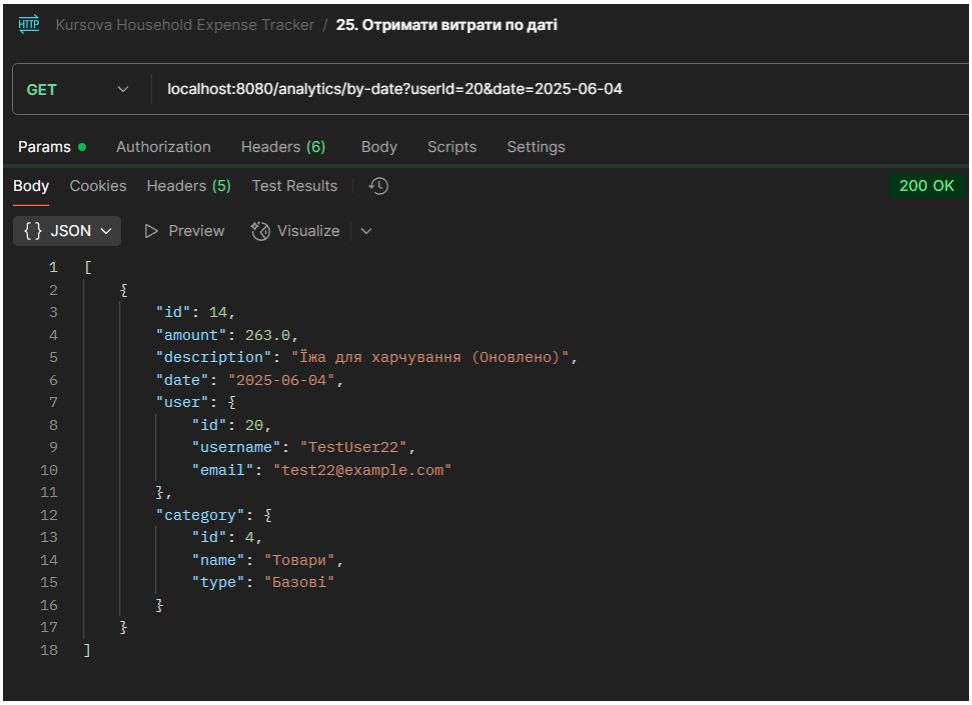

Рис 6.37 – Результат виконання запиту з пункту 25 (за датою 2025-06-04)

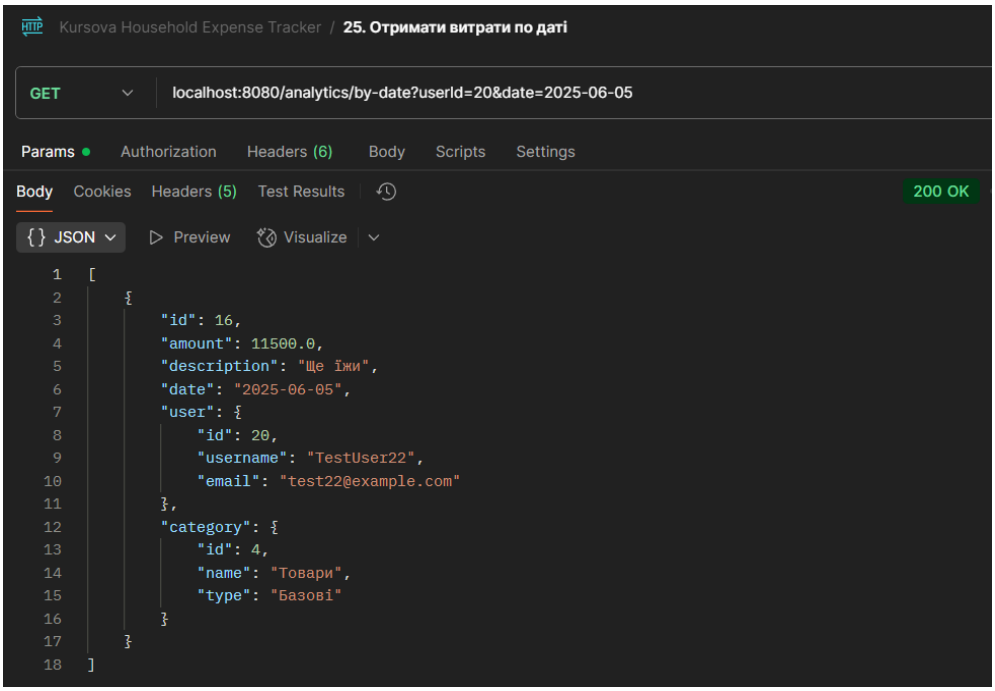

Рис 6.38 – Результат виконання запиту з пункту 25 (за датою 2025-06-05)

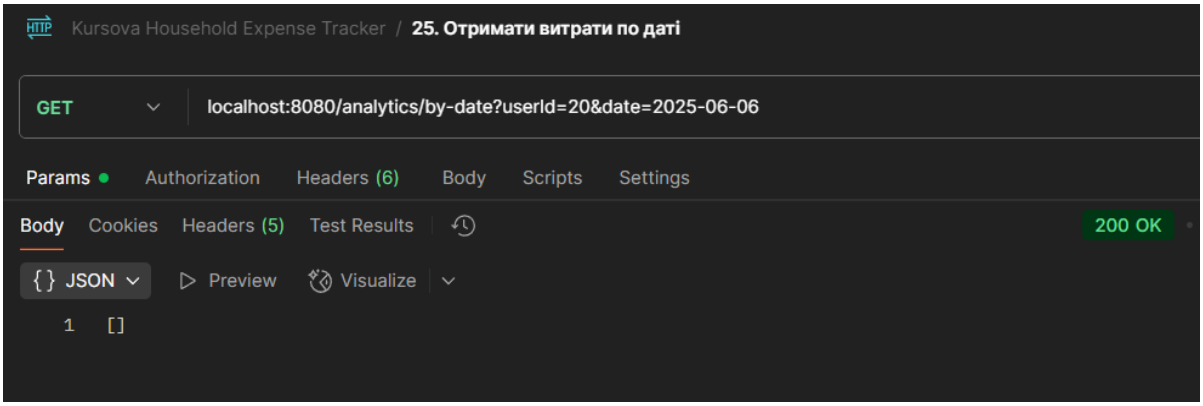

Рис 6.39 – Результат виконання запиту з пункту 25 (за датою 2025-06-06)

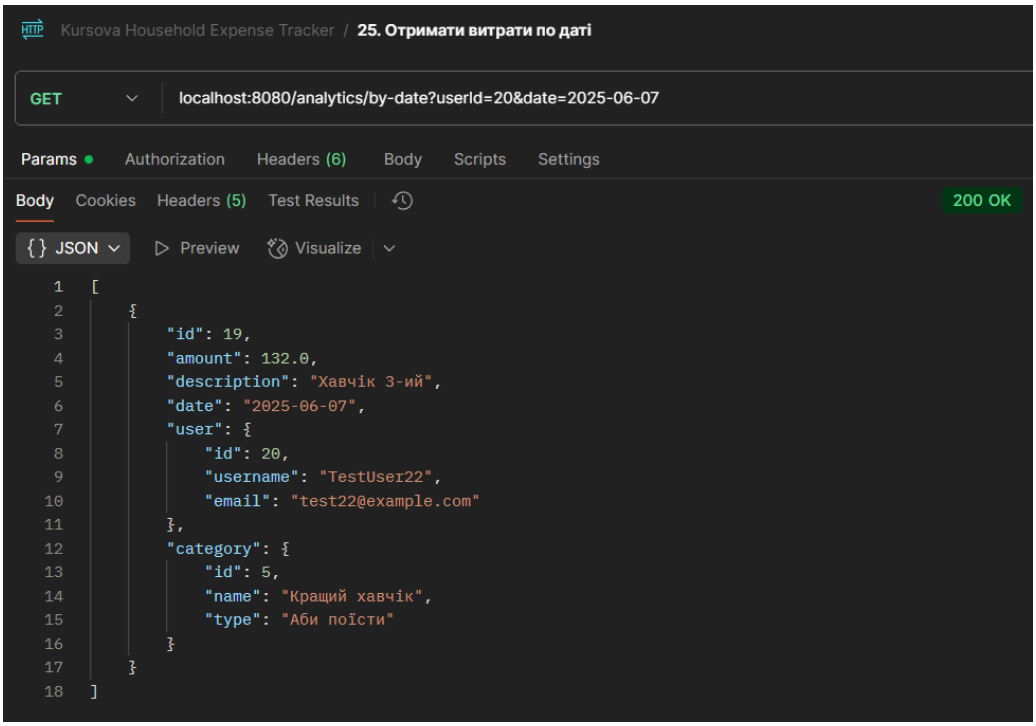

Рис 6.40 – Результат виконання запиту з пункту 25 (за датою 2025-06-07)