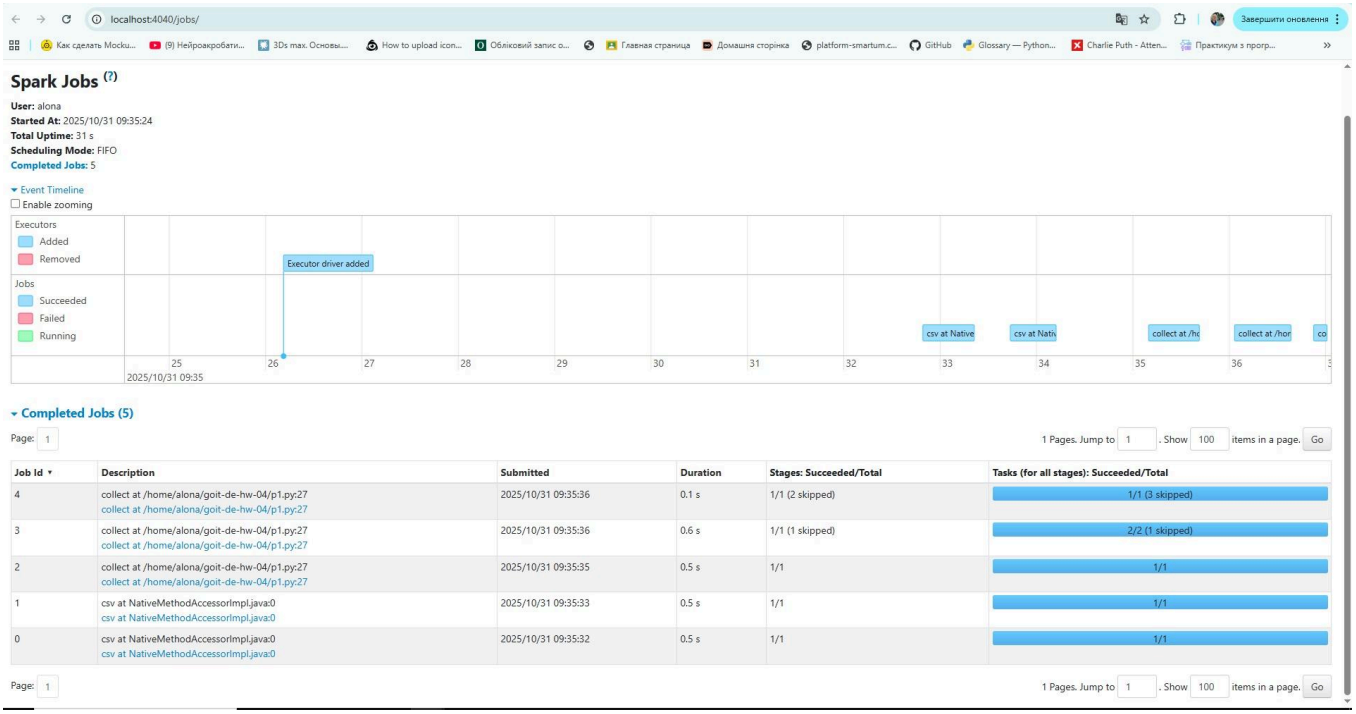
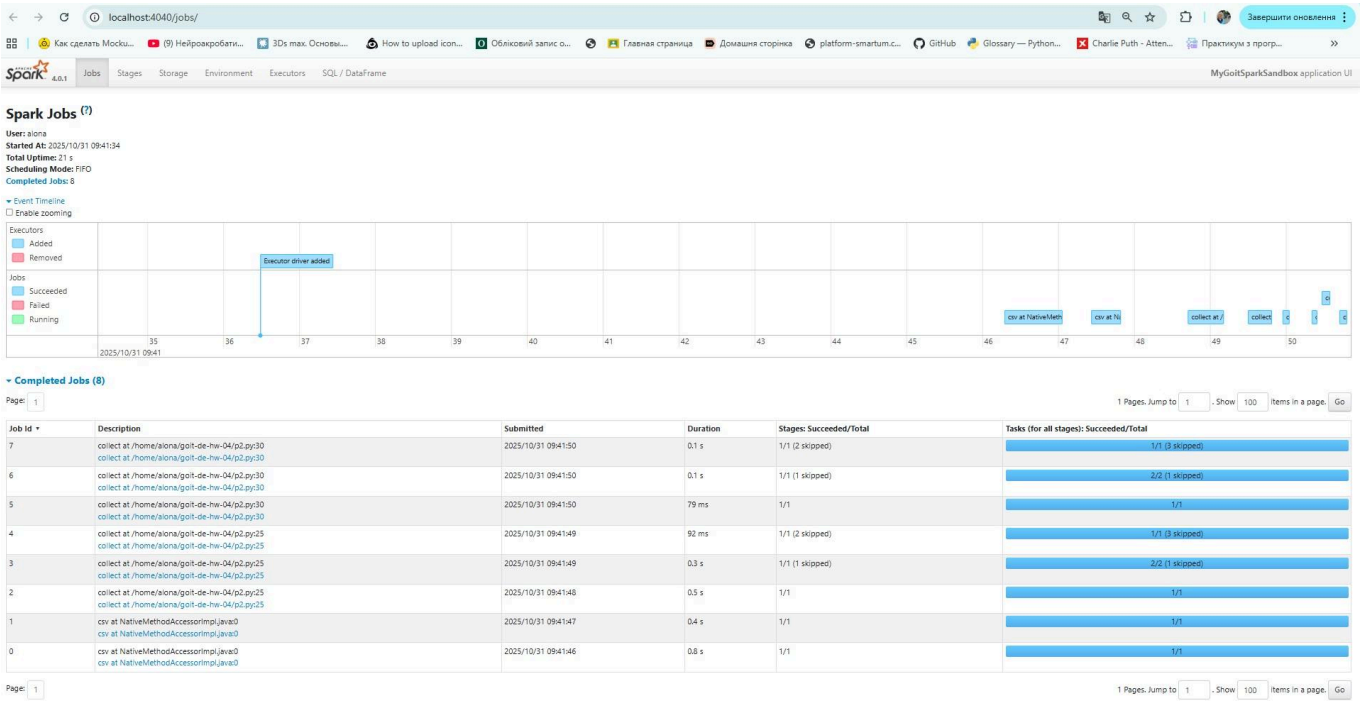
1. Spark виконує 5 jobs, бо Spark обчислює дані ліниво й кожна дія або shuffle створює новий DAG (Directed Acyclic Graph). Відбуваються наступні дії: перше читання CSV для визначення схеми даних (inferSchema), повторне читання файлу при побудові DataFrame, дія .collect() — перший прохід для обчислення групування, фільтрація count > 2 викликає новий DAG, остаточне збирання результатів у пам’ять драйвера. Skipped у Spark не виконував частину обчислень, бо результати вже були отримані раніше. Це означає, що Spark оптимізує роботу автоматично.



2. Перший collect(): Spark читає CSV двічі, бо inferSchema=True робить окремий прохід для визначення схеми + основне читання дає 2 jobs із описом csv at NativeMethodAccessorImpl.java. Далі виконує repartition → where → select → groupBy().count() і збирає результат і дає ще 1 job (shuffle-агрегація). Разом на перший collect() отримуємо 3 jobs.
   Ще одна трансформація: .where("count > 2") і знову collect().
   Оскільки нічого не кешували, другий collect() знову запускає весь ланцюжок від джерела: знову 2 jobs для CSV (інференс схеми + читання), і 1 job для агрегації/фільтра та збору. Разом на другий collect() дає ще 3 jobs.
   Отже, було 5, стало 8 - різниця +3 jobs саме через другий collect().



3. Cache() зменшив кількість jobs, бо другий collect() вже брав дані з кешу, а не запускав повний перерахунок від CSV. До cache() маємо ланцюжок: csv - repartition - where - select - groupBy().count() - collect(). Другий collect() без кешу знову проходить весь ланцюжок - це ще 3 jobs (окремо читання CSV з inferSchema, shuffle-агрегація тощо). З cache(): а)Перший collect() матеріалізує кеш: читає CSV, рахує groupBy().count() і кладе результат у пам’ять/диск (рівень за замовчуванням MEMORY_AND_DISK); б)другий collect() + фільтр where("count > 2") працює поверх уже закешованого DataFrame - без повторного читання CSV і без повторної агрегації. Це зазвичай один легкий job (інколи з “skipped” стадіями, бо Spark перевикористав проміжні результати). Тому сумарно jobs стало менше: перший прохід (читання+агрегація+кеш) - це кілька jobs; другий прохід (фільтр по кешу) - це один невеликий job.