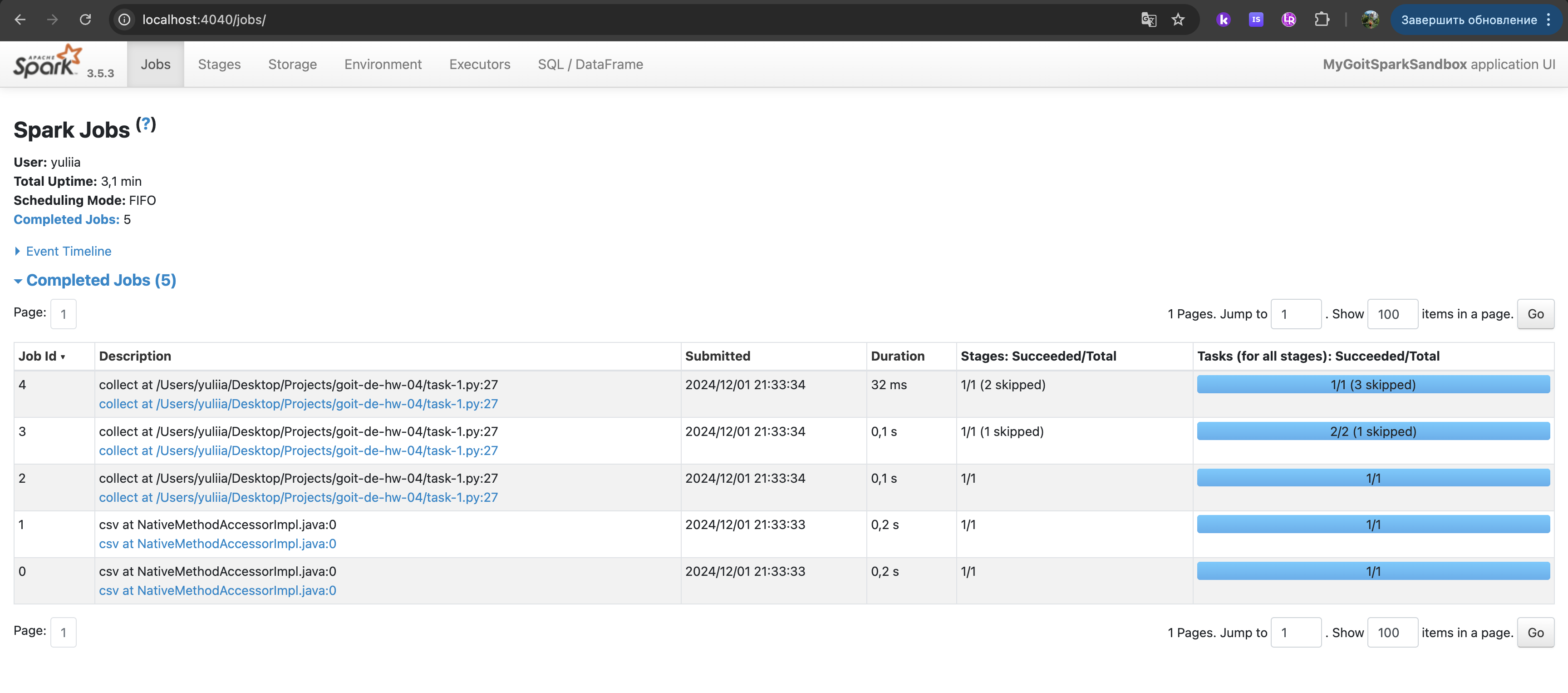
**Домашнє завдання до теми «Apache Spark. Оптимізація та SparkUІ»**

**Частина 1**

**Результат:**

**Job 1**: Spark ініціалізує завантаження даних із диска в DataFrame. Це включає визначення схеми даних та читання рядків із CSV-файлу.

**Job 2**: Spark переносить дані між розділами, щоб забезпечити рівномірний розподіл. Це корисно для паралельного обчислення.

**Job 3**: Spark виконує всі перетворення:

* Фільтрує рядки, де final\_priority < 3.
* Вибирає стовпці unit\_id і final\_priority.
* Групує дані за unit\_id і рахує кількість записів у кожній групі.

**Job 4**: Spark обчислює результат фільтрування, вибираючи тільки ті групи, де count > 2.

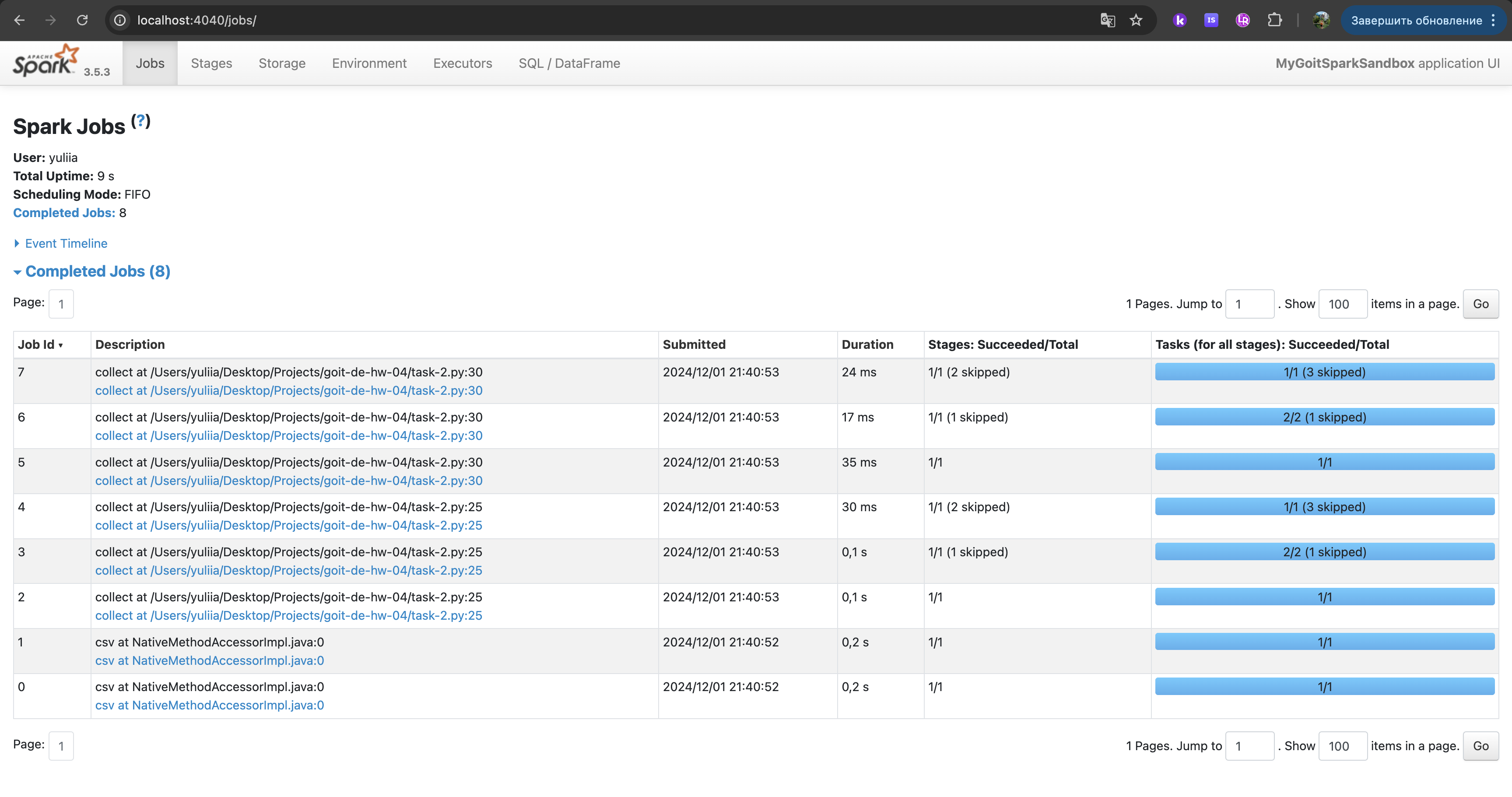
**Job 5**: Spark виконує всі попередні трансформації, якщо вони ще не були виконані, і збирає результати у формат, який може бути використаний у Python.

**Частина 2**

**Чому при додаванні однієї проміжної дії nuek\_processed.collect(), отримано аж на 3 Job більше?**

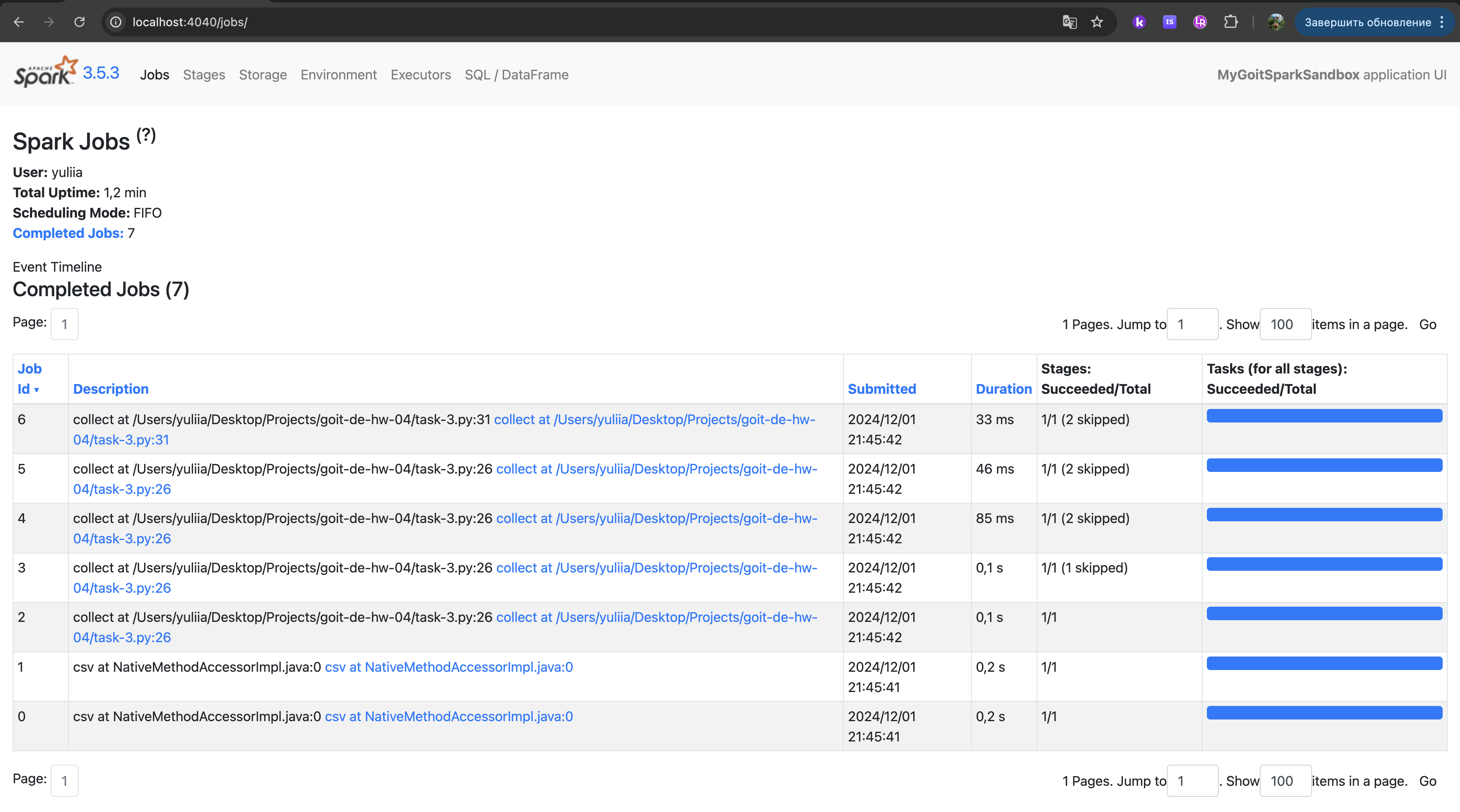
Додавання проміжного collect() у код створює значний вплив на роботу Spark. Це пов’язано з тим, що collect()є дією (**action**), яка змушує Spark завершити виконання всіх попередніх трансформацій та завантажити результат у локальну пам’ять.

В результаті Spark двічі виконує обчислення:

1. Перший collect() викликає обчислення для всіх попередніх трансформацій (до groupBy і count включно).
2. Другий collect() запускає обчислення ще раз для додаткової фільтрації where("count > 2").

Тому кількість Jobs зросла:

* 1 Job для читання CSV-файлу.
* 1 Job для repartition.
* 1 Job для фільтрації where("final\_priority < 3").
* 1 Job для select + groupBy + count.
* 1 Job для першого collect().
* Після першого collect() Spark завершує всі попередні обчислення.
* Далі, при другому collect():
  + 1 Job для фільтрації where("count > 2").
  + 2 Jobs для повторного обчислення групування (groupBy + count) і фільтрації.

**Частина 3**

**Чому при використанні cache() ми зменшили кількість Job?**

1. Перше виконання дій на закешованому DataFrame (наприклад, перший collect()) включає всі обчислення до точки кешування та зберігає результат. Це 5 Jobs.
2. Подальші дії на закешованих даних виконуються без повторних обчислень і лише на закешованих результатах. Це зменшує додаткові Jobs до 2 (для фільтру where("count > 2") та фінального collect()).

**Загалом: 7 Jobs**.