**Handling the Report Generation Issue:**

One of the first big issues I faced in my project was the app crashing during scheduled report generation. This was a task that I picked up as it was a major pain point. The reports were being generated periodically (quarterly, half-yearly, and yearly) for over 10,000 investors, and the system couldn’t handle the load.

Problem Identification:

When I started analyzing the problem, I found that the SQL query used for report generation wasn’t optimized. The query had subqueries at the column level, which made it very slow and resource-intensive. This caused the app to run out of resources and crash. Also, even though each report was independent, they were all running on a single thread, which further slowed things down.

Solution:

**Optimizing SQL Queries:**

I first focused on optimizing the SQL query. The subqueries at the column level were slowing everything down because for every row, the system had to run those subqueries. I rewrote the query to eliminate those subqueries and used joins and group-by operations where needed. This reduced the time it took to generate reports and decreased the load on the database.

**Implementing Multithreading:**

Since each report was independent, there was no need to run them in a single thread. I implemented multithreading using Java’s ExecutorService to process multiple reports simultaneously. By doing this, I could generate several reports at the same time, which reduced the overall processing time and made better use of system resources.

**Improving Error Handling and Load Management:**

To avoid any failures bringing down the whole system, I added better error handling in each thread. If a single report failed due to any issue, like a network error, it wouldn’t crash the entire report generation process. I also distributed the load across the threads to avoid overloading the system.

**Caching Data:**

I noticed that some of the data being accessed in the reports was the same for multiple reports. To reduce the load on the database, I introduced caching for commonly accessed data. This meant the system didn’t have to hit the database repeatedly for the same information, which further sped up the process.