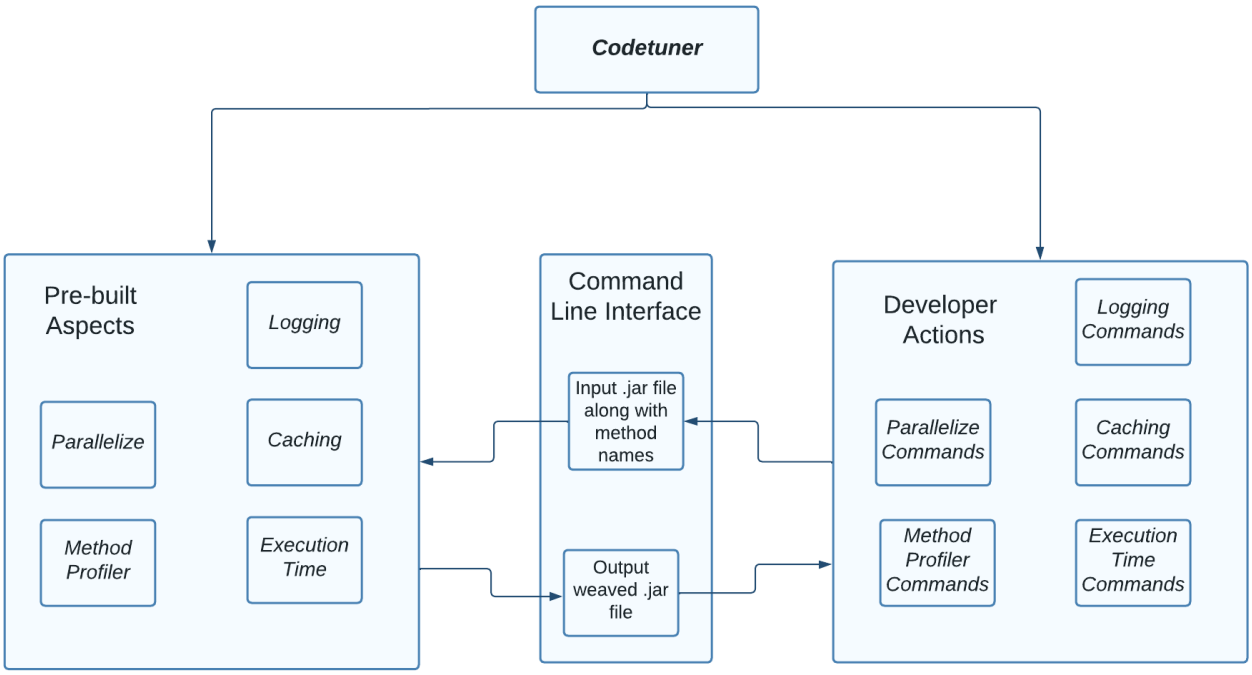
Low-Level-Design-Document.md

Low Level Design Document

1. Overall Architecture Diagram:

The application architecture is shown in the following diagram:



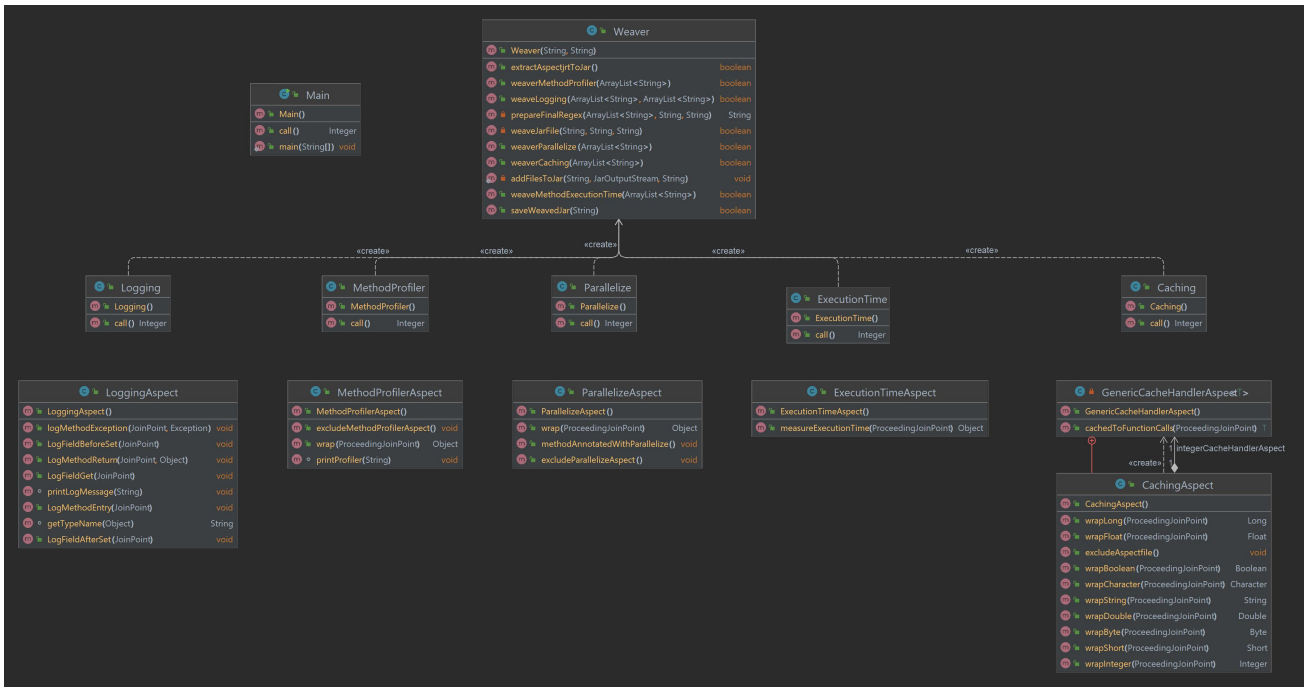
Explanation of the diagram:

- The developer's codebase is compiled into a jar file.
- The developer then enters the path to the jar file into the CLI along with the method names in which they wish to implement the aspects.
- The CLI then accesses the pre-built aspects written in AspectJ to weave the aspects into the developer's codebase at runtime.
- The CLI then outputs the weaved jar file which can be used by the developer.

2. UML Diagrams

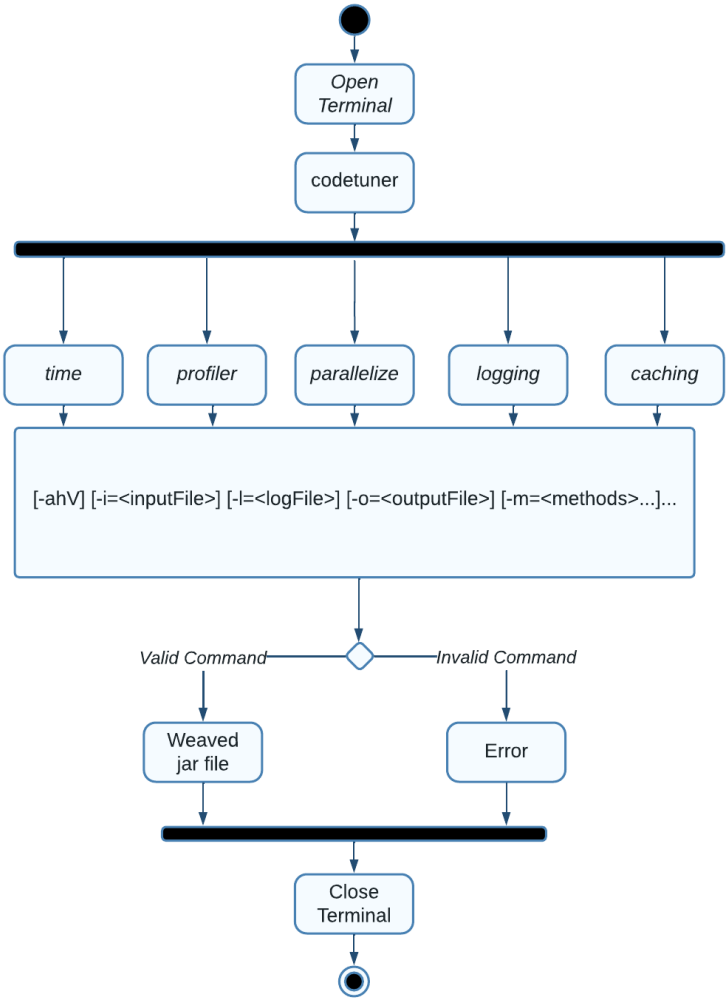
2.1 Static: Class Diagram

Following is the UML Class Diagram:

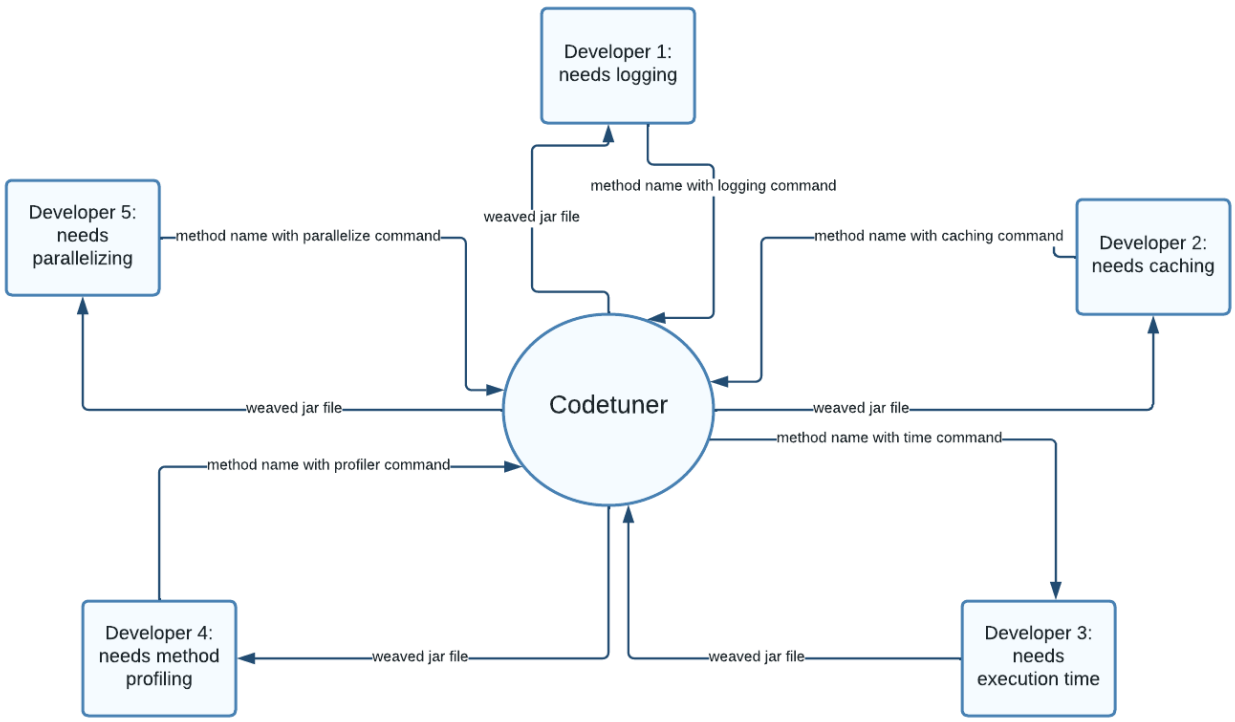


2.2 Dynamic: Activity Diagram

Following is the UML Activity Diagram:



3. Context Diagram



4. AOP Framework Choice: Aspect J

Why we used Aspect J as our AOP framework:

1. Ease of Use: AspectJ is relatively easy to use and can be quickly learned by beginners in AOP. The AspectJ documentation is comprehensive and provides helpful examples and tutorials. Additionally, the syntax and concepts of AspectJ are similar to Java.
2. Runtime Weaving: AspectJ supports runtime weaving, which means that aspects can be applied at runtime without the need to modify the source code. This was the very aim of our project.
3. Wide Range of Join Points: AspectJ supports a wide range of join points, which allowed us to define and apply aspects to various parts of the codebase.
4. Comprehensive Set of Features: AspectJ provides a comprehensive set of features for AOP, including support for caching, measuring execution time, CPU usage, memory usage, and logging.

5. CLI Framework Choice: PicoCLI

Why we used PicoCLI should as our AspectJ project's CLI:

1. Ease of Use: PicoCLI is a user-friendly and easy-to-use CLI framework. It uses annotations to define CLI options and commands, which are intuitive and familiar to Java developers.
2. Integration with Gradle: PicoCLI integrates well with Gradle, which is the build tool we are using. This makes it easy to include the CLI as part of the build process and distribute it as a part of the application.
3. Customization: PicoCLI provides many customization options that allowed us CLI to customize the help messages, command and option behavior, and even the output format.
4. Strong Typing: PicoCLI provides strong typing of CLI arguments, which allowed us to enforce type safety. This makes it easy to catch errors at compile time and avoid runtime errors.
5. Documentation and Support: PicoCLI has comprehensive documentation and strong community support, allowing us to quickly find answers to any questions or issues that arose during development.