# **Pull Request Summary and Modernization of Legacy ETL Pipeline**

## **Introduction**

This document is a comprehensive, hands-on exercise designed for intermediate software developers looking to elevate their development workflow using GitHub Copilot Chat alongside the GitHub CLI. In this exercise, you'll begin by forking a dedicated repository and then work on creating a detailed pull request (PR) summary via the CLI. You will further enhance that summary with GitHub Copilot Chat's insights to ensure your code changes are well-documented and easy to review.

Additionally, you will modernize Acme Corporation’s legacy Python-based ETL pipeline—hosted in the separate repository [acme-corp/etl-pipeline](https://github.com/datacouch-io/acme-corp/etl-pipeline)—by upgrading the code from Python 2 to Python 3, implementing robust error handling and logging, and integrating a comprehensive pytest suite for automated testing. This exercise reinforces best practices for code modernization and demonstrates how to communicate technical improvements effectively through clear and informative PR summaries.

## **Objectives**

* **Clone and Work on the Repository:**Learn to clone the repository, create a feature branch, and work on a modernized version of the legacy Java ETL pipeline.
* **Modernize Legacy Java Code:**Update legacy Java code by refactoring it with modern language features, improved error handling, and enhanced logging practices.
* **Create and Refine a Pull Request:**Use the GitHub CLI to create a pull request and then leverage GitHub Copilot Chat to generate and refine an effective PR summary that details the changes and improvements.
* **Leverage AI-Assisted Coding:**Utilize GitHub Copilot Chat for generating precise commands, code updates, and an enhanced pull request description, ensuring clear communication during code reviews.

|  |
| --- |
| **Note**: Below prompts are provided solely as examples of effective prompt construction. As participants, we must engage in an iterative process to refine these examples and develop the optimal prompt for the task at hand. |

## **Step 1: Clone the Repository Using GitHub Copilot Chat**

**Objective:**Clone the legacy repository and navigate to the etl-pipeline folder exclusively through chat commands.

**Process and Prompts:**

Start by instructing Copilot Chat to clone the repository:

|  |
| --- |
| @workspace: Initialize a new clone of the repository from "https://github.com/datacouch-io/acme-corp.git" and navigate to the etl-pipeline folder. Ensure that the entire process is handled via Copilot Chat. |

|  |
| --- |
| **Note**: Above prompt would help you to generate the required commands. Once commands are created, hover on the commands and click on **Insert into Terminal** option |

1. This prompt is detailed so that Copilot understands you need an automated clone process without any manual terminal commands.

After cloning, verify that the project was correctly imported:

|  |
| --- |
| @workspace: Provide a detailed overview of the directory structure within the etl-pipeline folder, including all key subdirectories and files. |

|  |
| --- |
| **Note**: Above prompt would help you to generate the required commands. Once commands are created, hover on the commands and click on **Insert into Terminal** option |

1. This confirmation helps you ensure that every file expected (e.g., source code, tests, configuration files) is present.

**Challenges:**

* **Challenge 1**: Consider whether additional folders (e.g., for configuration or scripts) are necessary. Use iterative /fix prompts to add any missing directories until the workspace is fully complete.
* **Challenge 2**: Experiment with including branch information or remote tracking details to confirm the clone process meets all project requirements.

**Step 2: Create a New Feature Branch Using GitHub Copilot Chat**

**Objective:**Create and switch to a dedicated feature branch for the modernization work without any manual intervention.

**Process and Prompts:**

Instruct Copilot Chat with:

|  |
| --- |
| /start: Generate the Git command to create and switch to a new branch named "feature/modernize-java-etl" in the cloned repository. |

1. This prompt clearly specifies the branch name and that you want to both create and check out the branch immediately.

To ensure the branch was created, ask:

|  |
| --- |
| @workspace: List all Git branches in the repository to confirm that "feature/modernize-java-etl" exists. |

|  |
| --- |
| **Note**: Above prompt would help you to generate the required commands. Once commands are created, hover on the commands and click on **Insert into Terminal** option |

**Challenges:**

* **Challenge 1**: Refine the branch naming convention if needed, such as incorporating a ticket number or additional description, using further iterative prompts until the branch name perfectly aligns with your workflow standards.

**Step 3: Analyze and Document Legacy Code Issues**

**Objective:**Examine the legacy code in LegacyETL.java to identify outdated practices and areas for improvement.

**Process and Prompts:**

Open the file using:

|  |
| --- |
| #file: Open the file "LegacyETL.java" located in the "com/acme/legacy/" directory. |

1. Generate a comprehensive summary by instructing:

|  |
| --- |
| /start: Generate a detailed summary of the LegacyETL.java file. List and explain at least five outdated practices or issues, such as the use of HttpURLConnection instead of HttpClient, insufficient error handling, reliance on System.out.println for logging, limited usage of modern Java features, and the absence of automated tests. |

1. This prompt is intentionally detailed—it specifies exactly what issues to list and gives concrete examples to steer the analysis.

Request modern alternatives with:

|  |
| --- |
| /start: Suggest modern alternatives for the legacy components in LegacyETL.java, such as using Java 11 HttpClient, try-with-resources, Java Streams for processing, and SLF4J for logging. |

**Challenges:**

* Expand the analysis to include hardcoded values and poor resource management. Use iterative prompts to generate additional code snippets that illustrate modern replacements for each outdated practice.

**Step 4: Formulate a Comprehensive Modernization Plan**

**Objective:**Develop a clear, detailed modernization plan that outlines every step required to update the ETL pipeline.

**Process and Prompts:**

Instruct Copilot Chat:

|  |
| --- |
| /start: Generate a detailed modernization plan for LegacyETL.java. The plan should include: - Upgrading to Java 11+ features (e.g., using HttpClient, try-with-resources, and Java Streams) - Replacing System.out.println with SLF4J logging - Implementing robust error handling with custom exceptions - Replacing hardcoded configuration values with environment variables or a properties file - Adding automated tests using JUnit and Mockito - Documenting changes with proper Javadoc comments |

|  |
| --- |
| **Note**: Above prompt would help you to generate the required commands. Once commands are created, hover on the commands and click on **Insert into Terminal** option |

1. The prompt lists each specific area to address, ensuring that the plan is comprehensive.

Extend the plan by including additional functionality:

|  |
| --- |
| /start: Extend the modernization plan to add a new method that calculates total sales (price multiplied by quantity) for each record, and handles missing or invalid data with appropriate logging. |

**Challenges:**

* Divide the plan into smaller milestones or subtasks and generate a checklist for each modernization step. Refine your plan by incorporating expected performance improvements and other benefits.

**Step 5: Migrate to Modern Java Features**

**Objective:**Update the legacy ETL pipeline to use modern Java features, starting with the HTTP client and streamlined resource management.

**Process and Prompts:**

Replace outdated HTTP code:

|  |
| --- |
| /start: Refactor the fetchSalesData method in LegacyETL.java to use Java 11 HttpClient instead of HttpURLConnection. Ensure that the new implementation uses try-with-resources for proper resource management and includes timeout configuration. |

1. This detailed prompt instructs Copilot Chat to implement the modern alternative while specifying necessary improvements.

Modernize data processing:

|  |
| --- |
| /start: Update the processData method in LegacyETL.java to use Java Streams for filtering records that contain the "product\_id" field. |

1. Enhance error handling:

|  |
| --- |
| /fix: Enhance the fetchSalesData method to include proper exception handling using custom exceptions, and log errors using SLF4J. |

**Challenges:**

* **Challenge 1**: Consider implementing retry logic for HTTP requests using HttpClient and refining your prompts until the code is robust under heavy load.
* **Challenge 2**: Test and compare the performance of the new implementation against the legacy approach, adjusting prompts based on Copilot’s feedback.

**Step 6: Integrate SLF4J Logging**

**Objective:**Replace all System.out.println calls with SLF4J logging for standardized log management.

**Process and Prompts:**

Add logging dependencies by instructing:

|  |
| --- |
| /start: Generate Maven dependency entries for SLF4J API (version 2.0.7) and Logback Classic (version 1.4.5) to be added to the pom.xml. |

|  |
| --- |
| **Note**: Above prompt would help you to generate the required commands. Once commands are created, hover on the commands and click on **Insert into Terminal** option |

1. Update logging statements:

|  |
| --- |
| /start: Refactor LegacyETL.java to replace all System.out.println calls with SLF4J logging calls. Use appropriate log levels (e.g., logger.info for informational messages, logger.error for error conditions). |

1. Request a detailed explanation:

|  |
| --- |
| /explain: Describe how using SLF4J and Logback improves log management compared to System.out.println, focusing on log level control, configurability, and performance. |

**Challenges:**

* **Challenge 1**: Experiment with additional log levels and generate prompts to create a logback configuration file (e.g., logback.xml) that specifies appenders and formatting patterns.
* **Challenge 2**: Verify that full exception stack traces are captured in the logs, refining prompts as needed.

**Step 7: Introduce Advanced Data Processing**

**Objective:**Add a new method to calculate total sales for each record and integrate it into the ETL pipeline.

**Process and Prompts:**

Create a new method:

|  |
| --- |
| /start: Generate a new method in LegacyETL.java named calculateTotalSales. This method should iterate over a list of JSON records, calculate total sales as the product of price and quantity (if both are present), and add a new field "total\_sales" to each record. |

|  |
| --- |
| **Note**: Above prompt would help you to generate the required commands. Once commands are created, hover on the commands and click on **Insert into Terminal** option |

1. Integrate the method:

|  |
| --- |
| /fix: Update the main method in LegacyETL.java to incorporate the calculateTotalSales method into the data processing workflow. Ensure that the output from calculateTotalSales is used along with the cleaned data. |

1. Enhance data validation:

|  |
| --- |
| /fix: Update calculateTotalSales to log a warning using SLF4J if either the price or quantity is missing or invalid. |

**Challenges:**

* **Challenge 1**: Explore additional data transformations, such as converting date strings to LocalDate objects or formatting currency values, by generating and refining prompts.
* **Challenge 2**: Experiment with merging this new method with processData for a streamlined pipeline.

**Step 8: Replace Hardcoded Values with Environment-Based Configuration**

**Objective:**Refactor the code to eliminate hardcoded values by using environment variables or configuration files, thereby enhancing flexibility and maintainability.

**Process and Prompts:**

Replace hardcoded values:

|  |
| --- |
| /start: Refactor LegacyETL.java to replace hardcoded values for the data URL and output file path with environment variables named SALES\_DATA\_URL and OUTPUT\_FILE\_PATH. Provide default values if the variables are not set. |

|  |
| --- |
| **Note**: Above prompt would help you to generate the required commands. Once commands are created, hover on the commands and click on **Insert into Terminal** option |

1. Open Edit with Copilot ***(cmd/win+shift+I)***, generate a configuration loader:

|  |
| --- |
| Generate code in LegacyETL.java to load configuration settings from a properties file named app.properties, retrieving keys such as sales.data.url and output.file.path. |

1. Improve configuration robustness:

|  |
| --- |
| /fix: Update the configuration loader to log warnings and use default values when required configuration keys are missing. |

**Challenges:**

* **Challenge 1**: Extend the configuration to support multiple environments (development, staging, production) by refining your prompts.
* **Challenge 2**: Generate a sample .env.example or app.properties file with detailed key descriptions using Copilot Chat.

**Step 9: Implement Automated Testing with JUnit**

**Objective:**Develop a suite of automated tests for the ETL pipeline using JUnit and Mockito, ensuring comprehensive coverage.

**Process and Prompts:**

Add testing dependencies:

|  |
| --- |
| /start: Generate the Maven dependency entry for JUnit 5 (JUnit Jupiter version 5.9.2) with test scope and add it to the pom.xml. |

1. Create a basic test class:

|  |
| --- |
| /start: Generate a JUnit 5 test class named LegacyETLTest.java in the test/ directory. The test class should include tests for fetchSalesData, processData, calculateTotalSales, and storeData methods. |

1. Integrate mocks for external HTTP calls:

|  |
| --- |
| /fix: Enhance LegacyETLTest.java by generating Mockito-based mocks for external HTTP calls in the fetchSalesData method. |

1. Expand test coverage:

|  |
| --- |
| /fix: Add additional test cases to LegacyETLTest.java to cover edge cases such as invalid data formats, missing fields, and processing large data sets. |

**Challenges:**

* **Challenge 1**: Develop tests that simulate heavy data loads and assess performance and memory usage. Iterate your test suite until all scenarios, including configuration loading and error handling, are thoroughly validated.

**Step 10: CI/CD Integration with GitHub Actions**

**Objective:**Automate the build, test, and deployment process using GitHub Actions, ensuring that your pipeline runs automatically on every push.

**Process and Prompts:**

Generate a workflow file:

|  |
| --- |
| /start: Generate a GitHub Actions workflow file named ci.yaml in the .github/workflows/ directory that sets up Java environments (Java 11, 17, and 20), installs Maven dependencies, runs the test suite using Maven, and reports the results. |

1. Refine the workflow file:

|  |
| --- |
| /fix: Enhance the ci.yaml workflow to include caching for Maven dependencies and configure it to trigger on push events to the main branch. |

**Challenges:**

* **Challenge 1:** Experiment with integrating additional quality tools (e.g., SonarQube) into your workflow. Refine your prompts until the workflow is robust and fully automated.

**Step 11: Stage, Commit, and Push Changes Using GitHub CLI**

**Objective:**Generate all necessary Git commands via GitHub Copilot Chat to stage, commit, and push your changes—completely automated.

**Process and Prompts:**

Generate Git commands:

|  |
| --- |
| /start: Generate the Git commands to stage all changes, commit them with the message "Modernize Java ETL Pipeline: Upgrade to Java 11 features, implement robust error handling, integrate SLF4J logging, and add comprehensive JUnit tests", and push the changes to the branch feature/modernize-java-etl. |

|  |
| --- |
| **Note**: Above prompt would help you to generate the required commands. Once commands are created, hover on the commands and click on **Insert into Terminal** option |

1. Verify that the branch has been updated:

|  |
| --- |
| @workspace: List all Git branches to confirm that "feature/modernize-java-etl" is updated with the latest commits. |

**Challenges:**

* **Challenge 1**: Iteratively refine your commit message to include references to specific improvements and new tests.
* **Challenge 2**: Use prompts to further clean up commit history (for example, squashing commits) ensuring a tidy Git log.

**Step 12: Create and Refine the Pull Request**

**Objective:**Generate a pull request using GitHub CLI commands and then refine the pull request summary through GitHub Copilot Chat, ensuring clear communication of all improvements.

**Process and Prompts:**

Create the pull request:

|  |
| --- |
| /start: Generate the GitHub CLI command to create a pull request from branch feature/modernize-java-etl to the main branch. The PR title should be "Modernize Java ETL Pipeline" and the description must detail: - Upgrading to modern Java features (e.g., HttpClient, try-with-resources, Java Streams) - Replacing System.out.println with SLF4J logging - Implementing robust error handling with custom exceptions - Utilizing environment-based configuration - Adding comprehensive JUnit tests |

|  |
| --- |
| **Note**: Above prompt would help you to generate the required commands. Once commands are created, hover on the commands and click on **Insert into Terminal** option |

1. Refine the pull request summary:

|  |
| --- |
| /fix: Review the current pull request description and generate an enhanced, detailed summary that explains the migration to modern Java practices, the integration of SLF4J for standardized logging, improvements in error handling and resource management, the transition to environment-based configuration, and the addition of comprehensive automated tests. |

1. Update the pull request using:

|  |
| --- |
| @git: Update the pull request description with the enhanced summary |

**Challenges:**

* **Challenge 1**: Refine your PR summary further by including specific metrics, such as code reduction or performance improvements.
* **Challenge 2**: Experiment with different formats until the final PR summary clearly and concisely communicates every aspect of your modernization work.

**Additional Advanced Challenges**

* **Implement Update Functionality:**Generate a new API route using Copilot Chat to provide a PUT endpoint for updating product details, then integrate and test it within the ETL pipeline.
* **Integrate a Global Error Boundary:**Create a custom React error boundary component through detailed Copilot Chat prompts that catches errors in the component tree and displays a fallback UI. Refine until the component handles all exceptions gracefully.
* **Refactor State Management with Context:**Migrate any shared state management (e.g., for a cart system if part of your broader application) to React Context. Generate detailed prompts to create a custom hook and refactor your components accordingly.
* **Automate Deployment with GitHub Actions:**Create and refine a GitHub Actions workflow file that automatically builds, tests, and deploys your application on every push to the main branch. Iterate your prompts until the CI/CD pipeline is fully integrated and functional.

## **Conclusion**

By following these steps, you have used GitHub Copilot to assist in every stage—from cloning the repository and creating a feature branch to modernizing the legacy Java code and generating a comprehensive pull request summary. This exercise not only modernizes the ETL pipeline using current Java best practices but also demonstrates how integrating AI-assisted tools like GitHub Copilot and the GitHub CLI can streamline your development workflow and enhance team collaboration.

***Happy coding!***