**1. Створи набір з GET, POST, PUT, PATCH, DELETE запитів до JSONPlaceholder, які надсилаються протягом 10 секунд у 3 ітерації.**

**Дата:** 2025-09-20

**Test Plan:** JSONPlaceholder.jmx

**Thread Group:** 20 user, 3 ітерації, ~10 секунд

**Тестовані endpoints**

- GET /posts/1

- POST /posts

- PUT /posts/1

- PATCH /posts/1

- DELETE /posts/1

**Результати (Summary Report)**

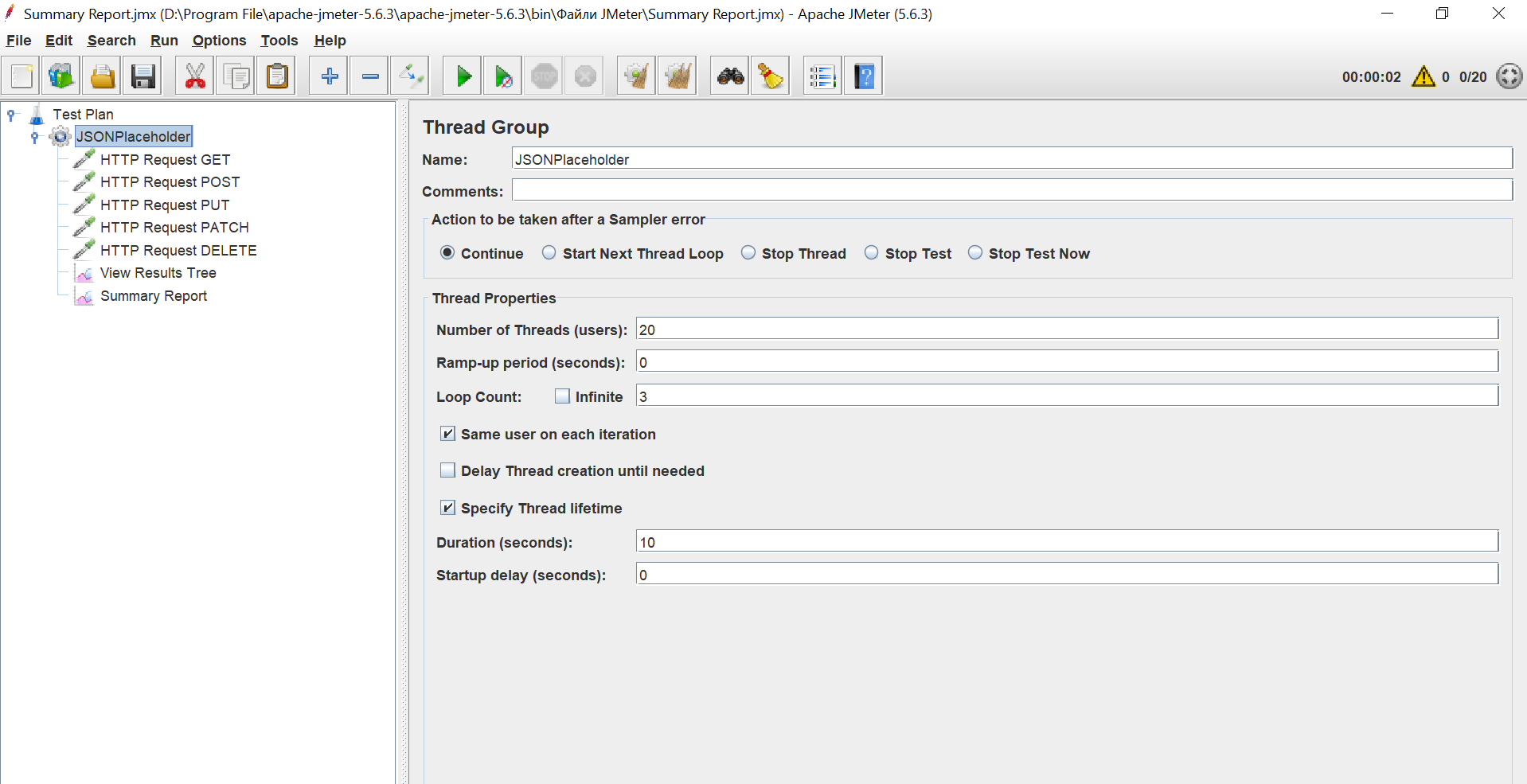
- Всі запити успішні (Response Code 200/201)

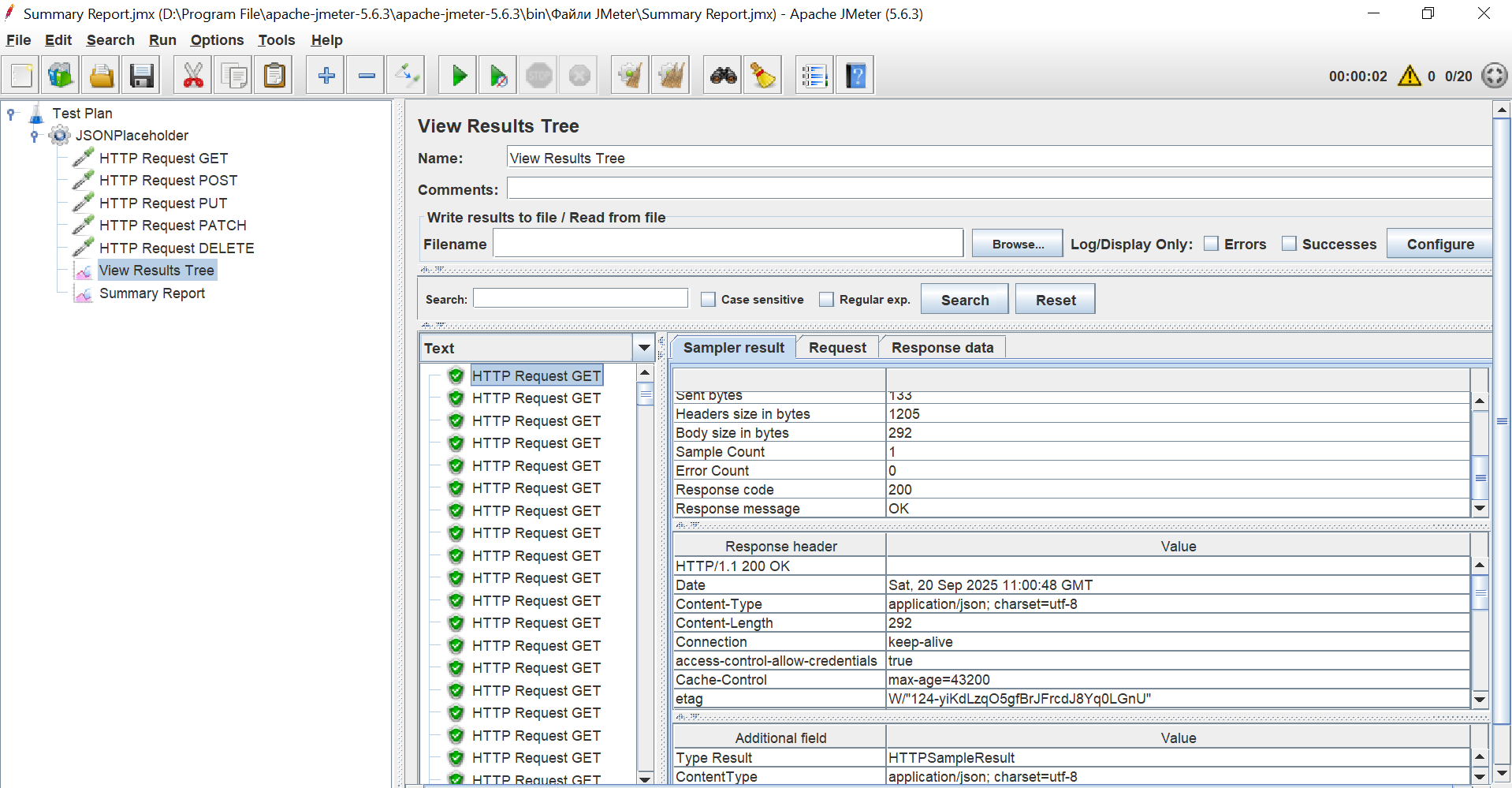
- Середній час відповіді: 159 ms

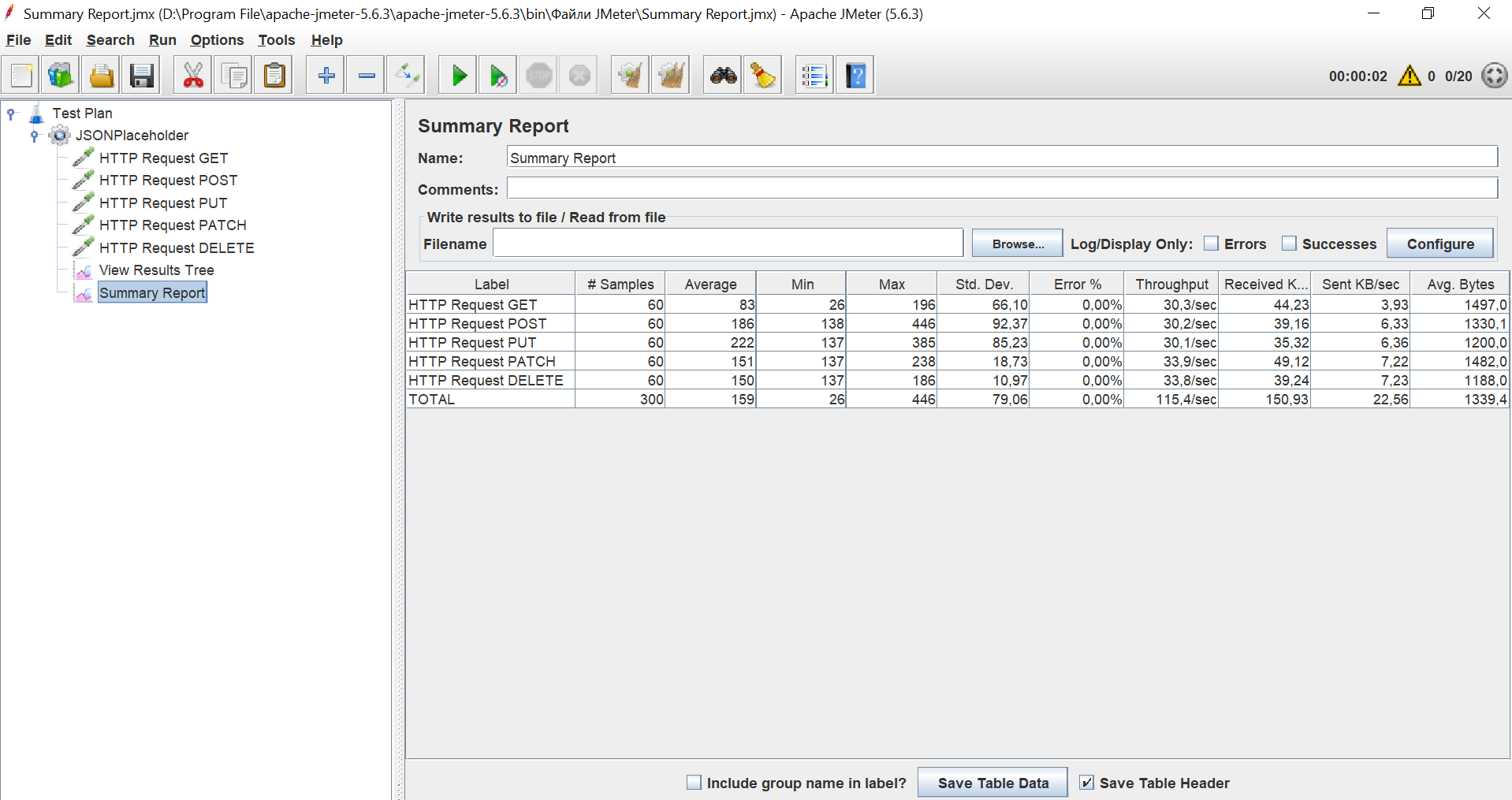
- Error %: 0%

**Висновок**

- Тест виконано успішно, всі запити пройшли без помилок.







**До кожного з запитів застосуй 3 різних assert’и.**

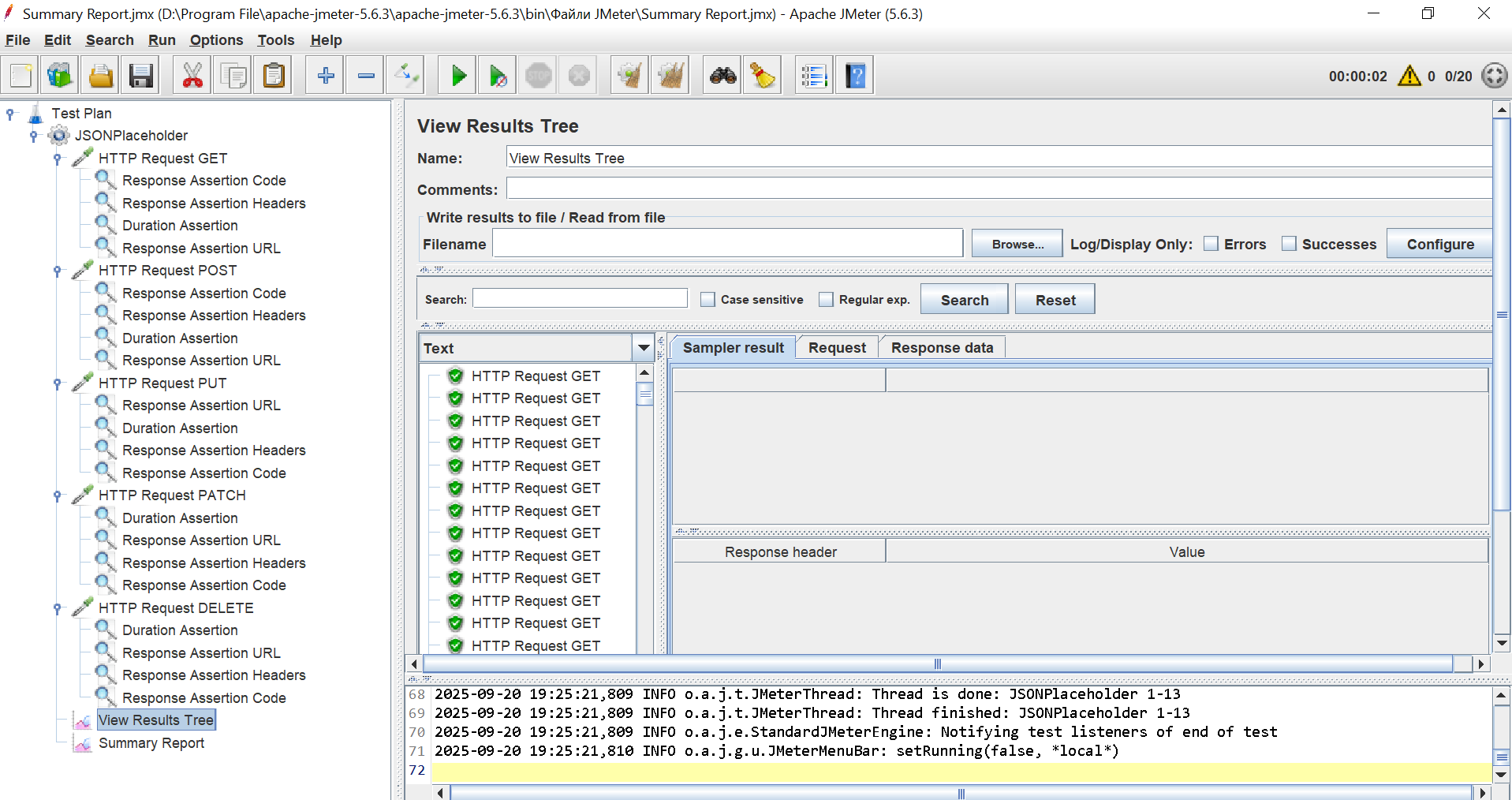
**Assertions**

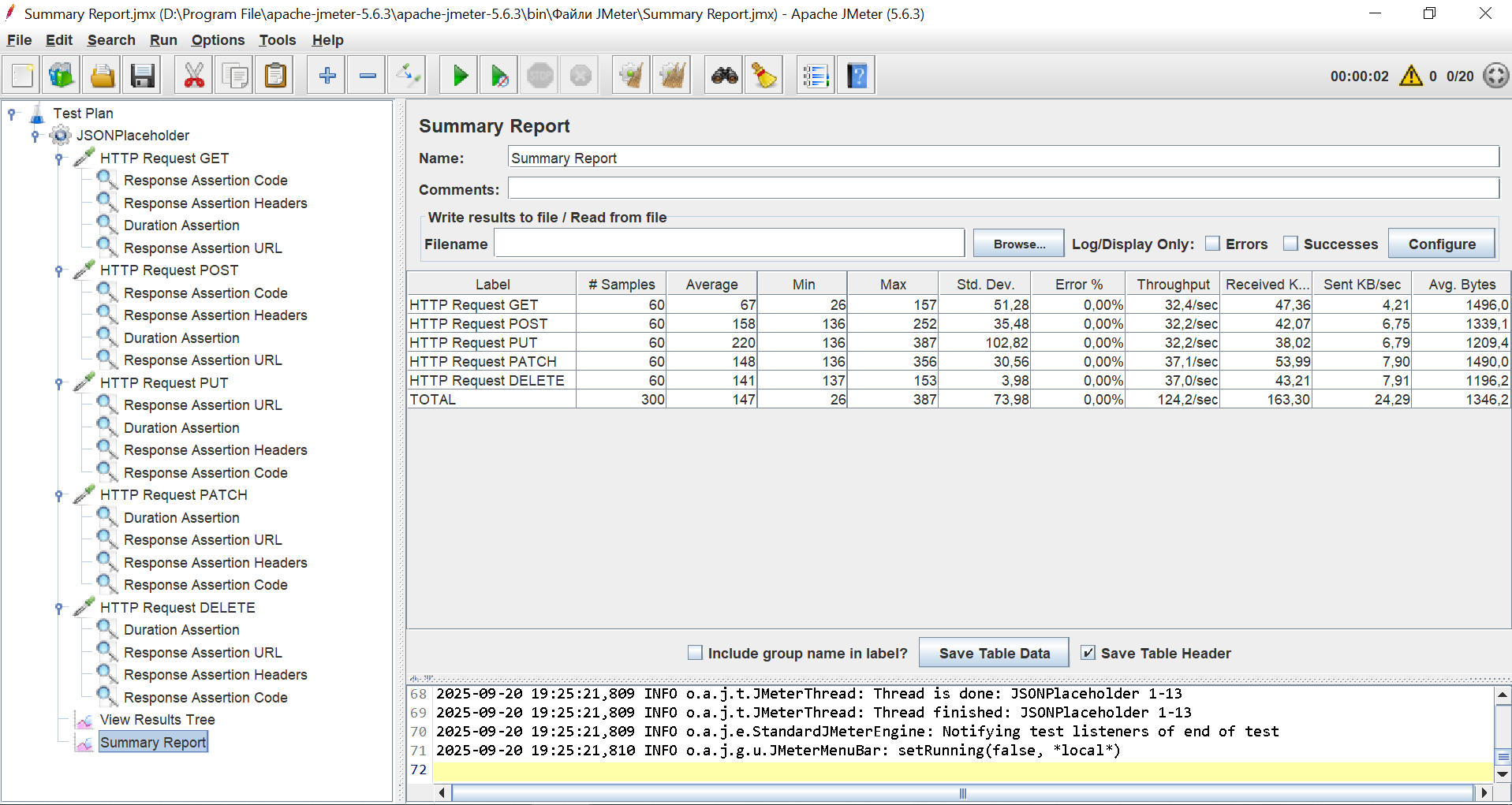
*1. Response Code – 200 / 201*

*2. Response Header - json*

*3. Duration Assertion – 5000mlsc*

*4. Response URL - /posts/1 and /posts*





**Опиши висновки щодо результатів тестування (базуючись на репортерах) в окремому файлі.**

**1. Опис тесту:**

- Виконано HTTP запити: GET /posts/1, POST /posts, PUT /posts/1, PATCH /posts/1, DELETE /posts/1

- Кожен запит тестувався з чотирма Assertions:

1. Response Code – 200 / 201

2. Response Header - json

3. Duration Assertion – 5000mlsc

4. Response URL - /posts/1 and /posts

**2. Використані Listener:**

- View Results Tree – для детальної перевірки кожного запиту

- Summary Report – для зведених показників: час відповіді, кількість успішних/провалених запитів

**3. Результати:**

- Всі HTTP запити пройшли успішно (Response Code відповідає очікуваному)

- Assertions у всіх запитах пройшли, Status – зелений

- Середній час відповіді (Total) для всіх запитів 147 ms

- Жодних помилок у View Results Tree не виявлено

**4. Висновки:**

- Сервер JSONPlaceholder стабільно обробляє запити всіх методів (GET, POST, PUT, PATCH, DELETE)

- Response Assertions та Duration Assertions підтверджують коректність і швидкість обробки запитів

<https://github.com/AndreyMaslo4/GitLesson/tree/main/%D0%A3%D1%80%D0%BE%D0%BA%20%D0%BD%D0%B0%D0%B2%D0%B0%D0%BD%D1%82%D0%B0%D0%B6%D0%B5%D0%BD%D0%BD%D1%8F>

**Створи новий тест-план на основі тест-плану з попереднього рівня.**

**В ньому:**

* **для кожного із запитів зроби стрес-тест використаного API;**
* **потрібно виявити такі мінімальні комбінації параметрів, за яких вебсервіс перестає витримувати навантаження.**

**У Thread Group налаштовано:**

*Кількість користувачів (Number of Threads): від 50 до 500*

*Ramp-Up Period: 10 секунд*

*Loop Count: 1 (один цикл)*

*Duration: 60 секунд*

*До кожного запиту додано 4 асерти:*

*1. Response Code – 200 / 201*

*2. Response Header - json*

*3. Duration Assertion – 5000mlsc*

*4. Response URL - /posts/1 and /posts*

***Використані лістенери:***

*View Results Tree – для перегляду детальних відповідей сервера та відладки.*

*Summary Report – для збору агрегованої статистики (кількість запитів, середній час, відсоток помилок, Throughput).*

***Результати***

*Усі запити (GET, POST, PUT, PATCH, DELETE) виконались успішно при 50 віртуальних користувачах.*

*Середній час відповіді: ~130–600 мс.*

*Максимальний час відповіді: ~1098 мс.*

*Всі асерти спрацювали коректно, Response Code = 200 / 201, Duration < 5000 мс.*

*Відсоток помилок: 0%.*

***Висновки***

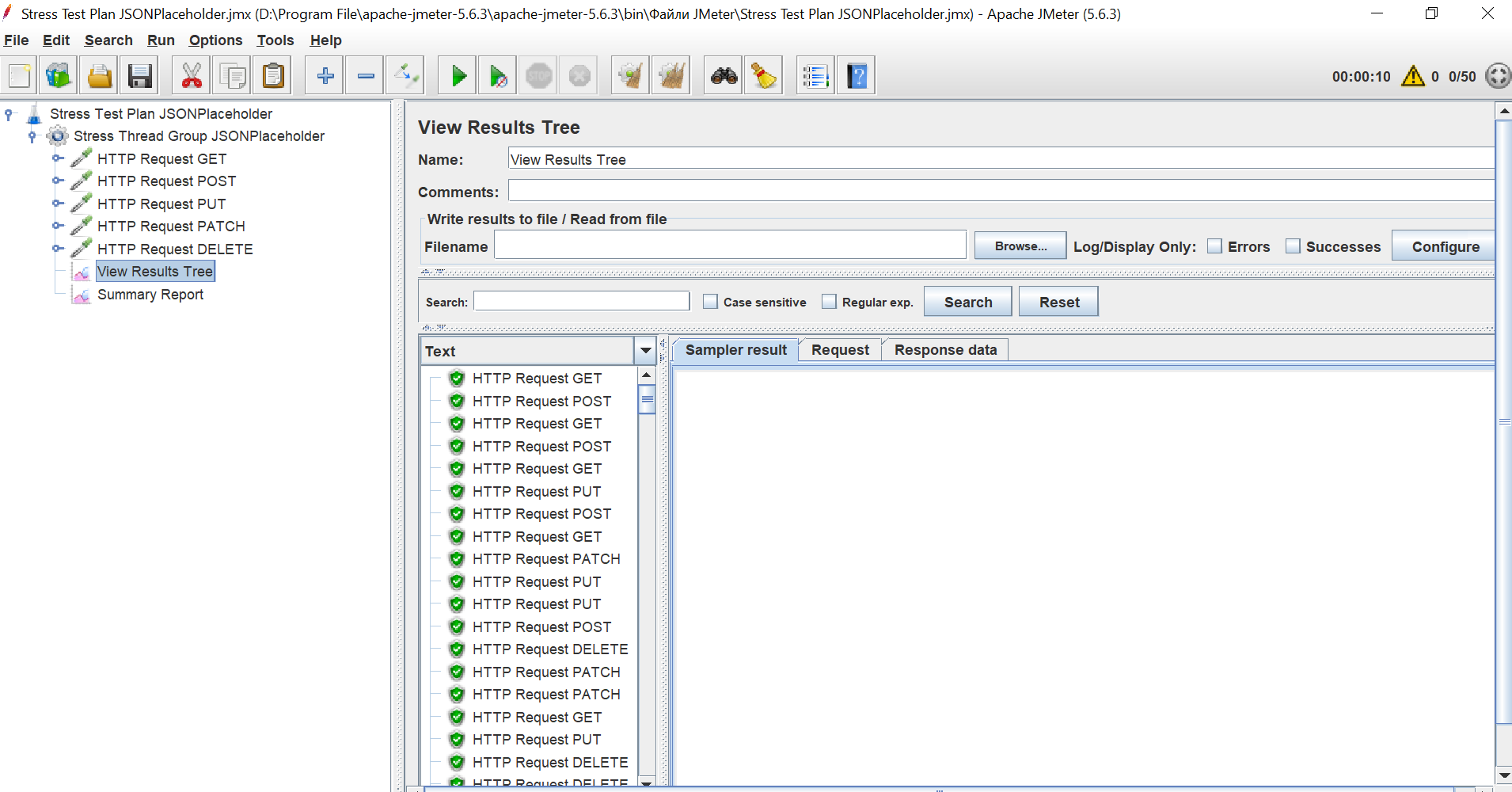
*Тестоване API витримує навантаження у від 50 до 500 одночасних користувачів без втрати стабільності.*

*Всі HTTP-методи (GET, POST, PUT, PATCH, DELETE) працюють коректно та повертають очікувані коди відповіді.*

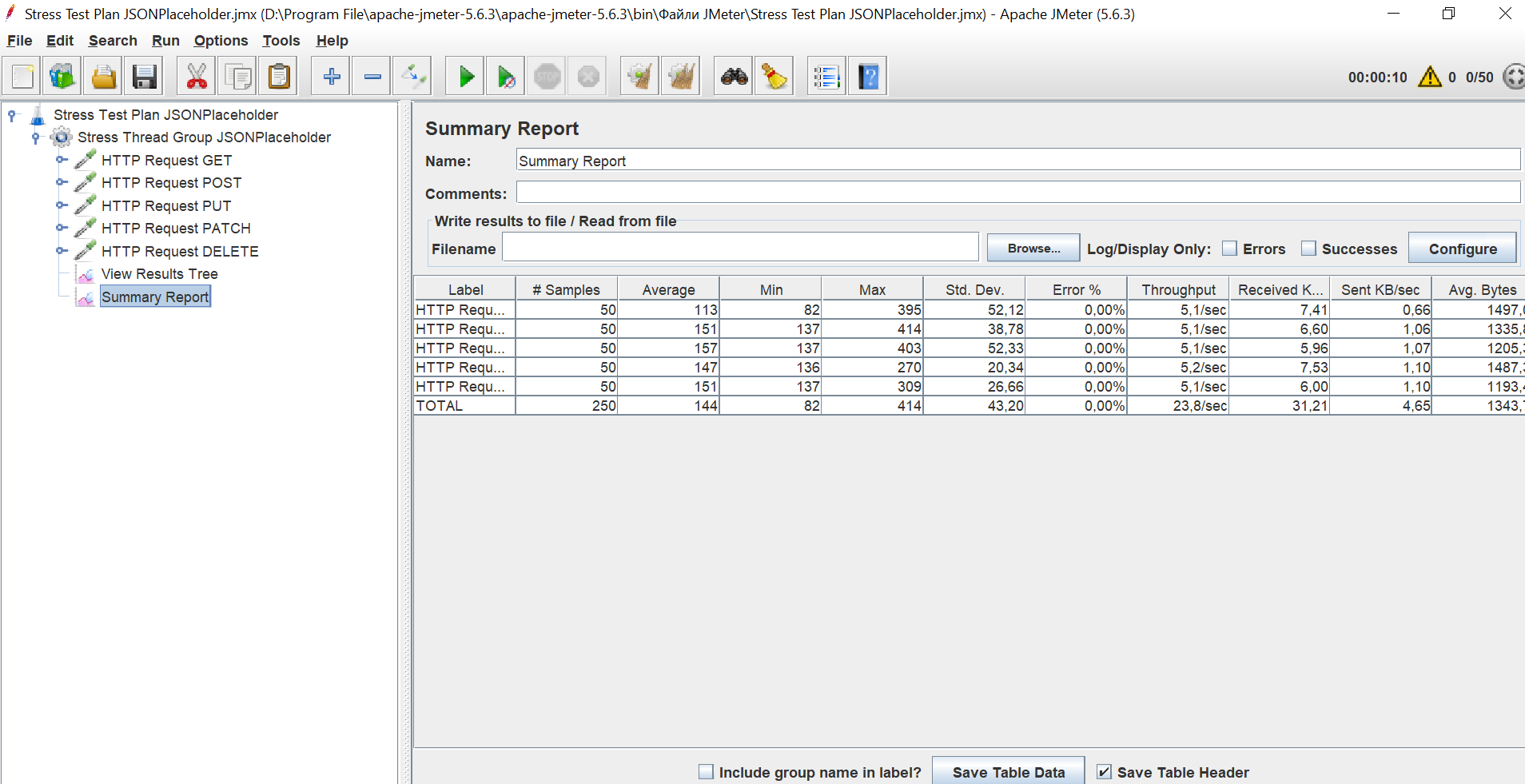
*Сервіс зберігає працездатність при стрес-навантаженні, час відгуку вкладається у встановлені 5 секунд.*

**Кількість користувачів = 50**

*View Results Tree*

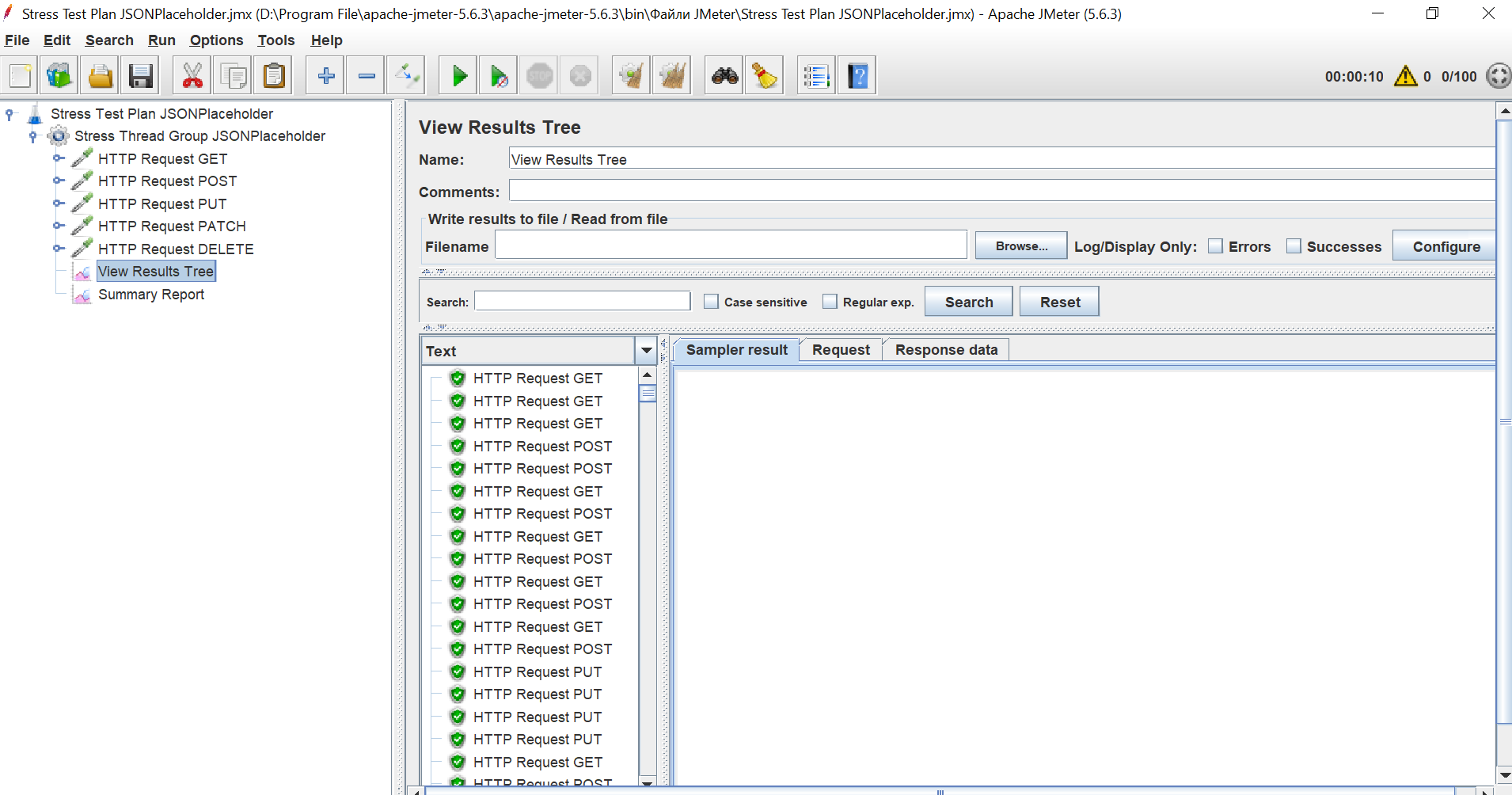


*Summary Report*

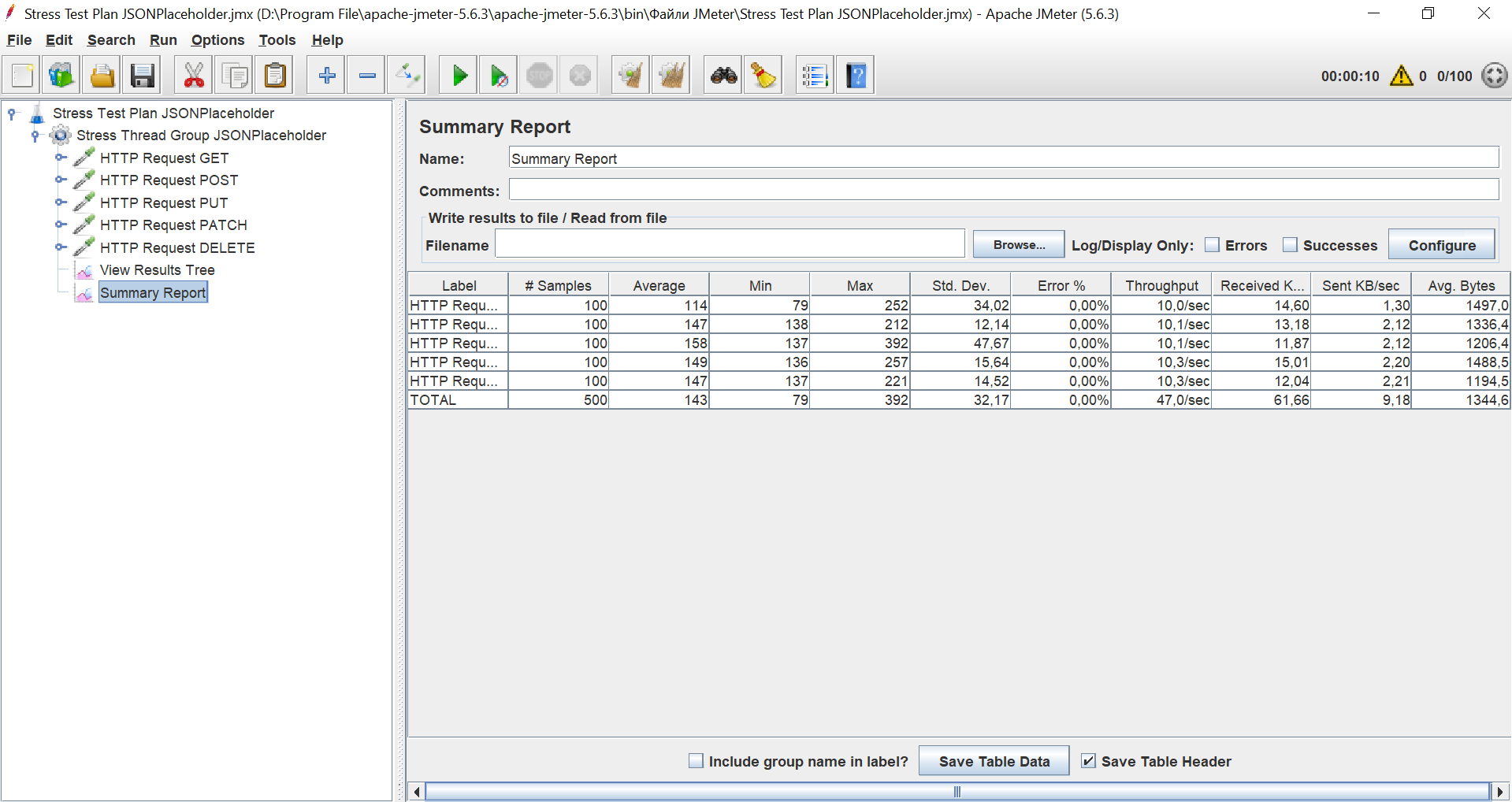


**Кількість користувачів = 100**

*View Results Tree*

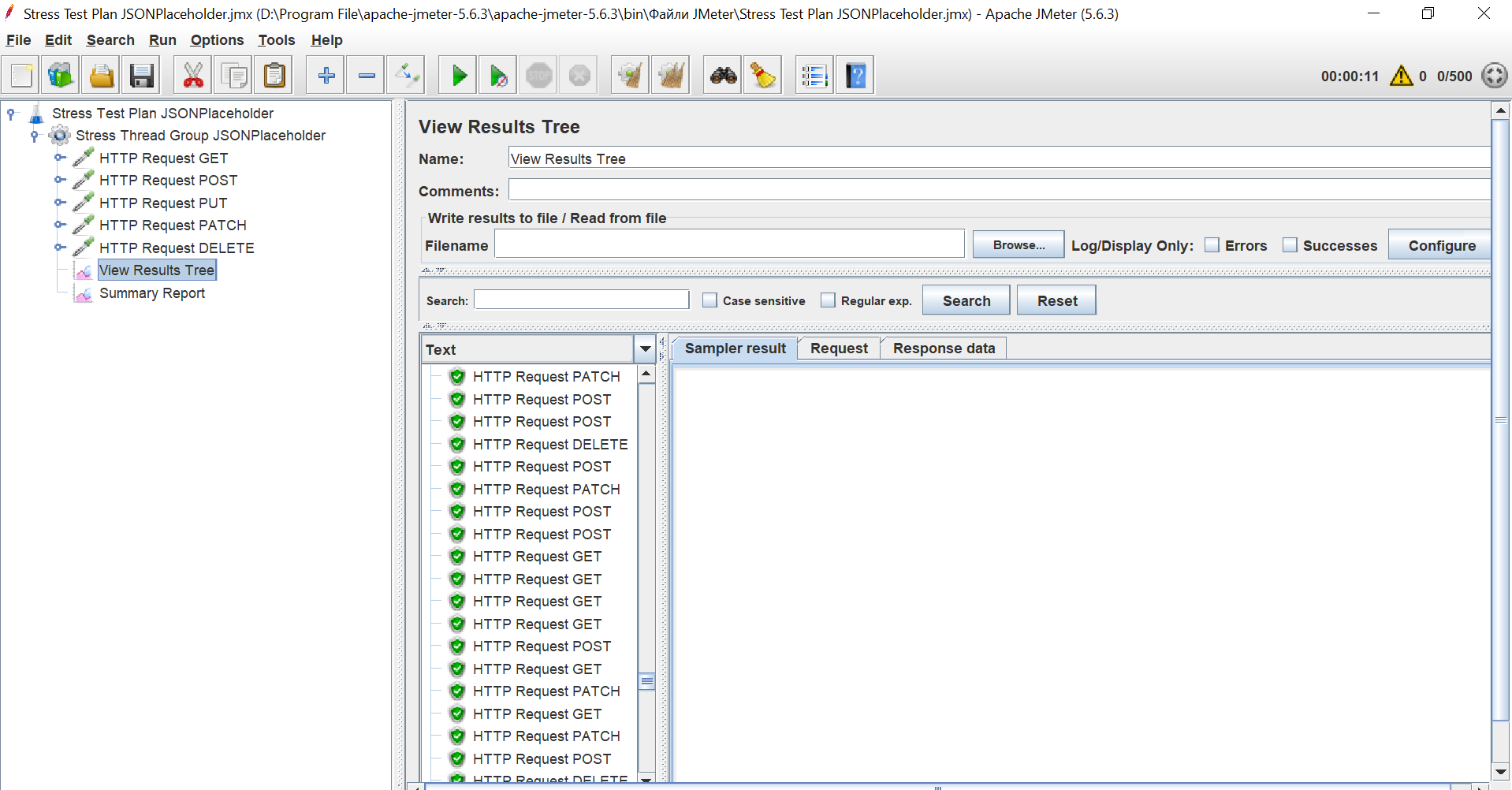


*Summary Report*

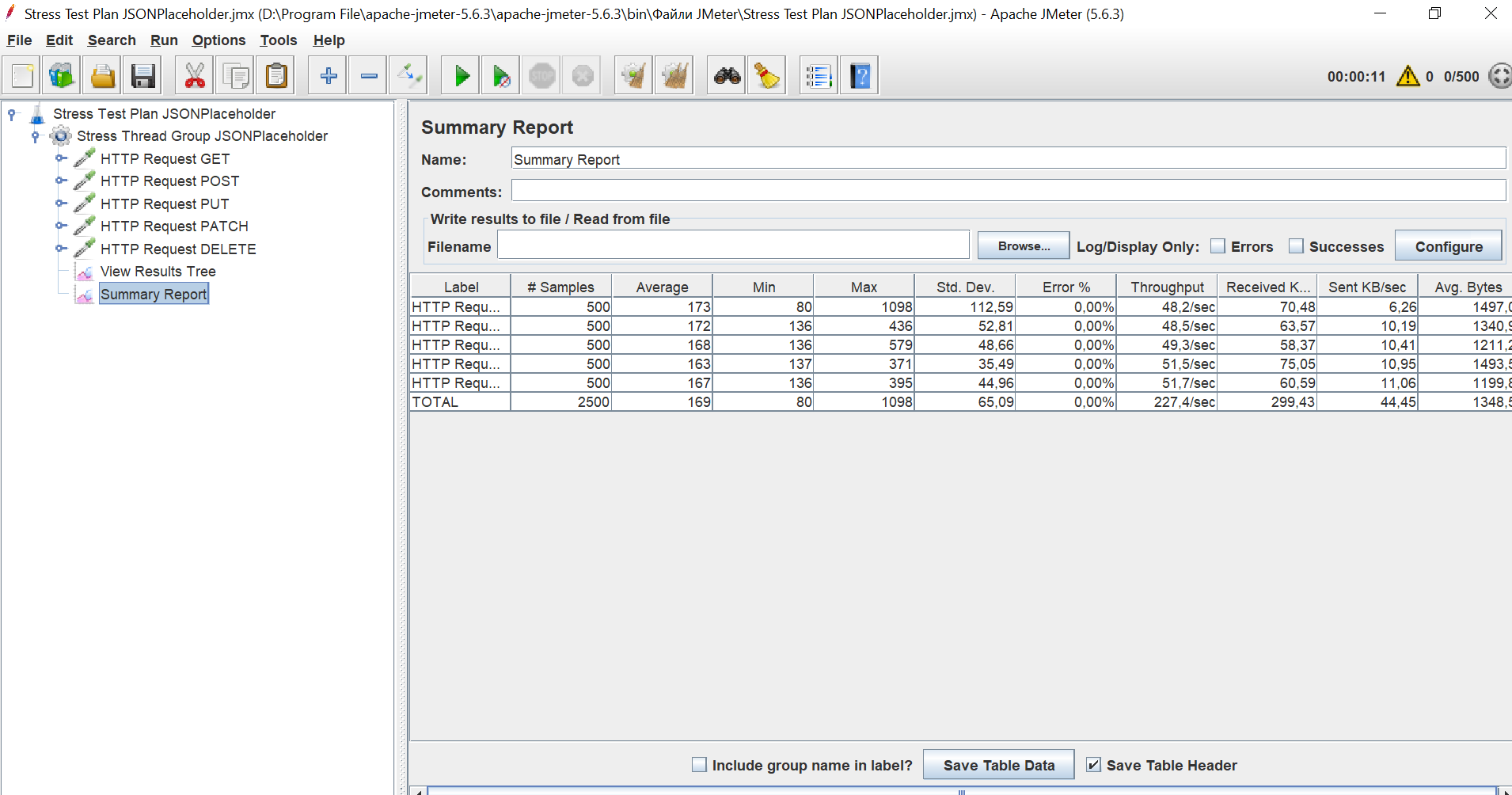


**Кількість користувачів = 500**

*View Results Tree*

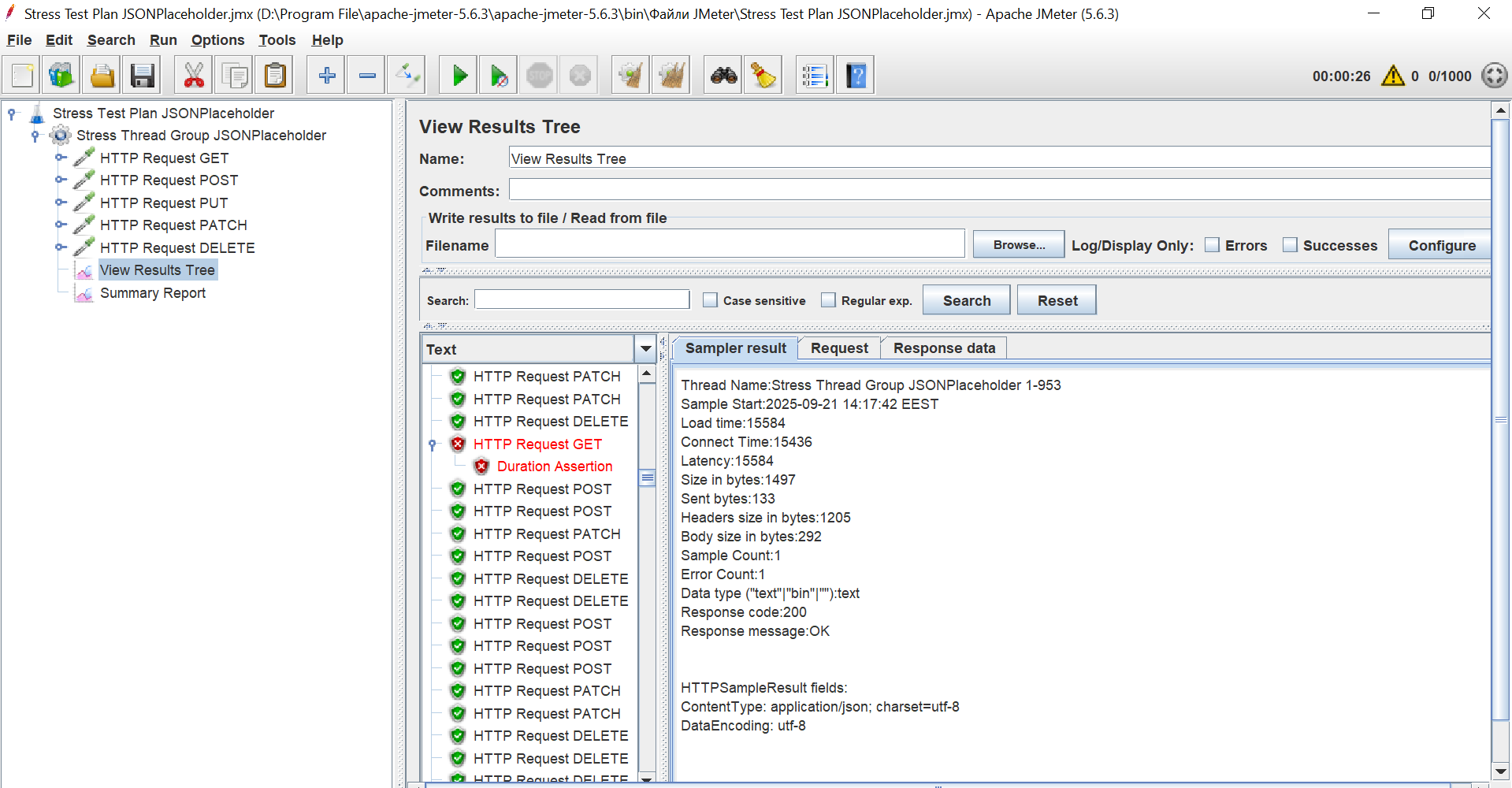


*Summary Report*

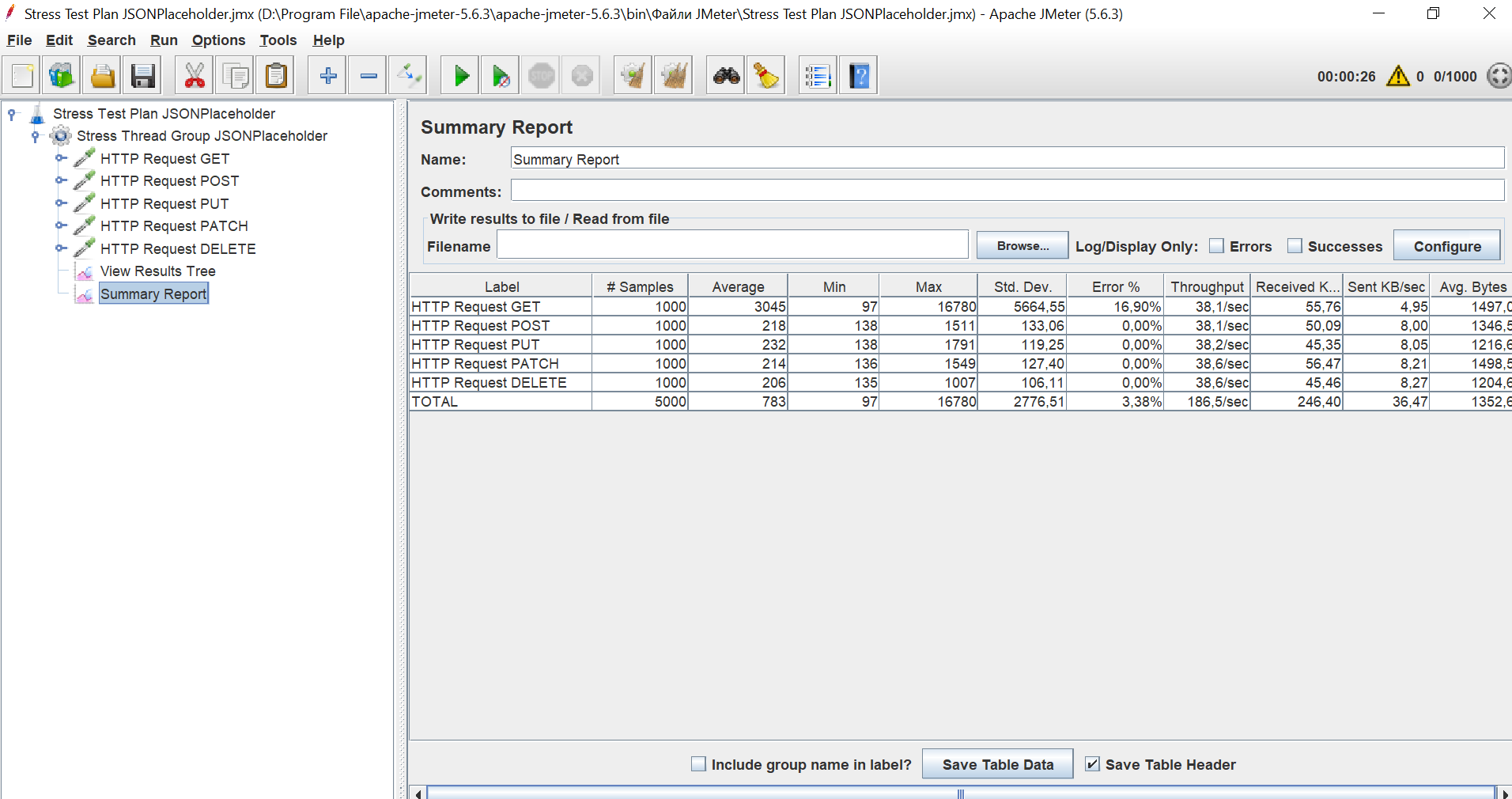


**Кількість користувачів = 1000**

*View Results Tree*



*Summary Report*



**Налаштування тесту:**

*Кількість користувачів: 1000*

*Ramp-up: 10 секунд*

*Тривалість: 60 секунд*

*Використані методи: GET, POST, PUT, PATCH, DELETE*

*Для кожного запиту були застосовані асерти: Response Code, Response Headers, Duration (<5000 мс), URL Assertion.*

*Лістенери: View Results Tree та Summary Report.*

**Результати виконання:**

*Запити POST, PUT, PATCH, DELETE залишились у зеленій зоні (усі асерти виконані успішно).*

*Запит GET не витримав навантаження:*

*Response Code = 200 (OK)*

*Проте Duration Assertion спрацював як помилка, оскільки фактичний час відгуку = 16 сек (> 5 сек).*

*Це вказує на суттєве зростання затримки при високій кількості користувачів.*

**Інтерпретація:**

*Система здатна обробляти одночасні запити до рівня ~500 користувачів без значних втрат продуктивності.*

*Починаючи з ~1000 користувачів, метод GET стає вузьким місцем, демонструючи надмірний час відгуку.*

*Інші методи під час тесту витримали навантаження, але прогнозовано можуть також деградувати при подальшому збільшенні трафіку.*

**Висновок:**

*Критична межа для цього API знаходиться у діапазоні між 500–1000 користувачів.*

*Метод GET потребує оптимізації.*

**Створений тест-план додай у свій репозиторій в GitHub.**