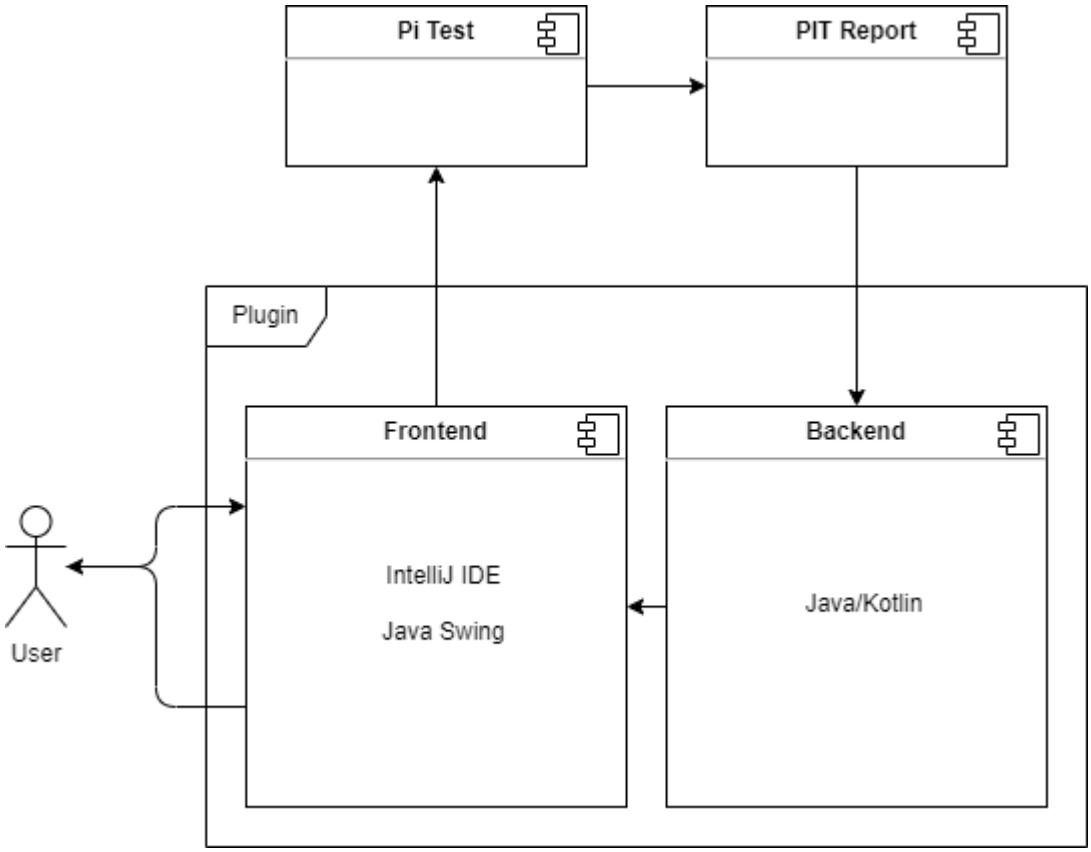
Runtime Components Diagram

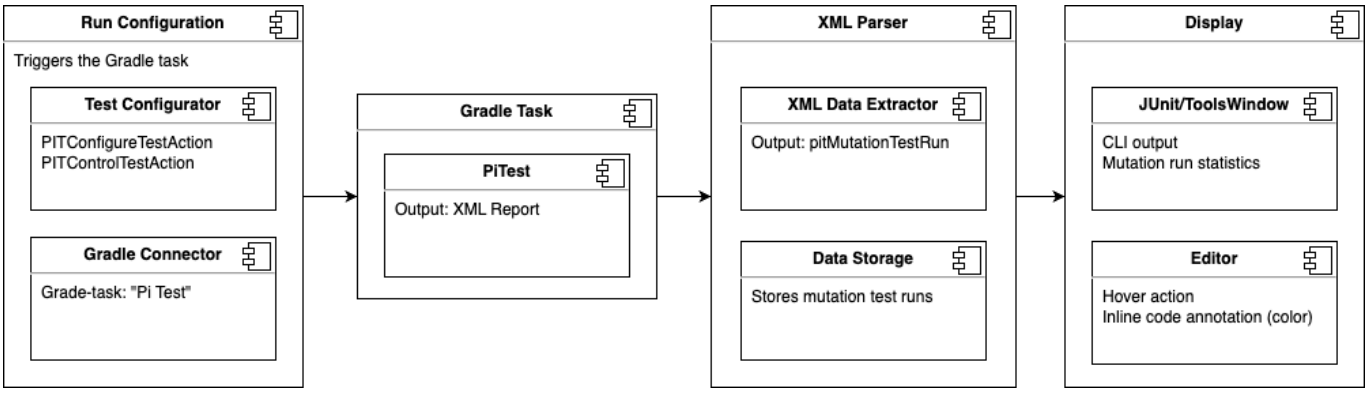


After installing our plugin, users gain the capability to configure mutation tests. This can be achieved by specifying values for verbosity, mutators, targeting tests, and more in a separate window. Additionally, within the code editor, buttons are provided for initiating, rerunning, and stopping the execution of the Pitest runs.

For the plugin implementation, we've opted the IntelliJ Platform Plugin SDK, which offers all the infrastructure that IDEs need to provide rich language tooling support. The SDK utilizes Java Swing for frontend development. The frontend components establish communication with the backend, which is implemented in Java/Kotlin, through Gradle. The choice of Java/Kotlin aligns with the directory structure that is generated by the IntelliJ IDE Plugin generator.

Upon clicking the "start" button, the Gradle-Task "pitest" is executed in the background. Our plugin parses the generated report to present Line Coverage, Mutation Coverage, and Test Strength. Passed tests are depicted in green, while failed tests are highlighted in red. Furthermore, mutation test results for each code line are presented as inline annotations.

Code Components Diagram



The TestConfigurator provides users in the editor with the ability to configure and manage mutation test runs. Our plugin will configure the test runs by editing the corresponding Gradle build file. The GradleConnector class enables us to establish a connection with Gradle, allowing us to execute the Gradle-Task "pitest". This execution will generate an XML report containing the results of the mutation tests. Our XMLParser will extract the necessary information and store it for further processing.

The results will be presented in the ToolsWindow as red and green bars. Additionally, we will print the results on the command line. When a user hovers over a specific code line in the editor, the results for that line will be displayed.

Technology Stack

Programming Language: Kotlin/Java [to be decided upon]

- **Intellij IDE:** The user will interface with the plugin through the Intellij IDEA IDE or Android Studio.
- **Intellij Plugin SDK:** main framework for development of the actual plugin
- **PITest:** The targeted tool is the PIT Mutation Testing system.
- **Gradle:** as the build system
- **Pitest-Gradle/Maven-Plugin:** Setup by the user to integrate running their mutation tests into the build environment
- **Git/Github:** for source code management & collaboration
- **JUnit:** as a test framework
- **Frontend:** The UI for the plugin will be developed using Swing or JavaFX
- **Plugin dependencies:** The developed plugin will have the following dependencies:
    - **GradleConnector:** Allows the plugin to run a gradle task, which in turn will execute the pitest mutation testing. Alternatively Maven Invoker API if maven is used as a build system.