ІНДИВІДУАЛЬНІ ЗАВДАНЯ

для лабораторної роботи №2

з дисципліни «Оптимізація та підвищення програмного продукту»

кафедра математичного забезпечення ЕОМ, ДНУ

202_/20_ н.р.

1. ПОСТАНОВКА ЗАДАЧІ

1. Використовуючи скрипт UnoptimizedDB.sql створити базу даних.
2. Заповнити базу даних випадковими даними.
3. Створити web api додаток (можна обрати будь-яку технологію, для .net зрчуно використовуваати dotnet new webapi), що підтримує CRUD операції
4. Забезпечити можливість пошуку користувачів за частиною email (SELECT FROM Users WHERE Email LIKE '%example.com';). Запит має підтримувати пагінацію.
5. Забезпечити отримання суми доходу за певний період (SELECT SUM(TotalAmount) FROM Orders WHERE OrderDate >= '2024-01-01';)
6. Забезпечити можливість отримати інформацію про замовлення користувача (з таблиці Orders) за його електроною поштою
7. Проаналізувати коментарі у скрипті. Виправити недоліки у створеній базі даних. Вказати, які саме коментарі були враховані, які не враховані і чому. Як при цьому змінився час на обробку запитів?
8. Веб додаток має підтримувати серверне кешування. Обрати одну із розглягнутих стратегій. Пояснити, чому саме вона була обрана.
9. Cache Aside (Lazy Loading)
10. Read Through Cache
11. Write Around
12. Write Back
13. Write Through

2. ОПИС РОЗВ’ЯЗКУ

Перш за все я відредагував базу даних, як було вказано у завданні. Для цього оптимізував деякі змінні для деяких умов. Наприклад грошові зміні тепер мають тип double, а тип картинки профілю на масив байтів.

Щоб створити базу даних з цього скрипту треба його скопіювати в командний рядок бази даних. Я обрав PostreSQL, а для роботи з ним PGAdmin. Там, щоб привести в дію скрипт потрібно використати Query tool (рис 1), та туди вставити скрипт. Після натиснути на стрілку Execute script (рис 2) й після цього створяться таблиці, та в даному випадку з даними, бо в скрипті є рядки зі створення таких (рис 3).

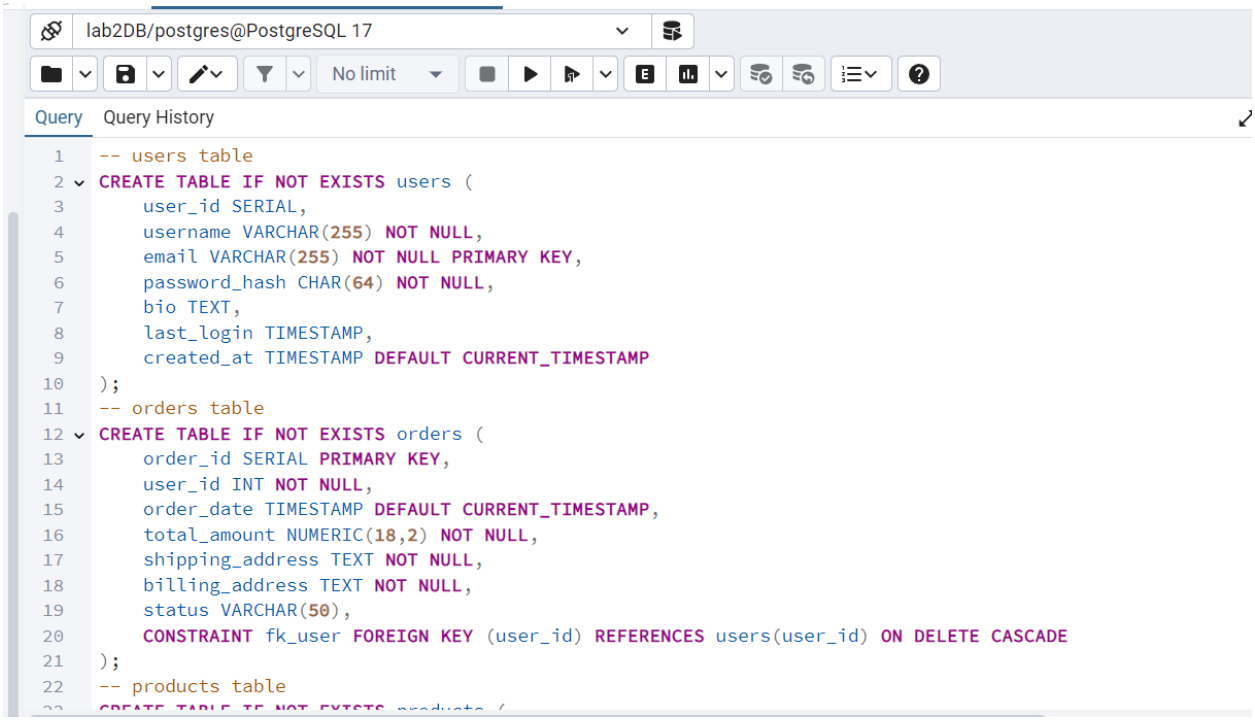

Рисунок 1 – вид полів вводу Query tool.

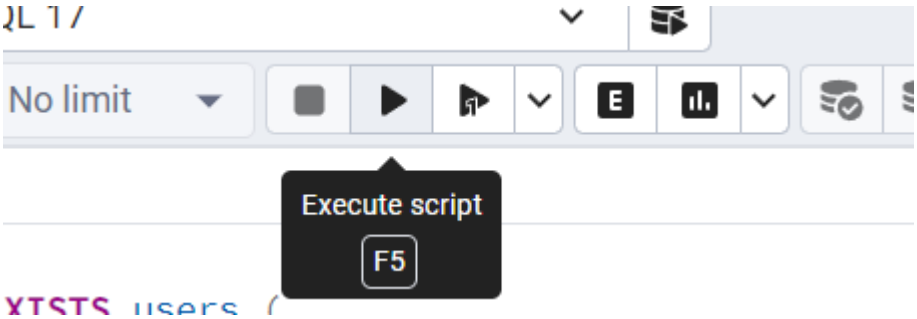

Рисунок 2 – Стрілка для активації скрипту.

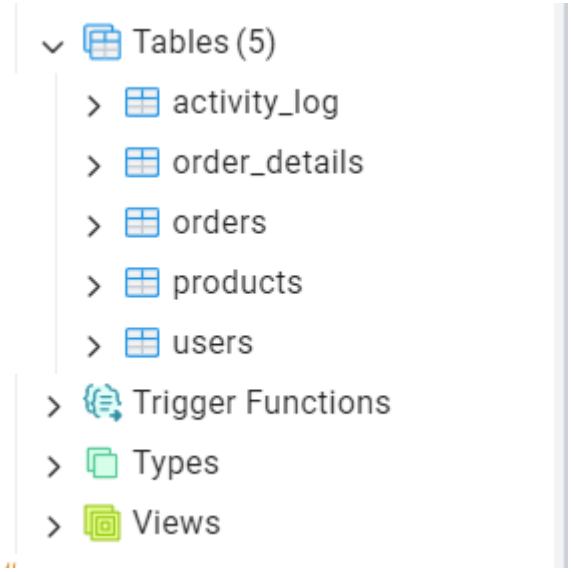

Рисунок 3 – Ново-створені таблиці.

Для роботи з базою даних я обрав Java зі Spring boot. Були розроблені репозиторії, класи сутностей, класи сервлетів та контролерів, до яких було додані CRUD операції (рис 4).

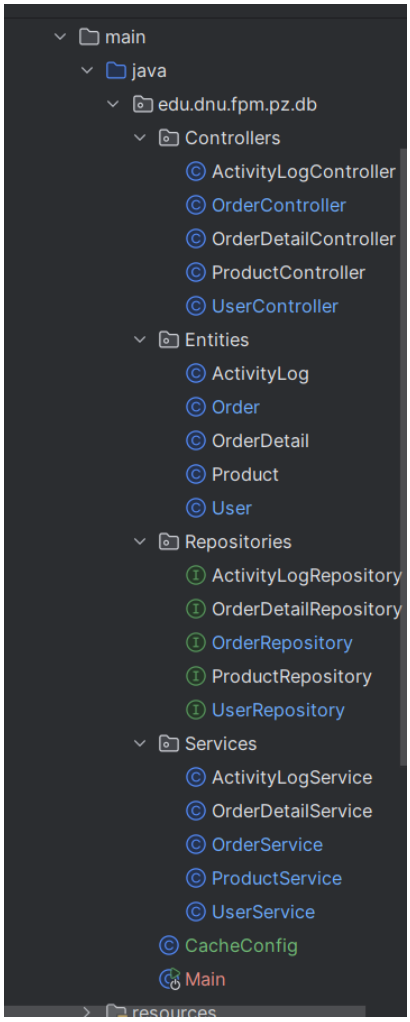

Рисунок 4 – Архітектура серверу.

Для заповнення таблиць бази даних випадковими значеннями я створив рест запит контролера “/generate?count={int}”, який за допомогою бібліотеки javafaker генерую задану кількість об’єктів з унікальними випадковими значеннями (лістинг 5, 6).

Лістинг 5 – Обробка контролеру рест запиту “/generate?count={int}” для users.

```
@PostMapping("/generate")
public ResponseEntity<String> generateUsers(@RequestParam int count) {
   userService.generateUsers(count);
   return ResponseEntity.ok("Users generated: " + count);
}
```

Лістинг 6 – Сервлет рест запиту “/generate?count={int}” для users.

```
public void generateUsers(int count) {
   for (int i = 0; i < count; i++) {
      User user = new User();
      user.setUsername(faker.name().username());
      user.setEmail(faker.internet().emailAddress());
      user.setPasswordHash(faker.internet().password());
      user.setBio(faker.lorem().sentence());
      userRepository.save(user);
   }
}
```

Приклад використання рест запитів на генерацію об’єктів (рис 7, 8).

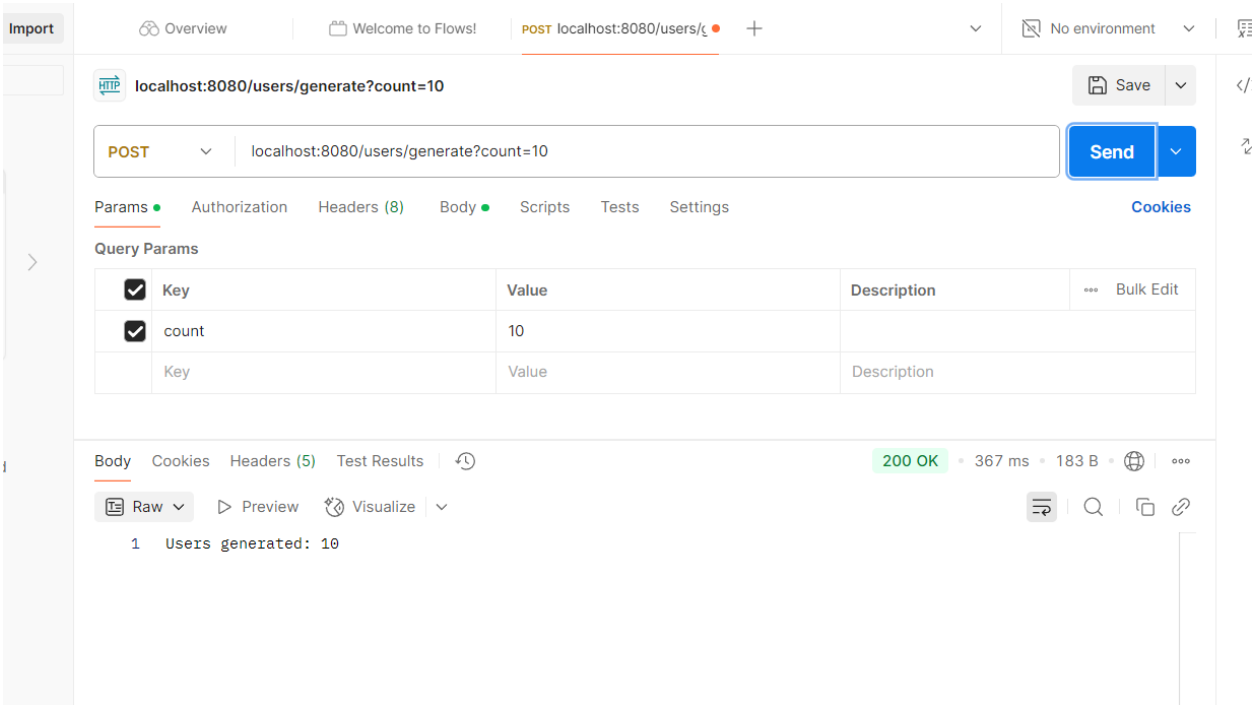

Рисунок 7 – Рест запит на генерацію 10 об'єктів користувачів.

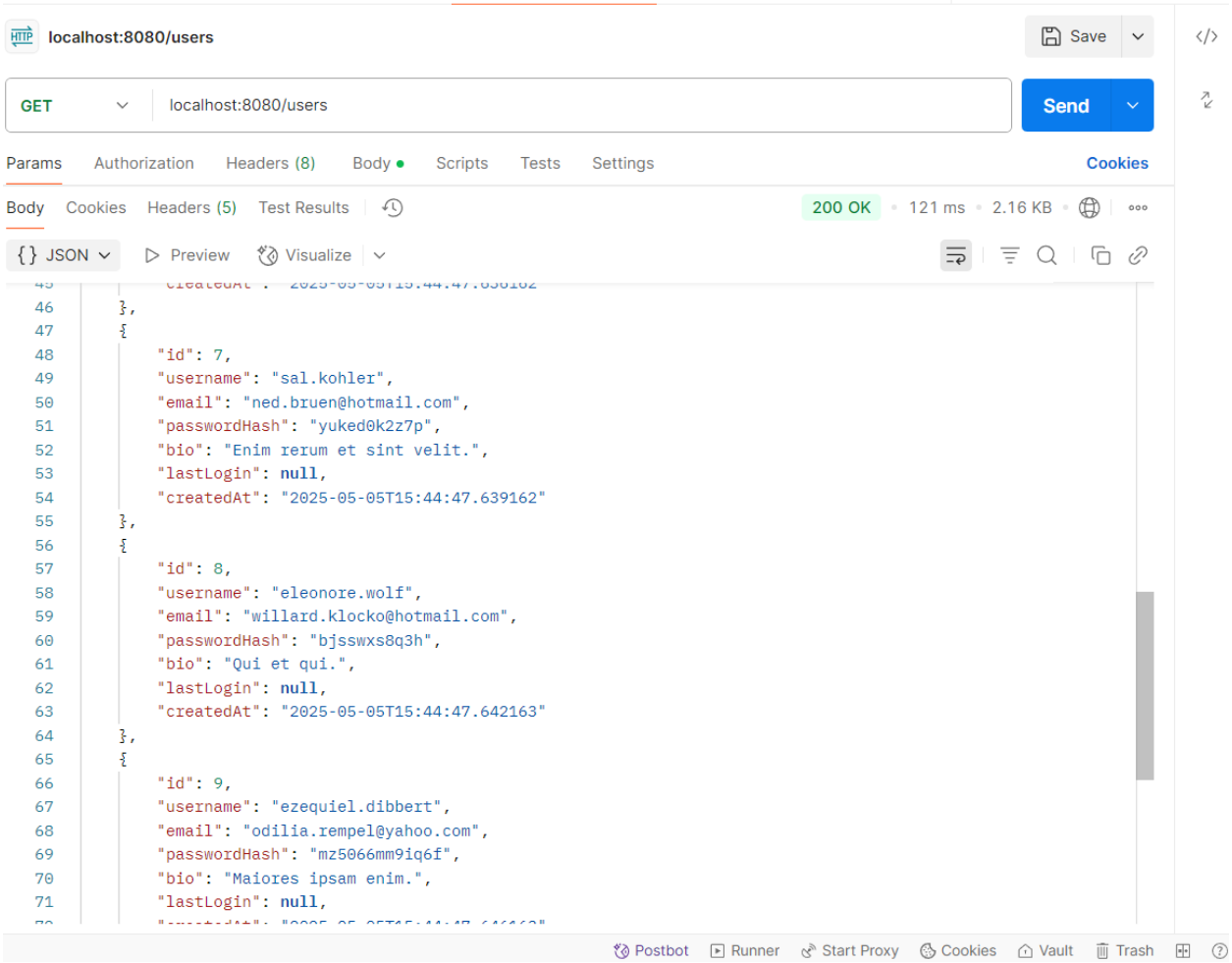

Рисунок 8 – Згенеровані 10 сутностей.

Для пошуку користувачів за частиною email, отримання суми доходу за певний період, отримання інформацію про замовлення користувача були додання запити Query до таблиці, аналогічні за синтаксисом до того, які було застосовано для генерації таблиць (лістинг 9, 10).

Лістинг 9 – Query запит для знаходження user за email.

```
@Query("SELECT u FROM User u WHERE u.email LIKE %:email%")
List<User> findByEmailContaining(@Param("email") String email, Pageable pageable);
```

Лістинг 10 – Query запити для отримання загальної суми за усі замовлення та для отримання усіх замовлень від конкретного користувача.

```
@Query("SELECT SUM(o.totalAmount) FROM Order o WHERE o.orderDate >= :startDate")
BigDecimal getTotalRevenueSince(@Param("startDate") LocalDate startDate);
@Query("SELECT o FROM Order o WHERE o.user.email = :email")
List<Order> findByUserEmail(@Param("email") String email);
```

Далі приведені приклади виклику цих Query запитів(рис 11, 12, 13).

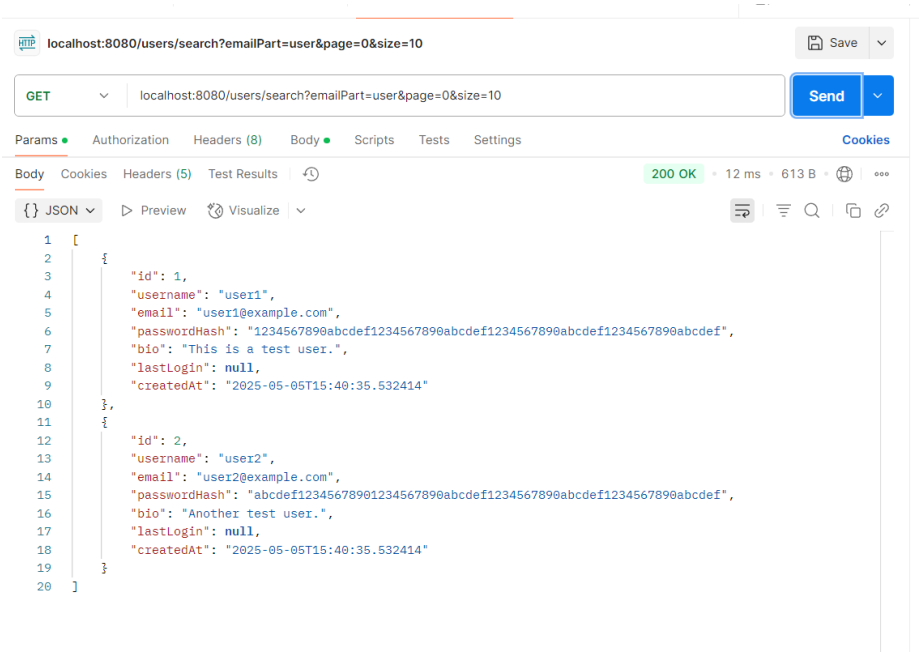

Рисунок 11 – Приклад виклику запиту на отримання користувачів за частиною email.

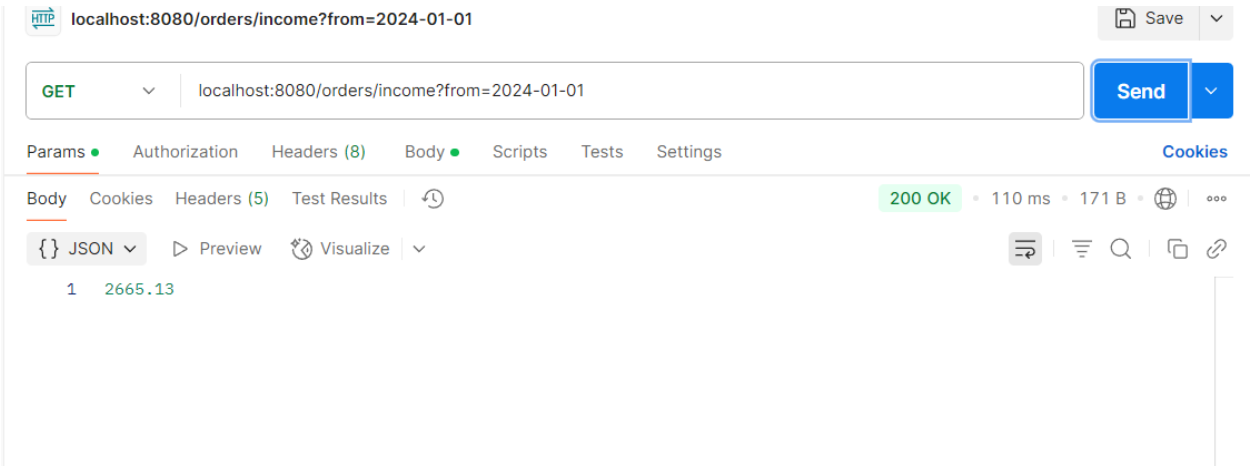

Рисунок 12 – Приклад виклику запиту на суми грошей за усі замовлення.

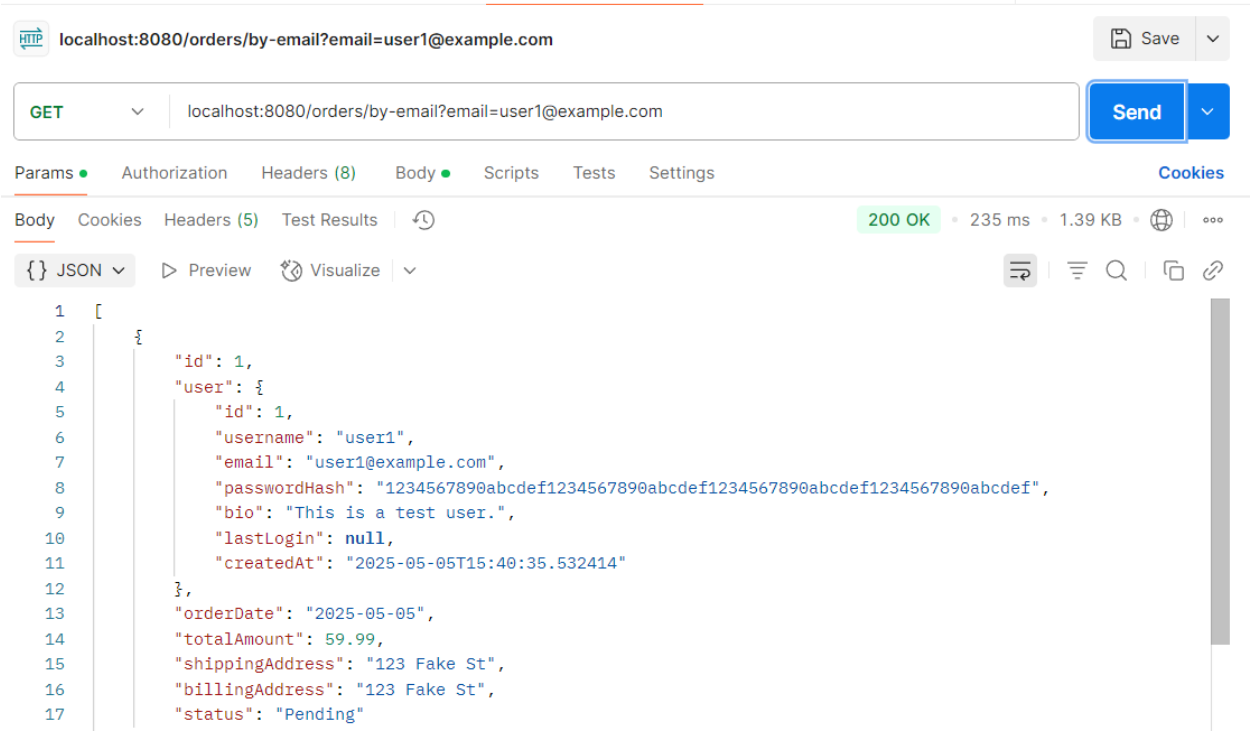

Рисунок 13 – Приклад виклику запиту на отримання запитів за email користувача.

Також запити підтримують пагінацію, так що там можна визначити відступ чи початок виведення сутностей за їх айді та також розмір сторінку виведення, за замовченням це 10 об’єктів, а відступ - 0.

За умовою ще потрібно було створити кешування даних. Я обрав ліниве кешування, де кеш утворюються після першого звертання до якихось даних, а далі при наступному звертанні до тих самих даних, буде використовуватись кешоване значення.

Я вирішив зробити хешування для даних користувачів, до яких звертаються за айді (лістинг 14). Для того щоб побачити різницю між використанням хешу та без нього був створений ще один сервлет (лістинг 15).

Лістинг 14 – сервлет звертання до користувача за айді з використанням лінивого кешування.

```
public User getUserById(Long id) {
    Cache.ValueWrapper wrapper = getUserCache().get(id);
    if (wrapper != null) {
        return (User) wrapper.get();
    }

    Optional<User> user = userRepository.findById(id);
    user.ifPresent(u -> getUserCache().put(id, u));
    return user.orElse(null);
}
```

Лістинг 15 – сервлет звертання до користувача за айді без використанням кешування.

```
public Optional<User> getUserByIdWithoutCache(Long id) {
    User user = userRepository.findById(id).get();
    return Optional.ofNullable(user);
}
```

Для перевірки ефективності кешування, я створив 10000 сутностей й викликав ті два сервлета (рис 16, 17).

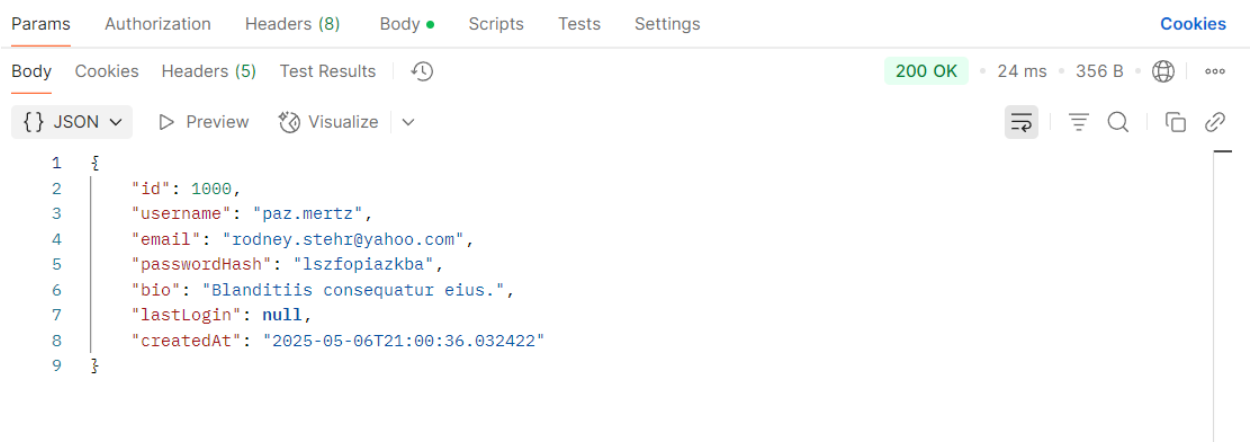

Рисунок 16 – Запит до сутності без кешування.

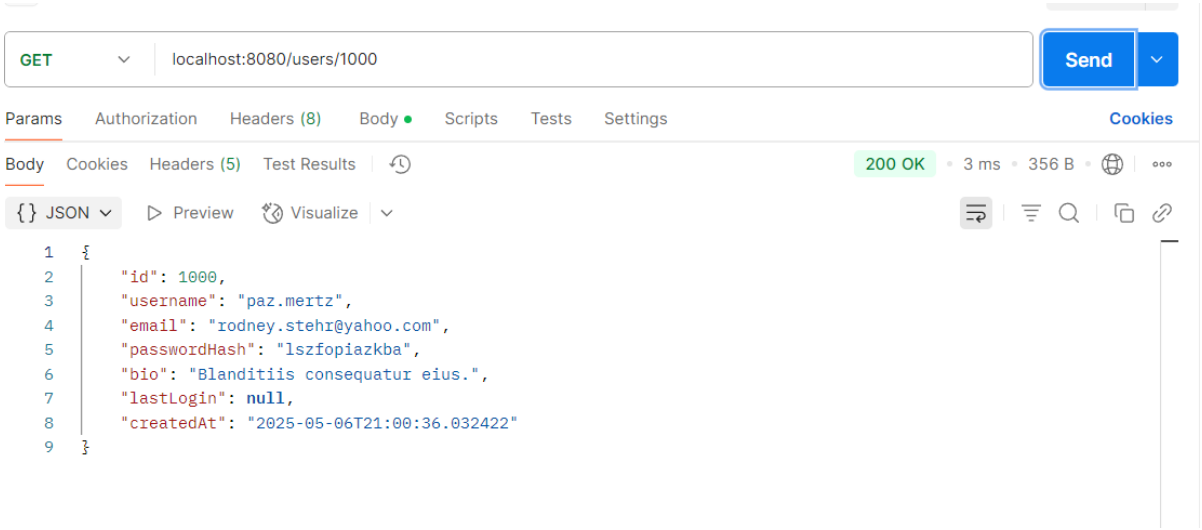

Рисунок 17 – Запит до сутності з кешуванням.