Project: TaskMaster Processing System (TMPS) – Original Code Review
Issue Analysis Report

Hamza Zarour

1) All issues you identified in the original code

Connection / ConnectionManager

- **Always creates a new Connection for each job.**
- **Does not reuse connections.**
- **Returns null when more than 10 connections are created.**
- **closeConnection does not really manage anything.**
- **No real “pool” or blocking behavior.**

JobExecutor (original naive executor)

- **Contains a big if/else on string job types ("EMAIL", "DATA", "REPORT").**
- **Acts as a “BIG class”: chooses type, runs logic, executes SQL, handles connection, prints logs.**
- **Directly depends on ConnectionManager and opens/closes connections itself.**
- **No permission checks at all.**

Templates / TemplateManager

- **Every time we need a job from a template, the code calls simulateHeavyLoad and rebuilds a HeavyTemplate.**
- **No reuse of templates: heavy cost is paid for each job.**
- **TemplateManager is responsible for heavy work, building templates, and creating jobs, all in one class.**
- **No registry for templates.**

MainApp

- **Builds heavy templates every time it runs.**
- **Calls the naive executor directly.**
- **Does not use a connection pool, proxy, or registry.**
- **Contains logic that should belong to the architecture.**

2) Why each issue is problematic

Connection / ConnectionManager

- **Creating a new connection per job is slow and wastes resources.**
- **Creating a template is also slow and wastes resources.**
- **Without reuse or blocking, the system cannot control how many connections are actually in use.**

JobExecutor

- **A big if/else on strings makes the code hard to extend and easy to break with typos.**
- **A "BIG class" is difficult to test, maintain, and reason about.**
- **No permission checks means any user can run any job, which is a security problem.**

Templates / TemplateManager

- **Rebuilding heavy templates every time seriously hurts performance (the simulated 3 seconds delay).**
- **Putting all responsibilities inside TemplateManager makes it harder to change or extend the template logic.**
- **Lack of a registry means there is no central place to manage available templates.**

3. How the Refactoring Solves the Problems

- **ConnectionPool fixes connection creation, reuse, limits, and blocking behavior.**
- **Strategy Pattern removes the if/else, isolates job logic, and allows adding job types without modifying the executor.**
- **Proxy Pattern adds permission checks, connection handling, logging, and time measurement and keeping the executor clean.**
- **Prototype Pattern solves template rebuilding by creating each heavy template once and cloning it efficiently.**