STATIC CODE ANALYZER WEBSERVICE

**Introduction**

The static-code-analyzer-web service is a gating service (used for java projects), which is used to check if a project passes the thresholds for different kinds of code analysis, which can be configured by the user.

The code analyzer comprises of 6 different types of tools, which are-

* Simian Tool, for duplication in code- <https://www.harukizaemon.com/simian/>
* Jacoco, for code coverage- <https://www.eclemma.org/jacoco/>
* Visual code grepper, for security vulnerabilities- <https://sourceforge.net/projects/visualcodegrepp/>
* PMD, for static warnings-<https://sourceforge.net/projects/pmd/>
* Cyvis, for cyclometric complexity- <http://cyvis.sourceforge.net/download.html>
* Junit test time compilation

**Getting Started**

The above tools are being run behind a web service to check if the project complies with the needs of the user or not, and as a result we get a go (if the code passes through all the 6 tool's thresholds) or a no go (if the tool fails to do so).

The project, so far, runs on the localhost. Hence, you can give any of the projects that are present on the localhost (the path to the project, along with the project name, are the inputs taken by the tool), and all the tools work together to give the results, in different forms.

* The tools and the types of output are as follows-
* PMD- a .xml file is generated
* Cyvis- creates a .txt file
* Simian tool- gives the output on the console, as 0 (if no issues) or 1 (if there are duplicates)
* VisualCodeGrepper – creates a .txt file.
* Jacoco – creates a .csv file
* Junit test time compilation – gives the output on the console (time taken to run unit tests).

We extract the data from the files/results that are generated by the tools, and parse them to produce useful data, which is in turn used to generate an output, which can be checked against the thresholds that are set by the user themselves. Hence, a go or a no-go result is generated.

**Project Directory Structure**

Various classes are**:**

**Application (annotation - @SpringBootApplication)**

This is the main class used to run our web service.

**Static Tool Controller(annotation - @Controller)**

This class works as a controller, and as a result directly interacts with the user interface. The Controller has 8 functions/mappings-

* The home page, where the user needs to give the input as the project path, and the project name.
* The tool selection, where the user can select the tool on which they want to run their project on.
* 6 functions, each for one tool, where it interacts with the classes that are tool-specific.

The controller has objects of all the other classes, which are needed by the tool-specific mappings. These Classes are explained further in the document.

We’re using GetMapping as we’re fetching the data from the .jsp pages to run the methods.

**Commands**

This class contains all the commands that are needed to run the tools. All the tools have a static function inside this class, which returns the command, using the arguments passed by the function (the project path and the project name). These commands are then run on the command line using the Process Builder/Process class.

This class also holds the maven commands that are needed to be run on the project before using the tools on the project (some of the tools need the .class files, hence compiling the project at least once is necessary).

Since the class holds only static function, creating an object for it is not necessary.

**StaticToolDao** (Interface)

This interface hold the functions that are needed by the tools to interact with the database.

We’re storing all the data we’re getting after running the tools on a project into a database, so that we can check a static tool run of a project to it’s previous runs, and decide if the newer run is an improvement over the last run or not.

It holds 7 functions, namely insert, getallrows, get, updatecoverage, updatesecurity, updatecomplexity, and updatewarnings; these functions are implemented in the StaticToolDaoImpl class.

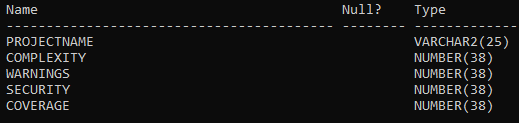
**StaticToolDaoImpl**

The implementation of the StaticToolDao class. Has the functions to insert new data, fetch all the data, fetch according to the project name, and update the single columns of the database (i.e the output of each of the tool, and then update it to the database).

Initially the values for all the column is set as -1, and it gets overwritten as the tool is ran over the project.



***Sample run. At this point, we’ve run complexity and coverage over the project named ExampleWebApp***



***The structure of the database, named as results***

***The insert command (for reference, just copy paste it if you want)-*** *create table results(projectname varchar2(25),complexity int,warnings int, security int,coverage int);*

**Results**

This class is designed to map with the database created to store each run results. Each column in the table corresponds to a single variable in the class where we will have all getter and setter methods for these variables.

**ResultsMapper**

This class implements RowMapper( interface that allows to map a row of the relations with the instance of user-defined class). It iterates the ResultSet internally and adds it into the collection. So wedon't need to write a lot of code to fetch the records as ResultSetExtractor.

**StaticToolService(annotation - @Service)**

This class holds the functions that are (mostly) common for all the tools/are required by all the tools. This contains all the function calls of the methods present in the StaticToolDaoImpl, and these functions are hence called in the Controller methods directly.

* runCommandLineArgument - This function which interacts with the Command prompt, to run the tools on the project. (This gets the argument as the command line for a specific tool (from the static method inside the Commands class), and runs the command to produce some kind of an output, which is needed to be parsed to get useful output.
* getAllClassesWithPackage - This function is used to search all the .class files, as the jacoco tool runs on the .class files itself.
* functionPropertiesFileReader – This function gets the data from the application.properties file, where we have stored the default values for the threshold of all the tools, just in case if the user wants to use the default values in place of providing the value by themselves.

**StaticWarningService(annotation - @Service)**

This class acts as a service for the PMD tool. It contains 2 functions-

* parseXml- this function parses the .xml file that is produced as an output by the tool, and extracts the data as a Nodelist. The xml file has a tag named “file” for each of the classes present in the project, inside which there are multiple “violations” tag, which are formed for every violation present in the class. We’re parsing this, and storing the result in a Nodelist.
* getNoOfwarnings- this function gets the nodelist from the parseXml, and then gets the number of violations in the nodelist. It returns the noofissues as its result.

**ComplexityService(annotation - @Service)**

This class acts as a service for Cyvis complexity tool. It has following functions

* consoleInteractor- This function gets the command for the cyvis tool from the Commands class, and runs it using the Process.runtime function. This is the only class that uses the Process instead of the processBuilder class, because we faced some technical issues if we tried the ProcessBuilder class.

The command makes a .txt file, which contains the details of each of the function in all the classes, and has its complexity next to it.

* extractDetails- This function parses through the text file, gets all the classes and its complexity and stores it into a map (the function-complexity pair), which can be used to get the complexity of each of the function

The .txt file contains string such as this-

**com.philips,StaticToolDao,updatewarnings,1,0,,updatecomplexity,1,0,,updatesecurity,1,0,,updatecoverage,1,0,,get,1,0,,getallrows,1,0,,insert,1,0,**

We split the string by the commas, and hence get the function name and the complexity, and then store it into a map.

* getMaxComplexity- this uses the comparator interface to get the maximum value in the map, which is basically the complexity of the project. This gets stored in the database, and the value needs to be less than the threshold set by the user.

**SecurityService(annotation - @Service)**

This class acts as a service for VisualCodeGrepper security tool. It has one function

* parseTextFile - This function opens the report(text file) that is generated by running visualcodegrepper command and parse it. This finds out at how many lines in the it shows security vulnerabilities,make a count of it and returns the count of vulnerabilities.

**CodeCoverageService (annotation - @Service)**

This class acts a service for Jacoco code coverage tool. It has one function

* parseCSVFile – This function opens the report(CSV file) that is generated by running jacoco command and parse it. At each row, count of instructions covered and instructions missed columns is observed. Finally, all the mathematical computations are done and result code coverage is returned.

**UnitTestService(annotation - @Service)**

This class acts as a service for unit test time compilation. It has two functions

* runCommand – This function is used to run testrun command from the command line using Process Builder and then to get what is output that is printed on the console. So, it uses a BufferedStream and returns a string that is printed on console till it reaches the end of output.
* parseString – This function takes the string obtained from running test run command using runCommand function and parse it to get only the value i,e time taken to compute unit test cases from the console.

**GatingService(annotation - @Service)**

This class acts as a service to gate (compare current result with threshold). It has following functions

* checkforGate – This function can be used for three tools(security tool, warnings tool and complexity tool). If the current result is less than threshold it says go else no-go.
* coverageGate – This function is used by coverage tool. If the current result is more than the threshold, is says go else no-go.
* duplicateGate – This function is used by duplication tool. If the duplicate value is 0, it says go and if it is 1, it says no-go.
* unitTestGate – This function is used for unit test time gating. If flag is true, it says go else no-go.

**Pom.xml**

**POM** is the fundamental unit of work in Maven. It is an XML file that contains information about the project and configuration details used by Maven to build the project. It contains default values for most projects.

**Control Flow for User Understanding**

1. Starting the application using application class.
2. When the user enter the url he/she will be directed to a page where one can give project name and project directory. The total process is done in StaticToolController.
3. In this StaticToolController,all the maven commands are run on the user given maven project and then all the .class files generated which are to be used by some of our tools.
4. Then user has around 6 choices of tools that can run on the given maven project.
5. In each tool, specific set of service functions will be called (runCommandLineargument,ParseTextFile functions and so on) which in turn will be called on StaticToolDao(Repository) class.
6. After that, the results will be taken to a Gating service from where user can get either a go or a no go.

**AutoBuildTest.bat**

This batch file runs all the maven commands on the project along with simian(for duplicates) and jacoco(for code coverage). Then, it runs the spring-boot application.