An-Najah National University

Course: Advanced Software Engineering

Semester: First 2025/2026

Refactored the TMPS System Running

Design Decision Document

Instructor: Dr. Mustafa Assaf

Prepared By

Fathi Al Heelo

12323885

Date: DEC 1,2025

Why was each design pattern chosen?

1.1 Strategy Pattern

The system has maluti class some behavior and job sash as (Email, reporting, data processing) but each job has unique execution logic, instead of using if statements, we can apply the stratigy pattern his pattern assist for encapsulate job specific behavior in separate strategy classes and allow adding new job without modifying existing code(applying OCP) also improve clearlarity and reduce duplicate.

1.2 Proxy Pattern

JobExecutionProxy intercepts calls to JobExecutor to perform: Permission Validation and execution time measurement and logging, and monitoring.

We use the proxy pattern because it allows adding cross cutting behavior without editing JobExecutor, also maintaining SRP.

1.3 Prototype Pattern

Templates are heavy to build, so a prototype allows cloning them instead of rebuilding every time.

This improves performance and reduces repeated work.

1.4 Factory Pattern

Helps to select the correct strategy for each job type .

And interest object creation , supports adding new job type easily.

1.5Registry Pattern

Stores all templete in one place and returens clones when needed .

This avoids duplication and make tempelte clean and scalable.

What alternatives you considered?

For example, when using if/else instead of a strategy, this is rejected because it makes JobExecutor large and hard; the strategy pattern solves that and applies OCP.

And logging and permissions inside JobExecutor are rejected because it violates SRP and makes the class harder to maintain. Proxy provides a cleaner solution.

And when creating templates with new each time it is slow and inefficient for heavy templates prototype offers faster cloning .

How the patterns interact?

Strategy handles job execution, and the factory selects the strategy.

Proxy wraps the executor to add security and logging .

And protoype and registry manage efficient template creation .

Together, they built a clean pipeline with each component doing one responsibility.

How your architecture improves scalability, flexibility, and maintainability?

Scalability ConnectionPool and prototype reduce heavy resource use , and the systeam can handle more job without loss performance.

Flexibility can add any job types and templetes or proxy

Behaviors can be added easily without modifying base classes.

Maintainability clear separation of responsibilities and solid principles make the code easy to read and test and update.