



Faculty of Engineering - Cairo University Credit Hour System Programs

Communication and Computer Engineering

CCEE

Graduation Project Report

Spring 2022

Measurement of Code Coverage by Black Box Testing of Web-based Applications



ONE Lab

SIEMENS

Prepared by:

Shaimaa Gamal Abostiet

Waed Raed Sabri

Anoud Emad Abdelmoneim

Reem Sameh Mohammed

Loay Samy El Masry

Supervised by:

Prof. Dr. Hassan Mostafa



**CUFE
CCE-E
Credit Hours System**

**Spring 2022
Senior-2 Level
Graduation Project-2
CCEN481**



**CAIRO CODE
COVERAGE**

Graduation Project-2

“Measurement of Code Coverage by Black Box Testing of Web-based Applications”

Final Report

Submitted by:

Shaimaa Gamal Abostiet

Waed Raed Sabri

Anoud Emad Abdelmoneim

Reem Sameh Mohammed

Loay Samy El Masry

Supervised by:

Prof. Dr. Hassan Mostafa

Acknowledgment

Listed below are the names of the people who provided us with significant help in developing our graduation project in addition to our sponsor Siemens EDA. To all we extend our sincere thanks.



Dr. Hassan Mostafa
Dr. Eman El Mandouh
Eng. Haytham Shoukry
Eng. Abdelrahman Saleh
Eng. Ziad Abdelati
Eng. Salma Faheem

Abstract

With the recent developments in software applications, the validation of such applications became an important process to take into consideration. Software testing is a process of ratifying the functionality of software. It is a crucial area which consumes a great deal of time and cost. There are many metrics that can be used to confirm the efficiency and effectiveness of the software testing such as code coverage. Code coverage is a software testing metric that determines the number of lines of code that is successfully validated under a test procedure, which in turn, helps in analysing how comprehensively a software is verified. Furthermore, we can use such code coverage data to optimize the test cases to make it more effective.

This project aims to measure the code coverage data by executing black box testing on a web application. Our approach was to calculate the code coverage data for the frontend and backend separately, by executing test cases using a test automation tool. The raw data collected from the test cases would then be formulated into readable html reports.

The test suite size tends to increase by including new test cases due to software evolution. Consequently, the entire test suite cannot be executed considering budget and time limitations. Which is why another objective was to utilize the code coverage data in reducing large test suites. We did so by applying some machine learning algorithms.

Table of Contents

Acknowledgment	2
Abstract	3
List of Figures	7
List of Tables	8
1. Introduction.....	9
1.1. Project Objective	9
1.2. Project phases.....	9
1.3. Background	10
1.4. Time plan.....	11
2. Building and testing a prototype Web Application.....	11
2.1. Backend.....	12
2.2. Frontend	13
2.3. Test case automation	15
3. Code coverage research and implementation	15
3.1. Back-end Code Coverage.....	16
3.1.1. Proposed approaches.....	16
3.1.2. Best-fit approach implementation.....	16
3.1.3. Requirements	18
3.2. Front-End Code Coverage.....	19
3.2.1. ngWebdriver	19
3.2.2. Unit testing.....	20
3.2.3. Istanbul (NYC).....	20
3.2.4. Webpack challenges.....	22
3.2.5. Remap Istanbul approach.....	24
4. Merging code coverage results	25
4.1. Backend merging steps.....	25
4.2. Frontend merging steps	27
5. Deployment on a Real Case Web Application	29
5.1. Applying the Code Coverage Measurement	29
5.1.1. Introduction.....	29
5.1.2. VIQ Workshops	30
5.1.3. Adjustments needed.....	31

5.1.4.	Deployment of Back-End code coverage measurement to the Real Case application	32
5.1.5.	Deployment of Front-End code coverage measurement to the Real Case application	34
5.1.6.	Edited Files	35
5.1.7.	Summary for Deployment on Real Case Web Application	36
5.2.	Running a full regression	36
5.2.1.	Preparation steps for running a full regression	36
5.2.2.	Processing Data for Test Cases	37
5.2.3.	Useful hints for processing data.....	37
5.2.4.	Processing Data for the Regression	39
5.2.5.	Results of running a full Regression on Coverage Analyzer using Regression Launcher Tool.....	40
5.3.	Final Package	41
5.4.	Challenges in Deployment Process	42
5.5.	Formatting the results for usage in applications.....	43
6.	Applications of Code Coverage	44
6.1.	Introduction to Code Coverage Applications.....	44
6.2.	Applications Using Code Coverage Results	46
6.2.1.	Selection and Prioritization.....	46
6.2.2.	Exposure of Unused Code	49
6.2.3.	Test Case Size Minimization	49
6.2.4.	Test Suite Reduction (TSR).....	49
6.3.	Test Suite Reduction Implementation Using Code Coverage & Machine Learning	50
6.3.1.	Introduction.....	50
6.3.2.	K – Mean Clustering Algorithm	54
6.3.3.	Greedy Algorithm	63
6.4.	Future work and conclusion	69
7.	Scalability of the Tool.....	70
7.1.	Frontend Scalability	70
7.2.	Backend Scalability.....	70
7.3.	Test Automation Tool	71
7.4.	Test Suite Reduction	71
8.	Overhead Calculations	71
8.1.	Disk Space.....	71

8.2. Build Time.....	71
8.3. Memory Consumption.....	72
8.4. Runtime	72
Conclusion	73
References.....	74
Appendix.....	77
Appendix A: Code	77
Appendix B: Licenses	99

List of Figures

FIGURE 1: TIME PLAN.....	11
FIGURE 2: API TESTING USING POSTMAN	13
FIGURE 3: FRONTEND API DESIGN	14
FIGURE 4: EMPLOYEE MANAGER WEB APPLICATION.....	15
FIGURE 5: LCOV - CODE COVERAGE REPORT	16
FIGURE 6: BUILDING JAR FILE USING INTELLIJ	17
FIGURE 7: EMPLOYEE MANAGER BACKEND COVERAGE REPORT USING JACOCO	18
FIGURE 8: EMPLOYEE MANAGER BACKEND COVERAGE REPORT - HITS & MISSES.....	18
FIGURE 9: NGWEBDRIVER TEST CASE SNIPPET.....	20
FIGURE 10: FRONTEND CODE SNIPPET FOR DUMPING COVERAGE DATA	21
FIGURE 11: EMPLOYEE MANAGER FRONTEND COVERAGE REPORT	22
FIGURE 12: EMPLOYEE MANAGER FRONTEND COVERAGE REPORT ERROR	22
FIGURE 13: WEBPACK INSTALLATION PROCESS	24
FIGURE 14: EMPLOYEE MANAGER BACKEND UNMERGED REPORTS	27
FIGURE 15: EMPLOYEE MANAGER BACKEND MERGED REPORT	27
FIGURE 16: EMPLOYEE MANAGER FRONTEND - ADD & DELETE FUNCTIONS HIT IN MERGED REPORT	28
FIGURE 17: EMPLOYEE MANAGER FRONTEND – DELETE FUNCTION HIT	28
FIGURE 18: EMPLOYEE MANAGER FRONTEND – ADD FUNCTION HIT	29
FIGURE 19: COVERAGE ANALYZER UI	30
FIGURE 20: GRADLE VS MAVEN	31
FIGURE 21: ADDED PLUGINS FOR GRADLE.....	32
FIGURE 22: COVERAGE ANALYZER BACKEND COVERAGE REPORT FOR A SINGLE TEST CASE	33
FIGURE 23: COVERAGE ANALYZER BACKEND COVERAGE REPORT FOR A SINGLE TEST CASE	33
FIGURE 24: COVERAGE ANALYZER BACKEND COVERAGE REPORT FOR A SINGLE TEST CASE - HITS & MISSES	33
FIGURE 25: DUMPCOVERAGE FUNCTION IN COMMON CLASS.....	35
FIGURE 26: FRONTEND MERGING FILE.....	39
FIGURE 27: BACKEND MERGING FILE	40
FIGURE 28: REGRESSION SUITE LAUNCHER TOOL RESULTS	40
FIGURE 29: COVERAGE ANALYZER - FULL REGRESSION BACKEND COVERAGE REPORT	41
FIGURE 30: COVERAGE ANALYZER - FULL REGRESSION FRONTEND COVERAGE REPORT.....	41
FIGURE 31: RESULTS OF MERGING SCRIPT	41
FIGURE 32: HTML NAVIGATOR/REPORT	42
FIGURE 33: VIQ CODE COVERAGE REPORT	42
FIGURE 34: BUILDNOPRODCOV TASK	43
FIGURE 35: APPLICATIONS USING TESTING METRICS	46
FIGURE 36: CLUSTERING ILLUSTRATION	54
FIGURE 37: K-MEAN PROCESS FLOW.....	56
FIGURE 38: VIQ BACKEND COVERAGE REPORT TO EXTRACT DATA	57
FIGURE 39: COVERAGE REPORT SOURCE PAGE TO EXTRACT TOTAL RESULTS	58
FIGURE 40: EXTRACTED DATA IN EXCEL FILE FORMAT.....	59
FIGURE 41: DESCRIPTIVE ANALYSIS TO COMPUTE Z-SCORES FOR K-MEAN ANALYSIS IN SPSS	60
FIGURE 42: RESULTS OF DESCRIPTIVE ANALYSIS.....	60
FIGURE 43: K-MEAN CLUSTER ANALYSIS IN SPSS.....	61
FIGURE 44: K-MEAN CLUSTER ANALYSIS IN SPSS.....	61
FIGURE 45: BAR GRAPH SHOWING CLUSTERING CRITERIA	61
FIGURE 46: RESULTS OF K-MEAN ANALYSIS - DISTANCE FROM CLUSTER CENTER & CLUSTER MEMBERSHIP	62
FIGURE 47: SAMPLE OF REDUNDANT TEST CASES	63
FIGURE 48: REQUIREMENT LIST	66
FIGURE 49: INPUT TO SETCOVER GREEDY ALGORITHM.....	67
FIGURE 50: SNIPPET FROM THE OUTPUT OF SETCOVER.....	67

FIGURE 51: SNIPPET FROM THE REDUCED LIST	68
FIGURE 52: BACKEND COVERAGE AFTER SETCOVER GREEDY ALGORITHM	68
FIGURE 54: FILES INCLUDED IN REACT.....	70

List of Tables

TABLE 3: MACHINE LEARNING ALGORITHMS FOR TEST SUITE REDUCTION ^[33]	51
TABLE 4: SUMMARY OF COVERAGE RESULTS	68
TABLE 5: COMPARISON BETWEEN GREEDY & K MEAN CLUSTERING	69
TABLE 6: MEMORY CONSUMPTION COMPARISON	72

1. Introduction

1.1. Project Objective

We were required to build a framework that obtains the code coverage for a full stack web application using automated test cases. Our aim is to create a tool “**Cairo Code Coverage**” which achieves this objective. One of the major challenges facing us is how to measure the code coverage with built files. Furthermore, look into the applications of code coverage results.

1.2. Project phases

In this section we will be discussing briefly the project phases. We were required to create a tool that calculates the code coverage for a web application, and then try it on one of Siemens EDA web applications.

Phase 1

In this phase we researched on the topic of building a single page web application, then created a web application which will be used to emulate the actual web app that we will test later on in the deployment. We created the backend using Java with Gradle^[24]/Maven^[18], then for the frontend we used Angular 8^[3].

Phase 2

Now that the web application is ready, we started getting familiar with the concept of test automation tools. Our tool of choice, which was the one used by the company, was Selenium^[7]. We started writing several, simple test cases for the web application. These test cases will generate some raw data which will be the input for our tool later on so that it can process it and produce readable coverage reports.

Phase 3

We started our code coverage research and implementation for both frontend and backend with different tools.

Phase 4

After creating the fully functioning tool, we started working on a real case application, deploying changes on the build process and generating code coverage reports.

Phase 5

Analysis of our results and post processing to further understand how we can utilize the code coverage data to enhance the test suites.

1.3. Background

In this project we focused on measuring code coverage for web application. Web applications has become an essential business tool that enables companies to communicate with customers, collaborate with employees, store vast volumes of data more effectively, and provide better information management. Web based applications provide user collaboration and sharing, only web browser is needed for the client and an only one powerful hardware for the server, and it works in the cloud. Furthermore, it offers platform independence meaning browser can be invoked on Linux, Windows, etc. However, web based applications have some challenges such as data safety and browser compatibility. This illustrates how web based applications are excelling over desktop application since desktop application are single user, machine dependent which necessitates specific hardware and software requirements on machine to be able to host the application, however it provides data security because it is completely isolated.

Another concept that we will be aiming the testing on is Single Page Applications (SPAs). Most recently, many of the web applications are tending to use SPAs rather than Multiple Page Applications (MPAs). On demand, SPAs reloads only the data necessary for the user using JavaScript for dynamic rewriting, therefore code resources like HTML, CSS, and scripts don't need to be loaded with every interaction resulting in having higher speed/performance than the MPAs where the entire web page content is refreshed. Moreover, SPAs Clear separation between Front-End (UI), and the Back-End (Application Core), which in our case is an advantage to be able to obtain code coverage data on both ends separately.

We need to get familiar with the concept of regression testing as well which is used by many large software companies. Regression testing is a common maintenance procedure for revalidating changed software. As software is modified and new test cases are added to the test-suite to test new or changed requirements or to maintain test-suite adequacy, the size and complexity of software systems is growing dramatically. In addition to this, the existence of automated tools, such as Selenium, has led to the generation of a huge number of test cases, the execution of which causes huge losses in cost and time.

The Code coverage data will be collected through testing the single page Web application therefore we need to get familiar with the type of testing that can be performed. Testing can be classified into 3 main types, White Box Testing, Grey Box Testing, and Black Box testing. The White Box Testing is executed as a part of the application build step. For example, the Java coded tests JUnit,

it mainly aims at testing the Backend only. The Grey Box Testing can be done after building the application without having the UI. It can be tested by calling the same Application Programming Interfaces (APIs) executed by the UI interactions and it returns JSON responses indicating whether it was successful or not. Lastly, the Black Box Testing, which mimic the end user actions by simulating the interactions with the GUI on the browser, this type of testing can be automated using different automation frameworks/tools.

Once we figure out the measurement of code coverage we will apply it on a real case application as a case study. The application we will be testing it on is Questa Verification IQ (VIQ). VIQ is a web-based platform of applications having to do with visualizing and managing verification data coming from simulation, emulation or Formal Verification of Hardware designs.

1.4. Time plan

In order to keep track of our work, we created this simple timeline for our project. It is to be noted that we started documenting our thesis before June, we made sure to keep track of our progress.

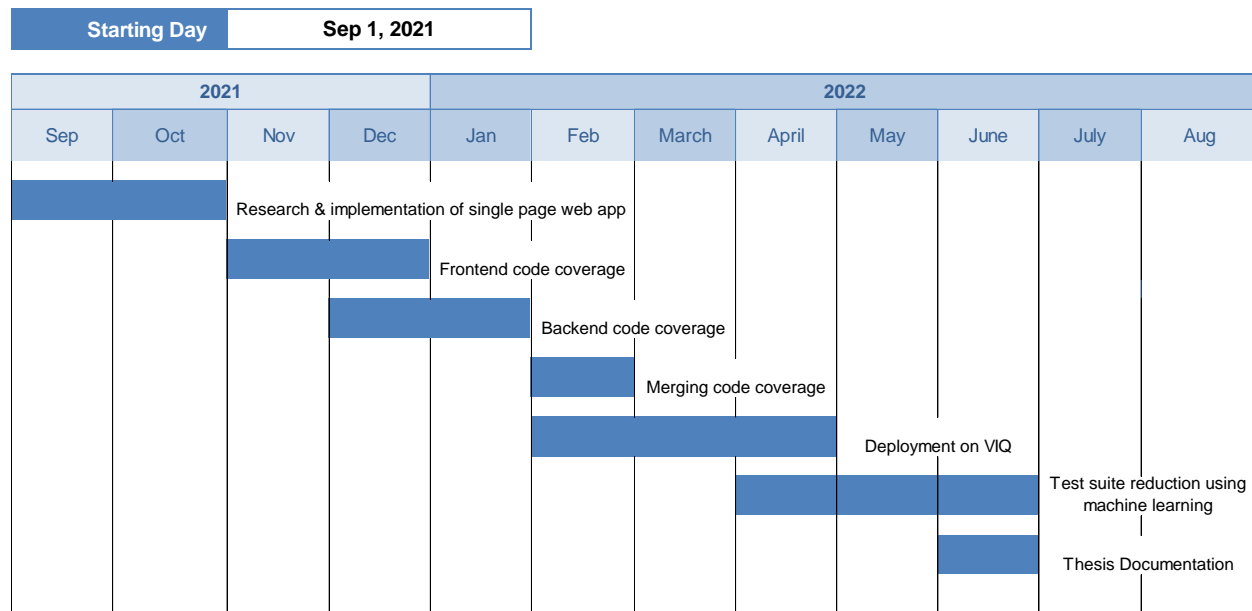


Figure 1: Time Plan

2. Building and testing a prototype Web Application

In this step we are required to build a single page web application using Angular 8 in the frontend and Java in the backend. In the beginning we did not have a web application to use for the testing, furthermore, in order to fully understand the concept of web applications and how what to test, etc.

we wanted to follow the steps from the beginning to build the web application. We found an online tutorial that shows a step by step process to create an Angular 8 frontend and Java with Maven backend web application. This web application will be used to perform the selenium test cases on. During this phase we took some time to learn about web development, as we did not have much background in it.

2.1. Backend

In order to get familiar with the concept of how to build a backend for a web application we started by looking into Node.js^[5] which is an open source, cross-platform, backend, JavaScript runtime environment used to run the backend server. It's one of the simplest environments to deal with, especially if you are just getting started with web development. We looked at a backend example based on Node.js in order to get familiar with the idea.

Generally, the API design for backend is divided into three stages, the first stage is receiving the HTTP requests by the client then it is handled by the controller which calls the service stage that can access the database directly through configuring the properties in the application.properties file.

After getting familiar with building a backend server, we started the implementation by bootstrapping the backend server using Spring Initializr^[1] that was used to form the structure and inject dependencies such as JPA repo and MySQL, the implementation can be found in Appendix A: Backend Code.

During the implementation of the backend server, we used Postman^[36], which is an API testing tool. This tool allows us to test the server by sending requests and verifying them before starting the implementation of the backend. Furthermore, we got familiar with Httpie^[6], which is another API testing tool that uses the command line interface directly.

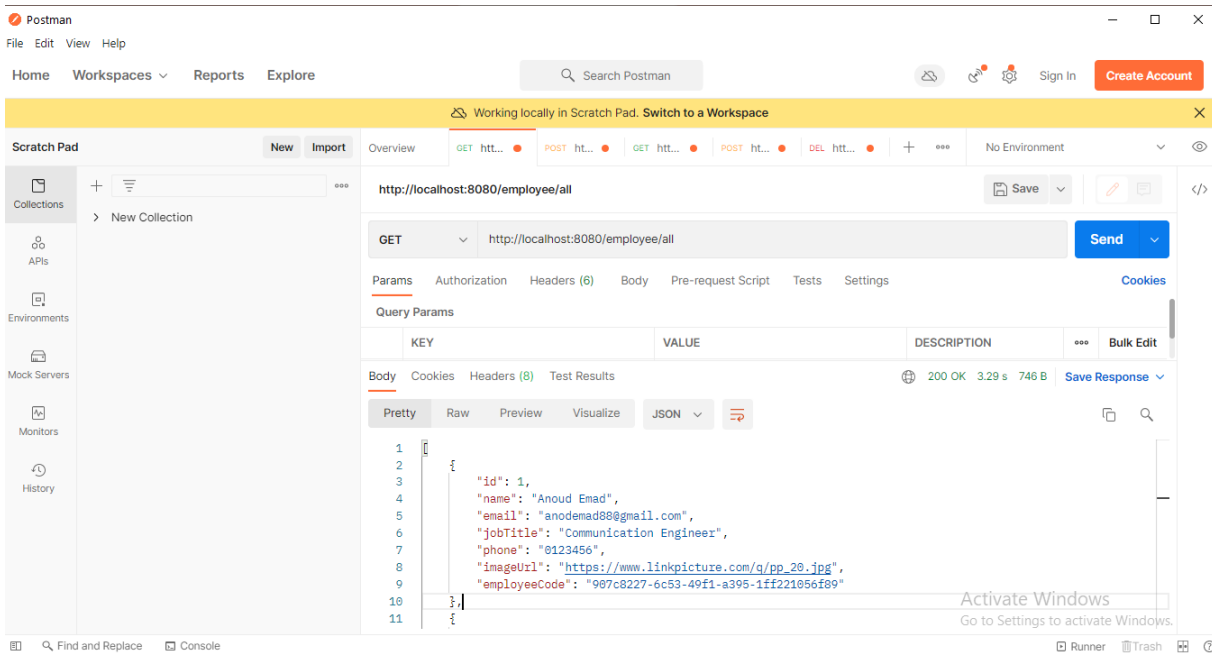


Figure 2: API Testing Using Postman

2.2. Frontend

In the previous phase we assumed the requests were sent by an unknown client. Throughout this upcoming phase we will implement this client. We used Angular framework to develop our web application. Angular is a framework and platform for building single page client applications. It is written in Typescript ^[2]. The architecture of an Angular application relies on certain fundamental concepts. The basic building blocks are angular components that are organized into NgModules^[4]. An Ngmodule can associate its components with related code, such as services, to form functional units. These components define views, which are sets of screen elements.

Next, we will dive into API design using Angular's components, forms and services. The flow begins with the users' actions using the UI this triggers angular events that are handled by the functions, included in the component, these functions call the services that construct the http requests to be then handled by the backend as mentioned previously.

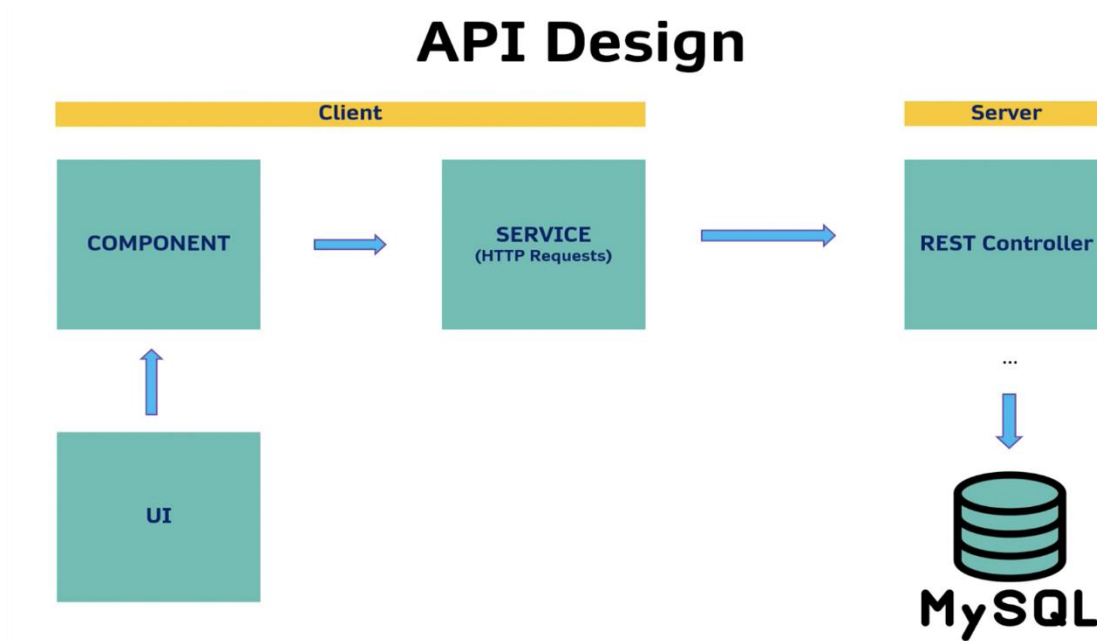


Figure 3: Frontend API Design

Cors configuration^[10] is done to give some permissions such as

- Allowing access to the servers and clients to be able to send requests
- Configure the headers of the requests
- Set allowed methods (GET, POST, PUT, DELETE)

A CORS configuration is a document that defines rules that identify the origins that you will allow to access your bucket, the operations (HTTP methods) supported for each origin, and other operation-specific information. The CORS configuration must be a JSON document.

After the configuration we start looking into how to create the GUI. In Angular, a template is a section of HTML. It renders a user interface to include as a part of the page that the browser displays. When, we built our Angular application the `app.component.html` file is the default template containing placeholder HTML. We extended this HTML with special angular syntax in our templates.

The following figure shows the final structure of our web application GUI. As we can see in the GUI we added a button for add, delete, update functionalities of our application, as well as a search bar for the search functionality.

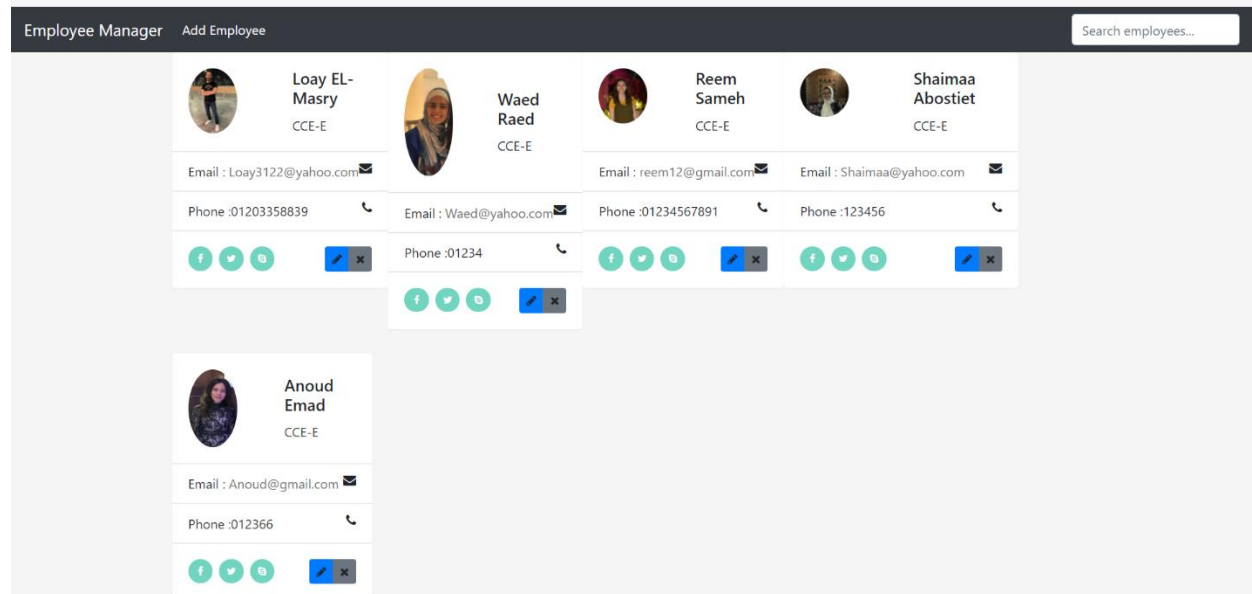


Figure 4: Employee Manager Web Application

2.3. Test case automation

In this step we are required to automate the testing of our application's GUI. Selenium^[7] is one of the most commonly used automation testing tools. It simulates, using selenium web driver, the user actions by locating elements using locator techniques and take action on the located elements such as a click and send keys. It can also preform navigation commands.

The reason for choosing Selenium as our test automation tool is because we wanted to follow the same structure as the one Siemens EDA had in VIQ web applications, this would make the deployment phase simpler.

We created several test case for each of the functionalities we have which are: Add, Delete, Update and Search.

3. Code coverage research and implementation

We started by doing research on code coverage and how to use it to measure the quality of testing. Firstly, we came across LCOV^[17] that helped us to visualize the end result of our coverage report.

LCOV is a graphical front-end for GCC's^[16] coverage testing tool gcov^[22]. It collects gcov data for multiple source files and creates HTML pages containing the source code annotated with coverage information. It also adds overview pages for easy navigation within the file structure. LCOV supports statement, function and branch coverage measurement. It generates a code coverage report as shown in the following figure:

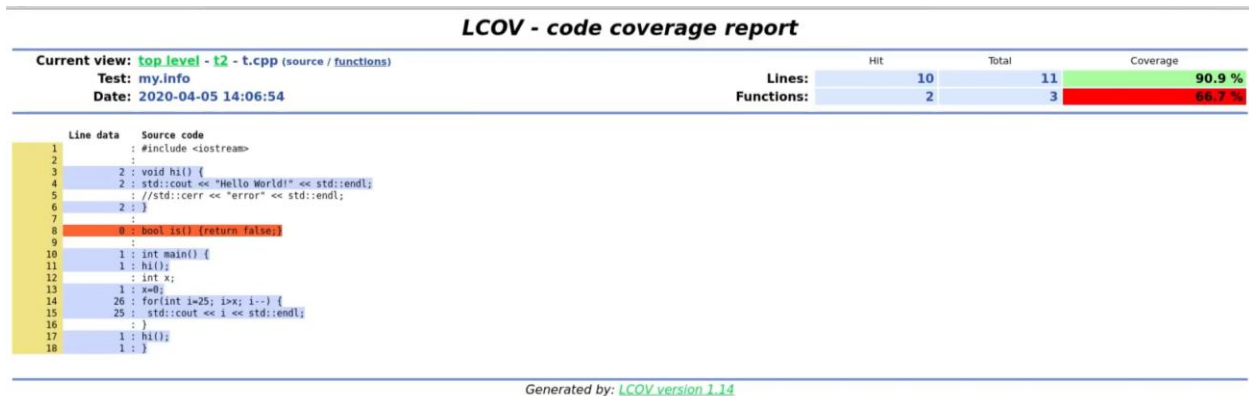


Figure 5: LCOV - Code Coverage Report

We found that in most code coverage tools you have access to the full source code, compiled files, etc. However, in our case we only have access to the UI therefore, we must use selenium test cases.

While conducting our research, the vision we had was that the results of our selenium test cases will be formulated into files from which we can extract the coverage data. Then, we can use the data to make the output code coverage report.

3.1. Back-end Code Coverage

3.1.1. Proposed approaches

We thought of three approaches when searching for backend code coverage. The best-case scenario was to not need any modification on source files or build files and collect coverage data through API requests. However, we didn't find any resources regarding this approach. The second approach we thought of was to instrument the build files similar to what we did in the front-end. We found several possible tracks, including EMMA^[23]/JaCoCo^{[29][30]}, they instrument .jar files and collect coverage data in a .exec file. We also found Codeception^[20] which eases remote testing. The third approach is the worst-case scenario in which we will need to modify the source files to collect coverage data using Maven plugins^[31].

3.1.2. Best-fit approach implementation

After analysing the previous three approaches and checking their resources, we found that the second approach is the best one. This is due to the fact that most of the applications already work with build files or Web application Archive (war) files, as well as having enough resources to pursue this method. Here are the steps of this approach:

1. Building Jar file using IntelliJ:

We started by creating a jar artifact with options “From modules with dependencies” and “copy to the output directory and link via manifest” selected. This first option is needed to include the dependencies in the build files and the second option is to avoid errors during instrumentation of build files.

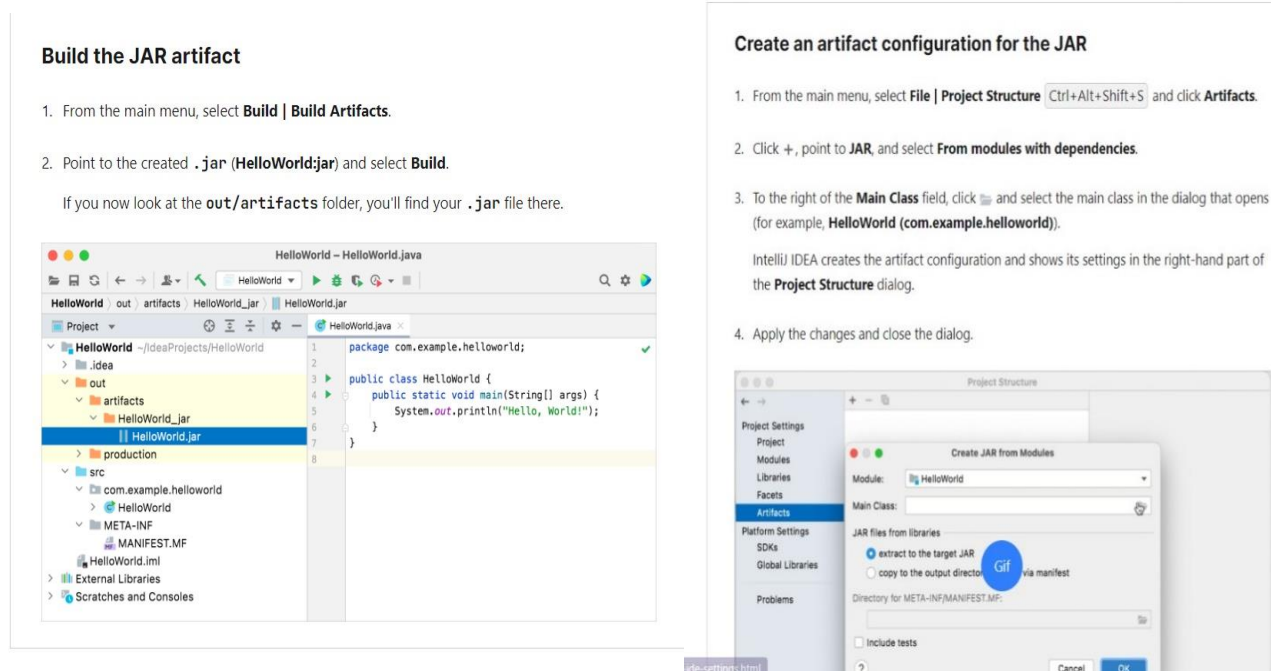


Figure 6: Building JAR File Using IntelliJ

2. Instrument and run Jar file using Jacoco:

We used the command:

```
java -javaagent:jars/org.jacoco.agent-0.8.7-runtime.jar -jar
employeeanagerjar2/employeeanager.jar
```

3. Run test cases

4. Generate HTML Report from jacoco.exec file

We used the command:

```
java -jar jars/org.jacoco.cli-0.8.7-nodeps.jar report jacoco.exec --classfiles=target/classes --
html coverage
```

The following figures show the coverage html reports resulting from the previous steps. The Missed instructions column refers to the java byte code instructions missed.

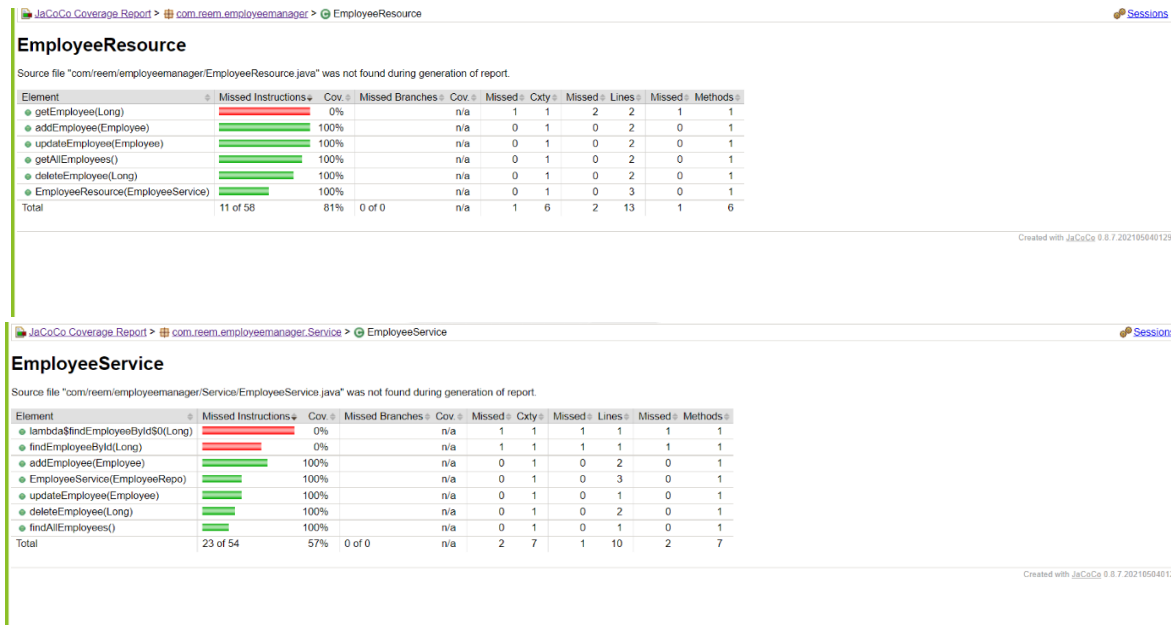


Figure 7: Employee Manager Backend Coverage Report Using JaCoCo

3.1.3. Requirements

In order to get the source files annotations of the hit and missed functions, a path to a copy of the source files is needed as we didn't find an option to include mapped source files like the case of the front-end. This does not defy our main goal as we are still collecting coverage using the built files. Also, java agents need to be installed with a version compatible with the java version of the project.

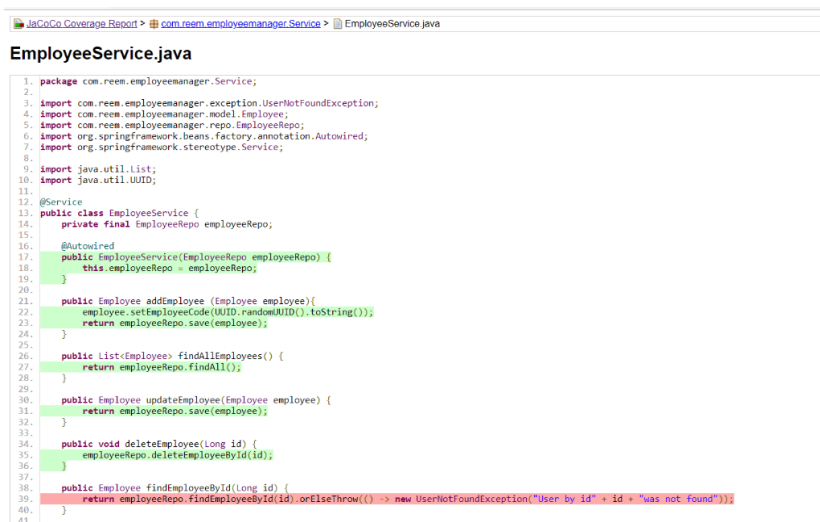


Figure 8: Employee Manager Backend Coverage Report - Hits & Misses

3.2. Front-End Code Coverage

In this section we will be discussing research and the several approaches we took in order to get the code coverage results through executing the Selenium test cases.

3.2.1. ngWebdriver

ngWebDriver^[9] is a small library of WebDriver locators and more for AngularJS (v1.x) and Angular (v2 through v9), for Java. It works with Firefox, Chrome and all the other Selenium-WebDriver browsers.

We have seen how we can use selenium webdriver and java to test angular JavaScript controls, but there are few limitations like we need to stick to a particular locator strategy (xpath or css) as selenium itself do not have locators method specific to angular controls (like ngbinding, ngrepeater etc), and sometime angular element's actions lag behind the selenium line by line execution (synchronisation issue).

To overcome above said issues while testing angular js controls with selenium and java, a library known as ngWebDriver came out to add features to selenium and can write the scripts in java (so that we need not to switch or learn any other languages like javascript for protractor)

ngwebdriver basically taken the advantage of protractor and passing the javascript to browser to handle angular controls and also allows to write scripts in java language without any synchronisation issue.

Let's list our few angular controls that are different than normal html elements:

- ng-model
- ng-binding
- ng-repeat

If your application has only the above attributes allocated for the browser elements, ngwebdriver has capability to identify the above elements directly using:

- `byAngular.binding()`
- `byAngular.model()`
- `byAngular.options()`
- `byAngular.repeater()`

Note – Along with above angular specific locators, we even can use selenium specific locators like id, name, className, css selector etc.

The following figure shows a code snippet of the test case we wrote using ngWebDriver method.

```
NgWebDriver ngdriver= new NgWebDriver (jsDriver);

driver.manage().deleteAllCookies();
driver.manage().window().maximize();
driver.manage().timeouts().implicitlyWait(30, TimeUnit.SECONDS);
driver.manage().timeouts().pageLoadTimeout(30, TimeUnit.SECONDS);
driver.get("http://localhost:3000/");
ngdriver.waitForAngularRequestsToFinish();
Thread.sleep(1000);
System.out.print("===== Source code as follows =====\n");
System.out.print( driver.getPageSource()+"\n"); // --> File (splitting 10 functions)
System.out.print("===== Source code ended =====\n");
System.out.print("===== Element Source code as follows =====\n");
System.out.print( driver.findElement(By.className("nav-link")).getAttribute("outerHTML")+"\n");
System.out.print("===== Source code ended =====\n");
// WebElement element = driver.findElement(By.className("nav-link"));
/*System.out.print(jsDriver.executeScript("var items = {}; "
+ "for (index = 0; index < arguments[0].attributes.length; ++index) "
+ "{ items[arguments[0].attributes[index].name] = arguments[0].attributes[index].value }; "
+ "return items; ", element));*/
// System.out.print(jsDriver.executeScript("var item = arguments[0].attributes[2]; return item; ", element));
WebElement employee = driver.findElement(ByAngular.binding("employee?.name"));
// ng repeat (for)
// System.out.print(employee.toString());
```

Figure 9: ngWebDriver Test Case Snippet

Why not use ngWebDriver in our tests?

ngWebDriver won't work because we don't have access to the source code (i.e. bindings, etc.) Therefore, our test cases will follow the same template as the Siemens EDA test automation team's test cases of the application, meaning we will use WebDriver and the same algorithms we implemented for our test cases.

3.2.2. Unit testing

The first idea that came to our minds was to make use of ng-test with option `--code-coverage` which uses the test cases inside `.spec.ts` files. However, this method would require us to re-write the existing Java Selenium test cases using JavaScript and integrate them with Jasmine framework which is not the most desirable conclusion for us. Therefore, we tried to find another method that would perform the same functionalities as ng-test `--code-coverage`.

3.2.3. Istanbul (NYC)

Upon our search for an alternative method, we came across a framework called Istanbul^[12]. Istanbul instruments your ES5 and ES2015+^[25] JavaScript code with line counters, so that you can track how well your unit-tests exercise your codebase. It has a command client called `nyc`^[12] which works well with many testing frameworks.

Using Istanbul with Selenium to Get Code Coverage:

First we build our project to get the build files that we will work on later. The option `source-map` output source maps for scripts and styles in addition to the build files.

Source mapping writes the source files in a compressed format.

```
ng build --source-map
```

Now, we instrument our build files using nyc. The following command modifies the built files in order to record coverage data while the selenium test cases are running, and then it saves said data in a variable called window.coverage. This is useless unless you actually do something with that data.

- --exclude-after-remap=false: because source map from the ng build will map all the files, however, in nyc there is an option to exclude specific files. So we need to exclude after remap
- --all: Needed with the --source-map option to get the source files

```
nyc instrument dist/myapp dist/myapp --exclude-after-remap=false --complete-copy --in-place --all
```

Serving the application using lite-server, which is recommended by Angular, to be able to run our test cases.

```
lite-server --baseDir="dist/myapp"
```

After the previous step, our web application will open. Now we can go ahead and run our test cases. One of the problems we faced was not finding the window.coverage variable at first, so instead, we took an easier approach which is writing a code for the process of getting the data saved in window.coverage and saving it in a .json file that will be used later to generate the coverage report.

```
JavascriptExecutor js = (JavascriptExecutor) driver;  
Object str = js.executeScript("return window.__coverage__");  
  
GsonBuilder builder = new GsonBuilder();  
Gson gson = builder.create();  
  
String coverage = gson.toJson(str);  
Files.write(Paths.get("C:\\Users\\Shaimaa Abostiet\\employeemanagerapp\\coverage\\coverage.json"), coverage.getBytes());
```

Figure 10: Frontend Code Snippet for Dumping Coverage Data

Finally, we can generate the code coverage report using the .json file that we generated.

```
nyc report --reporter html -t coverage --report-dir coverage/coverage.json
```

All files src/app
 60.52% Statements 23/38 54.54% Branches 12/22 60.71% Functions 12/28 58.06% Lines 18/31
 Press n or j to go to the next uncovered block, b, p or k for the previous block.

File	Statements	Branches	Functions	Lines
app-routing.module.ts	100%	3/3	100%	2/2
app.component.ts	50%	13/26	37.5%	6/16
app.module.ts	100%	2/2	100%	2/2
employee.service.ts	71.42%	5/7	100%	2/2

Figure 11: Employee Manager Frontend Coverage Report

At this point we have successfully generated a code coverage report for all our application’s source files. However, we still couldn’t remap the source maps to the original source code. Our goal now is to configure webpack so that it can remap the code files correctly, so that the nyc report command can get the hits and misses within the code.

All files / src/app employee.service.ts
 71.42% Statements 5/7 100% Branches 2/2 66.66% Functions 4/6 80% Lines 4/5
 Press n or j to go to the next uncovered block, b, p or k for the previous block.

```

1 | 0 | Unable to lookup source: C:\Users\Shaimaa Abostiet\employeemanagerapp\dist\employeemanagerapp\webpack:\src\app\employee.service.ts (ENOENT: no such file or directory, open 'C:\Users\Shaimaa Abostiet\employeemanagerapp\dist\employeemanagerapp\webpack:\src\app\employee.service.ts')
2 | 0 | Error: Unable to lookup source: C:\Users\Shaimaa Abostiet\employeemanagerapp\dist\employeemanagerapp\webpack:\src\app\employee.service.ts (ENOENT: no such file or directory, open 'C:\Users\Shaimaa Abostiet\employeemanagerapp\dist\employeemanagerapp\webpack:\src\app\employee.service.ts')
3 | 0 |   at Context.defaultSourceLookup [as sourceFinder] (C:\Users\Shaimaa Abostiet\AppData\Roaming\npm\node_modules\nyc\node_modules\istanbul-lib-report\lib\context.js:17:15)
4 | 0 |   at Context.getSource (C:\Users\Shaimaa Abostiet\AppData\Roaming\npm\node_modules\nyc\node_modules\istanbul-lib-report\lib\context.js:71:21)
5 | 0 |   at annotateSourceCode (C:\Users\Shaimaa Abostiet\AppData\Roaming\npm\node_modules\nyc\node_modules\istanbul-reports\lib\html\annotator.js:216:40)
6 | 0 |   at HtmlReport.onDetail (C:\Users\Shaimaa Abostiet\AppData\Roaming\npm\node_modules\nyc\node_modules\istanbul-reports\lib\html\index.js:409:33)
7 | 0 |   at Visitor.value (C:\Users\Shaimaa Abostiet\AppData\Roaming\npm\node_modules\nyc\node_modules\istanbul-lib-report\lib\tree.js:38:38)
8 | 0 |   at ReportNode.visit (C:\Users\Shaimaa Abostiet\AppData\Roaming\npm\node_modules\nyc\node_modules\istanbul-lib-report\lib\tree.js:88:21)
9 | 0 |   at C:\Users\Shaimaa Abostiet\AppData\Roaming\npm\node_modules\nyc\node_modules\istanbul-lib-report\lib\tree.js:92:19
10 | 0 |   at Array.forEach (C:\Users\Shaimaa Abostiet\AppData\Roaming\npm\node_modules\nyc\node_modules\istanbul-lib-report\lib\tree.js:91:28)
11 | 0 |   at ReportNode.visit (C:\Users\Shaimaa Abostiet\AppData\Roaming\npm\node_modules\nyc\node_modules\istanbul-lib-report\lib\tree.js:92:19)
12 | 0 |   at C:\Users\Shaimaa Abostiet\AppData\Roaming\npm\node_modules\nyc\node_modules\istanbul-lib-report\lib\tree.js:92:19
    
```

Figure 12: Employee Manager Frontend Coverage Report Error

3.2.4. Webpack challenges

Webpack^[26] is a static module bundler for JavaScript applications. It takes modules, whether custom files or files installed through npm^[11], and converts them to static assets. This enables us to take a fully dynamic application and package it into static files, which can then be uploaded or deployed to a server. Webpack functionalities can be extended by using Plugins and Loaders. We can configure webpack through webpack.config.js

Webpack Configuration

Webpack has an entry point which in our case was the project root file. It inspects that file and traverses its import dependencies recursively, for example if you’re importing @angular/core, it adds that to its dependency list for potential inclusion in the bundle. It opens the file and follows its network of import statements until it has built the complete dependency graph from the root file down.

Loaders are transformations that are applied to the source code of a module; therefore, loaders are kind of like “tasks” in other build tools and provide a powerful way to handle front-end build steps. They can transform files from a different language like type script to java script.

Angular integrates with webpack when generating source maps during the build process. The problem with using ng build `--source-map` is that we don't have control over the build features and which loaders are used. The error above is due to a missing step or configuration in remapping the source files, so we first tried to add options in `tsConfig.js` file using several loaders like Istanbul-loader or give webpack configuration as an option, but such an option was not found.

As we were trying to solve this error, we took several approaches:

Build using Webpack:

We thought of customizing our build options by using webpack to build our application instead of ng build so we needed to install it with the following steps:

Webpack Installation

The following steps show how we can install webpack in our web application. To install the latest release run the following command:

```
npm install --save-dev webpack
```

If you want to call webpack from the command line, you'll also need to install the CLI.

```
npm install --save-dev webpack-cli
```

To create the `webpack.config.js` which is going to contain the options.

```
webpack-cli init
```

The figure shown below are how we answered the questions after we ran the above command


```
webpack-cli init
? Which of the following JS solutions do you want to use? ES6
? Do you want to use webpack-dev-server? Yes
? Do you want to simplify the creation of HTML files for your bundle? Yes
? Do you want to add PWA support? Yes
? Which of the following CSS solutions do you want to use? CSS only
? Will you be using PostCSS in your project? No
? Do you want to extract CSS for every file? Yes
? Do you like to install prettier to format generated configuration? Yes
[webpack-cli] i INFO Initialising project...
conflict package.json
? Overwrite package.json? overwrite
force package.json
identical src/index.js
conflict README.md
? Overwrite README.md? overwrite
force README.md
identical index.html
conflict webpack.config.js
? Overwrite webpack.config.js? overwrite
force webpack.config.js
create .babelrc
warn @angular-devkit/build-angular@12.0.10 requires peer of webpack@>4.0.0
```

Figure 13: Webpack Installation Process

This method did not build the styles and html files, therefore, we couldn't serve the applications.

Custom Build

Since the build provided by angular has webpack implemented in it, as a work around to avoid the webpack error ,we created a custom build script file called build.js also we gulp ,this didn't include Webpack, to build the application with. However, this approach failed because the build was incomplete and hence the serving failed since there was no runtime or index files that are required from the build.

3.2.5. Remap Istanbul approach

Previously the nyc command was generating code coverage on source files only, but it had no access to the source maps via webpack. Therefore, we replaced the last command with a new command using remap – Istanbul^[13].

Remap-Istanbul command has 3 libraries that perform the following:

- **lib/loadCoverage** - does the basic loading of a Istanbul JSON coverage files.
- **lib/remap** - does the remapping of the coverage information. It iterates through all the files in the coverage information and looks for JavaScript Source Maps which it will then use to remap the coverage information to the original source.
- **lib/writeReport** - a wrapper for the Istanbul report writers to output the any final coverage reports.

The following steps were required to install the remap- Istanbul:

```
npm install -g Istanbul
```

```
npm install remap-Istanbul
```

The following command was the one used instead the last previous one:

```
./node_modules/.bin/remap-istanbul -i coverage.json -o html-report -t html --exclude  
node_modules
```

In this approach, we faced an error shown below, thus error states that it could not create such directory 'webpack:/'.

Error: EINVAL: invalid argument, mkdir

**'C:\Users\anod\Desktop\employeemanager\employeemanagerapp\covOut\webpack:\
node_modules\@angular\common__ivy_ngcc__\fesm2015'**

Firstly, As a temporary solution, we replaced every 'webpack:/' with 'webpack//', which were exactly 3 instances, in the coverage.json file. Then instead of replacing it manually we used the following command:

```
(gc coverage/Add_coverage.json) -replace 'webpack:/' , 'webpack//' | Out-File  
coverage/Add_coverage.json -encoding ASCII
```

4. Merging code coverage results

All the previous implementations and trials were tested on a single test case, However when we start applying on the Coverage Analyzer which is the company's web application we are required to run a full regression. Therefore, we started in the following section to merge the coverage results of several test cases on our application.

4.1. Backend merging steps

We want to merge the coverage results coming from different test cases; as we need to merge the coverage of test cases under a single scope in order to deal with a single number. We can also merge the statistics of several scopes later on.

Steps

1. Run the first test case and dump coverage results in jacoco1.exec file

```
java -javaagent:jars/org.jacoco.agent-0.8.7-runtime.jar=destfile=jacoco1.exec -jar
employeemanagerjar2/employeemanager.jar
```

2. Run the second test case and dump coverage results in jacoco2.exec file

```
java -javaagent:jars/org.jacoco.agent-0.8.7-runtime.jar=destfile=jacoco2.exec -jar
employeemanagerjar2/employeemanager.jar
```

3. Merge .exec files

```
java -jar jacoco-0.8.7/lib/jacococli.jar merge jacoco1.exec jacoco2.exec --destfile merged.exec
```

4. Get the merged html report








```
java -jar jars/org.jacoco.cli-0.8.7-nodeps.jar report merged.exec --classfiles=target/classes --html
coveragemerge --sourcefiles=../Web/employeemanager/employeemanager/src/main/java
```

Results validating the merging steps

The first two figures show the results of the first and second test case separately, where in the first test case add and delete employee functions were hit and in the second test case update employee function was hit.

JaCoCo Coverage Report > com.reem.employeemanager.Service > EmployeeService

EmployeeService

Element	Missed Instructions	Cov.	Missed Branches	Cov.	Missed	Cxty	Missed	Lines	Missed	Methods
● lambda\$findEmployeeById\$0(Long)		0%		n/a	1	1	1	1	1	1
● findEmployeeById(Long)		0%		n/a	1	1	1	1	1	1
● updateEmployee(Employee)		0%		n/a	1	1	1	1	1	1
● addEmployee(Employee)		100%		n/a	0	1	0	2	0	1
● EmployeeService(EmployeeRepo)		100%		n/a	0	1	0	3	0	1
● deleteEmployee(Long)		100%		n/a	0	1	0	2	0	1
● findAllEmployees()		100%		n/a	0	1	0	1	0	1
Total	29 of 54	46%	0 of 0	n/a	3	7	2	10	3	7

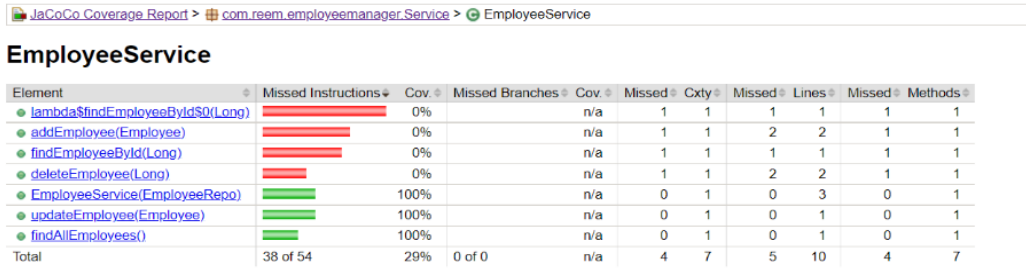


Figure 14: Employee Manager Backend Unmerged Reports

In the following figure we can see the merged coverage report where the three functions are hit.

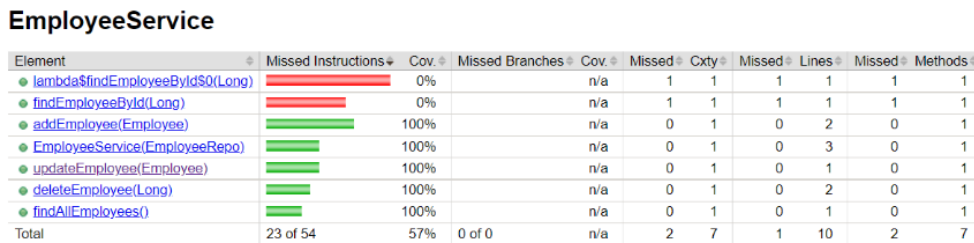


Figure 15: Employee Manager Backend Merged Report

4.2. Frontend merging steps

We used the same command but we included all the input .json files ,these files represent different test case files ,For example, the command below include a file with the add functionality and the other one include the delete functionality.

```
istanbul-merge --out <output.json> <input1.json> <input2.json> .....
```

Results validating the merging steps

The following screenshot shows the coverage report when the two files are used (Add_coverage.json, Delete_coverage.json). Here as shown both the Add and Delete functions are hit

```

46
47 public onAddEmployee(addForm: NgForm): void {
48   document.getElementById('add-employee-form').click();
49   this.employeeService.addEmployee(addForm.value).subscribe(
50     (response: Employee) => {
51       console.log(response);
52       this.getEmployees();
53       addForm.reset();
54     },
55     (error: HttpErrorResponse) => {
56       alert(error.message);
57       addForm.reset();
58     }
59   );
60 }
61
62 public onUpdateEmployee(employee: Employee): void {
63   this.employeeService.updateEmployee(employee).subscribe(
64     (response: Employee) => {
65       console.log(response);
66       this.getEmployees();
67     },
68     (error: HttpErrorResponse) => {
69       alert(error.message);
70     }
71   );
72 }
73
74 public onDeleteEmployee(employeeId: number): void {
75   1x this.employeeService.deleteEmployee(employeeId).subscribe(
76     (response: void) => {
77       console.log(response);
78       this.getEmployees();
79     },
80     (error: HttpErrorResponse) => {

```

Figure 16: Employee Manager Frontend - Add & Delete Functions Hit in Merged Report

The following fig. shows the coverage report when the only one file is used (**Add_coverage.json** or **Delete_coverage.json**). Here as shown either the Add or Delete functions are hit

```

46
47 public onAddEmployee(addForm: NgForm): void {
48   document.getElementById('add-employee-form').click();
49   this.employeeService.addEmployee(addForm.value).subscribe(
50     (response: Employee) => {
51       console.log(response);
52       this.getEmployees();
53       addForm.reset();
54     },
55     (error: HttpErrorResponse) => {
56       alert(error.message);
57       addForm.reset();
58     }
59   );
60 }
61
62 public onUpdateEmployee(employee: Employee): void {
63   this.employeeService.updateEmployee(employee).subscribe(
64     (response: Employee) => {
65       console.log(response);
66       this.getEmployees();
67     },
68     (error: HttpErrorResponse) => {
69       alert(error.message);
70     }
71   );
72 }
73
74 public onDeleteEmployee(employeeId: number): void {
75   1x this.employeeService.deleteEmployee(employeeId).subscribe(
76     (response: void) => {
77       console.log(response);
78       this.getEmployees();
79     },
80     (error: HttpErrorResponse) => {

```

Figure 17: Employee Manager Frontend – Delete Function Hit

```
47 public onAddEmployee(addForm: NgForm): void {
48 1x document.getElementById('add-employee-form')!.click();
49 this.employeeService.addEmployee(addForm.value).subscribe(
50 (response: Employee) => {
51 1x console.log(response);
52 this.getEmployees();
53 addForm.reset();
54 },
55 (error: HttpResponse) => {
56 alert(error.message);
57 addForm.reset();
58 }
59 );
60 }
61
62 public onUpdateEmployee(employee: Employee): void {
63 this.employeeService.updateEmployee(employee).subscribe(
64 (response: Employee) => {
65 console.log(response);
66 this.getEmployees();
67 },
68 (error: HttpResponse) => {
69 alert(error.message);
70 }
71 );
72 }
73
74 public onDeleteEmployee(employeeId: number): void {
75 this.employeeService.deleteEmployee(employeeId).subscribe(
76 (response: void) => {
77 console.log(response);
78 this.getEmployees();
79 },
80 (error: HttpResponse) => {
```

Figure 18: Employee Manager Frontend – Add Function Hit

5. Deployment on a Real Case Web Application

5.1. Applying the Code Coverage Measurement

5.1.1. Introduction

After emulating VIQ environment and getting the coverage results, we started deploying our package on Coverage Analyzer which is a product that visualizes the RTL code coverage by HW Verification saved as a Universal Coverage Data Base (UCDB). We started with coverage analyzer because it's a standalone branch from VIQ with an easier setup.

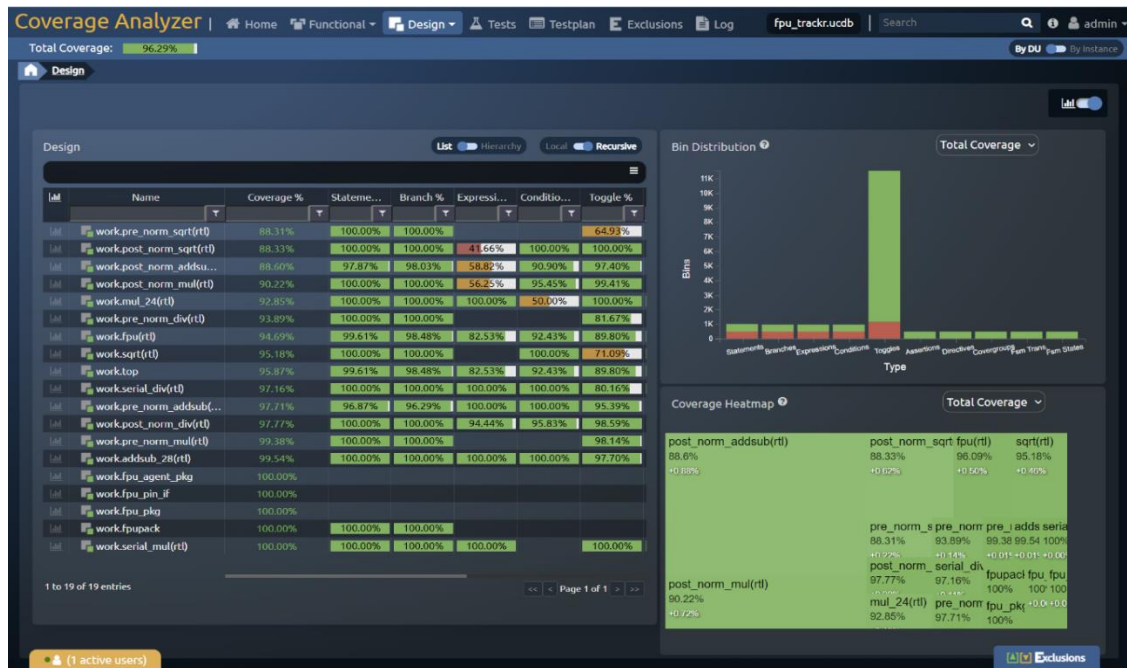


Figure 19: Coverage Analyzer UI

5.1.2. VIQ Workshops

We viewed the structure of Coverage Analyzer test cases regression suite^[15]. Regression Testing is defined as a type of software testing to confirm that a recent program or code change has not adversely affected existing features. We found that we have a tree of test cases and each leaf containing selenium test cases for certain functionalities.

A Perl script is responsible for managing the process of running each test case. By running the test script Coverage Analyzer is started, we observed the sequence of steps it uses which we will try to imitate using our code coverage scripts.

The leaf test case flow:

1. Compile java files (mainly responsible for the functionality of the selenium test case)
2. Provide files needed for coverage analyzer to work properly, for example ucdb files
3. Launch the backend and frontend server using the war file and pass the required arguments, for example the server port
4. Run compiled files of the test case
5. Kill the process of coverage analyzer and compare results with golden references

Afterwards, we unzipped the .war file of the application to view its structure, and how it's created from the build script.

We found that the frontend and backend are built separately then copied in the same directory and zipped into one Web Application Archive (war) file.

Furthermore, we investigated the build script and the tasks responsible for creating the war file.

Examples of the tasks used:

- **Scrub:** responsible for cleaning any files from a previous build and install needed dependencies, like node modules
- **Build:** runs angular build responsible for building frontend
- **Copy:** copies the build files of the frontend to the same directory of the backend

After investigating the building method used for the application, we realized that the backend is built by Gradle. Therefore, we needed to migrate our project from Maven to Gradle.

5.1.3. Adjustments needed

a) Migration from Maven to Gradle

As mentioned above, the Coverage Analyzer application uses Gradle not Maven for building, in addition, Gradle is more customizable than Maven and provides a wide range of IDE support custom builds while Maven has a limited number of parameters and requirements, so customization is a bit complicated.

The following figure compares between Gradle and Maven for three types of builds in regards to their duration. As shown in the figure, Gradle has less building times in all of the build types.

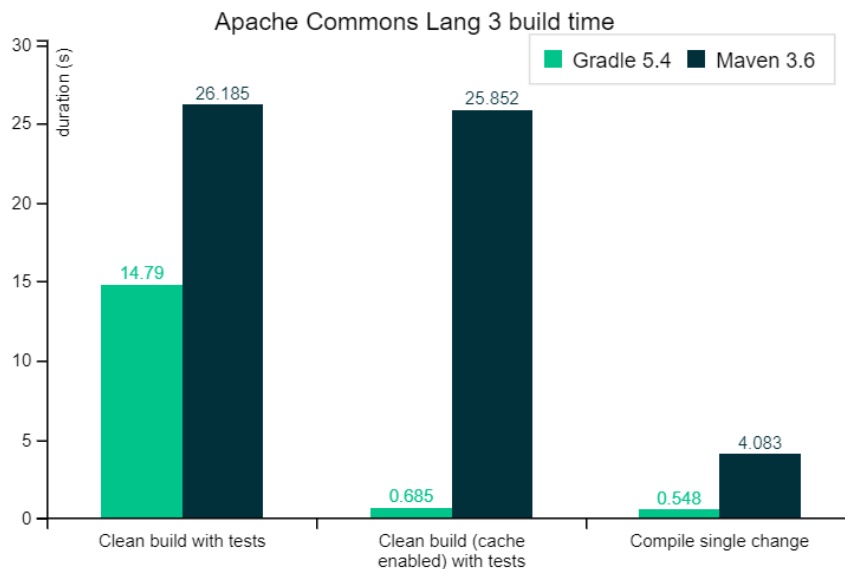


Figure 20: Gradle vs Maven

We migrated the project by installing Gradle from their website, then inside the backend root we ran the command “*gradle init*”

b) *Generating WAR File*

After migration, we generated the war file using the following steps:

- Added the war plugin inside the file ‘build.gradle’, this will produce a .war file upon building

```
1  plugins {
2      id 'org.springframework.boot' version '2.6.3'
3      id 'io.spring.dependency-management' version '1.0.11.RELEASE'
4      id 'java'
5      id 'war'
6  }
```

Figure 21: Added Plugins for Gradle

- Run command “*gradlew build*” to build the application, it will produce a war file similar to the VIQ one but without the frontend.

Merging Frontend & Backend Into one WAR File

In order to add the frontend to the same war file as the backend, normally, a plugin provided by gradle is used that automatically copies the build files of the frontend to the backend directory.

What we did was to first build and instrument the frontend like we would normally do, then we manually copied the resulted build files to the backend in the following directory:

webapp\build\resources\main\static

Then we used Gradle build again to produce a war file containing both frontend and backend, and we can run it using:

java -javaagent:jars/org.jacoco.agent-0.8.7-runtime.jar -jar employeeManager-0.0.1-SNAPSHOT.war

5.1.4. Deployment of Back-End code coverage measurement to the Real Case application

We started with the backend because it does not need special handling in war file creation (i.e instrumentation). It just needs a modification in the war file running command. We started on a small scale and picked a random test case to use. We added the java agent to the java command inside the perl script in order to collect coverage data for backend.

Challenges

The challenge that faced us here was that the jacoco.exec file was empty after running the test case. After analysis we found that the problem was inside subroutine Cleaning_Coverage_Analyzer that terminated coverage analyzer using kill -9 which terminates without saving the data. Consequently, we replaced this command with kill -15 which gracefully terminates the program and saves data. These changes were in run_ca.pl file in running CA and cleaning CA subroutines.

After this change we successfully ran the test case and got the backend coverage report.

JaCoCo Coverage Report

Element	Missed Instructions	Cov.	Missed Branches	Cov.	Missed	Cxty	Missed	Lines	Missed	Methods	Missed	Classes
com.mentor.dvt.coverageanalyzer.service.impl		6%		2%	5,362	5,673	11,310	12,165	1,438	1,711	16	81
com.mentor.dvt.coverageanalyzer.model		5%		1%	2,252	2,414	3,841	4,147	1,203	1,358	68	81
com.mentor.dvt.coverageanalyzer.ini.mapper		19%		11%	866	994	2,173	2,729	160	244	4	21
com.mentor.dvt.coverageanalyzer.service.dto		6%		0%	1,700	1,837	2,540	2,754	1,287	1,423	77	91
com.mentor.dvt.coverageanalyzer.service.mapper		9%		3%	617	710	1,711	1,923	168	258	18	88
com.mentor.dvt.coverageanalyzer.analysis.expressions		0%		0%	553	553	904	904	263	263	11	11
com.mentor.dvt.coverageanalyzer.web.rest		11%		1%	260	304	564	654	180	224	3	29
com.mentor.dvt.coverageanalyzer.covergroups.service		1%		0%	179	185	452	461	61	67	0	5
com.mentor.dvt.shared.web.rest.util		7%		2%	255	270	465	504	56	70	9	12
com.mentor.dvt.coverageanalyzer.domain.enumeration		48%		5%	126	155	230	335	64	89	13	25
com.mentor.dvt.coverageanalyzer.covergroups.analysis		0%		0%	163	163	360	360	66	66	6	6

Figure 22: Coverage Analyzer Backend Coverage Report for a Single Test Case

BinsClusteringService

Element	Missed Instructions	Cov.	Missed Branches	Cov.	Missed	Cxty	Missed	Lines	Missed	Methods	
buildHistogramBinsRanges(List, Integer, Integer)		0%		0%	20	20	49	49	1	1	
clusterOnBins(List, Integer)		0%		0%	5	5	20	20	1	1	
getBinsDistribution(List, Integer, Integer, Integer)		0%		0%	5	5	24	24	1	1	
clusterOnCoverage(List)		0%		0%	6	6	18	18	1	1	
buildCoverageClusterDTO(AbstractCoverElementWithAvgBins, List, Int)		0%		0%	2	2	15	15	1	1	
getNumClusters(Integer)		0%		0%	3	3	1	1	1	1	
getFinalMinBinValue(Integer)		0%		0%	2	2	1	1	1	1	
getFinalMaxBinValue(Integer)		0%		0%	2	2	1	1	1	1	
static { ... }		100%		n/a	0	1	0	3	0	1	
Total		683 of 741	7%	74 of 74	0%	45	46	129	132	8	9

Figure 23: Coverage Analyzer Backend Coverage Report for a Single Test Case

The following figure shows the hits and misses illustrated by the highlights, where red indicates a miss and green indicates a hit, inside the html report.

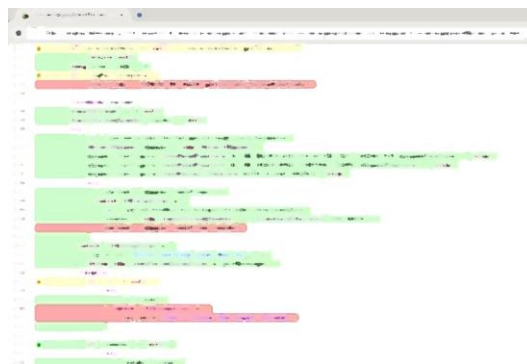


Figure 24: Coverage Analyzer Backend Coverage Report for a Single Test Case - Hits & Misses

5.1.5. Deployment of Front-End code coverage measurement to the Real Case application

In this step we needed to adjust the build steps so we started by checking the build command that was used “./gradlew -PScrubFe -Pqaproduct clean build -x test”

We found that -P option with gradlew calls custom tasks defined in “**gradle/profile_prod.gradle**”, where ScrubFe was used for cleaning and installing needed packages and qaproduct runs the script in package.json corresponding to QA production which runs with the production configurations specified. Finally, we generate the war file.

QA Production build

There are several configurations in qaproduct build but we’ll focus on three main configurations.

- **Source map option:** set to false
- **Optimization:** set to true
- **Build Optimizer:** set to true

Challenges

Build Failure with Source Maps Option

We first tried to set the Source maps option to true but the build failed. This was due to the conflict that occurs between the source maps and optimization options. Consequently, we contacted the R&D to find a solution and eventually came to the conclusion that we need to use the Development build (Debug build) instead.

Memory Issue

Here we faced another problem with the memory, so we needed to increase it with the following command:

```
node --max_old_space_size=8192 node_modules/@angular/cli/bin/ng build --  
configuration=dev
```

Test Cases Failure

Due to the difference between the production and debug builds, for example the optimization options, 8% of the test cases failed. This was acceptable percentage for now so we proceeded with this solution.

The next steps was to add the instrumentation command in package.json and add a corresponding task to call it in gradle/profile_dev.gradle and defined task qacov to call the development build with our configurations and the instrumentation task. Therefore the final build command is “./gradlew -PScrubFe -Pqacov clean build -x test”

Finally, to dump the coverage data we added the snippet used to dump the coverage in the coverage.json file in the EXIT function that is called at the end of each test case and defined environmental variable (CA_COVERAGE) to only call the dump when we are collecting coverage. We also created a script that adds the needed imports in Test Cases for the coverage data dumping code.

```
public static void dumpCoverage(WebDriver driver) throws Exception {
    if(System.getenv("CA_COVERAGE") != "")
    {
        JavascriptExecutor js = (JavascriptExecutor) driver;
        Object str = js.executeScript("return window.__coverage__");
        GsonBuilder builder = new GsonBuilder();
        Gson gson = builder.create();
        String coverage = gson.toJson(str);
        Files.write(Paths.get("coverage.json"), coverage.getBytes());
    }
}
```

Figure 25: dumpCoverage Function in Common Class

5.1.6. Edited Files

To further illustrate the changes mentioned above, the following files were modified as follows:

Run Common File

This is a common perl file that contains subroutines that are used to run all the test cases.

In this file, after running the test case we added the following in run test subroutine:

- A timestamp before the generation of the coverage reports to calculate the overhead
- The command for merging the json files of the frontend coverage reports. This was not needed for the backend as the exec file is generated after closing the server so it will already contain the coverage of the whole directory
- A code block for checking the number of json files in the directory to check if merging is needed. The json files are named “test_case.json” and the merged file is “coverage.json”, if there was only one file it’s renamed “coverage.json”
- A timestamp after report generation for overhead calculation

Run_ca.pl File

In this file we edited the java command with java agent to run the war file subroutine running_CA as well as editing the kill command (kill -15) in clean_CA subroutine

Post Script execution in the regression suite launcher tool

We edited the post script in the regression launcher tool when running on the full regression to archive the coverage data as the regression launcher tool archives only the data of the failed test cases.

5.1.7. Summary for Deployment on Real Case Web Application

To summarize the whole work on Coverage Analyzer, we modified parts of the flow starting from the build to the launch of the application and the termination. Also, we generated frontend and backend coverage reports for each leaf test case.

5.2. Running a full regression

Here, we are trying to collect coverage for the whole test case suite to:

- Measure the quality of testing from code coverage perspective
- Check if there are any challenges in getting coverage for the full regression

5.2.1. Preparation steps for running a full regression

1. Defining the required Shell environment variables:

- Set the required fields like "MTI-HALOS, MTI_HOME. TEST_SUITE, etc"
- Set "setenv" field with the additional coverage variables and the archive directory (for coverage results) → Example
 - **CA :** CA_COVERAGE=1 CA_SRC=../CA_main/main/coverageanalyzer (or other <ws> specified)
archive_path="/bata/halos/viq_codecoverage/AllRegression_2""
 - **VIQ :** VIQ_COVERAGE=1 VIQ_SRC=../CA_main/main/coverageanalyzer
archive_path="/bata/halos/viq_codecoverage/AllRegression_2""
 - You can find the post run script under <your main workspace>/tests/viq_automation/run_scripts/post_run.csh

2. Editing the post script

- Set "other options" field in the regression launcher tool with the following line → "--postrun-script <post run absolute path> --force-completion --test-timeout=40 --timeout=200" for example: "--postrun-script=/bata/lalmasry/CA_main/main/tests/viq_automation/run_scripts/post_run.csh"

3. Results

- Raw coverage results can be found in your archive_path under <testcase_path from ca>/coverage.json(or jacoco.exec) → example: /bata/halos/viq_codecoverage/AllRegression_2/ca/code_coverage/codeEditor/branchesSV1/coverage.json. " ../viq_codecoverage/AllRegression_2" was my archive_path and "ca/code_coverage/codeEditor/branchesSV1: is the testname

5.2.2. Processing Data for Test Cases

- **Generate frontend html coverage report :** "remap-istanbul -i coverage.json -o <Frontend_Coverage report_name> -t html --exclude node_modules"
- **Generate backend html coverage report :** "java -jar \$JARS/org.jacoco.cli-0.8.7-nodeps.jar report jacoco.exec --sourcefiles=\$CA_SRC/src/main/java/ --classfiles=\$CA_SRC/build/classes --html <Backend_Coverage report_name>"
- **Merge front end results:** "istanbul-merge --out <merge_file.json> <input1.json> <input2.json> "
- **Merge backend results:** "java -jar \$JARS/org.jacoco.cli-0.8.7-nodeps.jar merge <input1.exec> <input2.exec> ... --destfile <merged.exec>"
- The whole processing data section can be automated "to be done"

5.2.3. Useful hints for processing data

Make a list with all the coverage.json and exec files for further processing

- cd to your archive_path
- To generate Jsonfiles list:

```
find * -name coverage.json > Jsonfiles.list
```

- To generate execfiles list from only testcases that finished run successfully:

```
cat Jsonfiles.list | sed 's,coverage.json,jacoco.exec,g' > Execfiles.list
```

Merge files in sets then merge the final sets

The concept of merging files in sets can be used to make the results more organized, for example merging the coverage files of test cases with similar functionality. This can be done using the following steps:

- Merge json files in Jsonfiles.list from line_1 to line_99 (The first 100 lines) into first merged set

```
istanbul-merge --out merge_1set.json `sed -n 1,99p Jsonfiles.list`
```

- Merge json files in Jsonfiles.list from line_100 to line_199 (The second 100 lines) into second merged set

```
istanbul-merge --out merge_2set.json `sed -n 100,199p Jsonfiles.list`
```

- Merge the two output files
- Generate coverage report for the final exec or json file

The previous merging steps can be done both on the Back-End and Front-End files.

Merge all files at once

The following command can be used:

```
istanbul-merge --out merge_all.json `cat Jsonfiles.list`
```

Suggested solutions for common issues

In case you found corrupted .json or .exec files, you can remove them from the lists and run the commands again

You can use *merge_backend.pl* to automate the process of merging exec files and remove the corrupted files from the list after forming Execfiles.list.

In case you have an issue in the installation of packages you can use *.../nodeJS/node-v10.16.3-linux-x64/bin/istanbul-merge* instead of *istanbul-merge* and same for all other npm commands.

5.2.4. Processing Data for the Regression

Preparing the list of tests with coverage

This list can be prepared as mentioned previously.

Merge FrontEnd Coverage

- cd to Coverage Parent directory (archive_path)
- Run the following PERL program merge_frontend.pl

```
#open(PASSN,'>', "./file");
print"Merging frontend started\n";
$results="/home/lelmasry/Development/nodeJS/node-v10.16.3-linux-x64/bin/istanbul-merge --out merged.json `cat jsonfiles.list`;
print "$? \n";
while($? ne "0")
{
  `rm file`;
  open(PASSN,'>', "./file");
  print PASSN $results;
  $lineTodelete = `tail -1 file | sed 's,.*ca,ca,g';
  print "$lineTodelete is corrupted";
  chomp($lineTodelete);
  #print "echo -n '$lineTodelete' | sed 's,/,\,g";
  #sline = `echo -n '$lineTodelete' | sed 's,/,\,g' | sed 's,\,\,g';

  $sline = `echo -n '$lineTodelete' | sed 's,\.\,g';
  #quotemeta($lineTodelete);
  #print $sline;
  #print "`sed 's,.$sline,,g' -i file`;
  print"`sed 's,$sline,,g' -i jsonfiles.list`;
  `sed 's,$sline,,g' -i jsonfiles.list`;

  $results="/home/lelmasry/Development/nodeJS/node-v10.16.3-linux-x64/bin/istanbul-merge --out merged.json `cat jsonfiles.list`;
  close(PASSN);
  print "$? \n";
}
#print"`sed '/^$/d' file`";
```

Figure 26: Frontend Merging File

- Merged file will be archive path/merged.json

Merge Backend Coverage

- cd to Coverage Parent directory (archive path)
- Run the following PERL program merge_Backend.pl


```

#open(PASSN,'>', "./file");
print "Merging backend started\n";
$results="java -jar $ENV{"JARS"}/org.jacoco.cli-0.8.7-nodeps.jar merge \cat ExecFiles.list\ --destfile merged.exec`;
print "$? \n";
while($? ne "0")
{
`rm file`;
open(PASSN,'>', "./file");
print PASSN $results;
$lineTodelete = `tail -1 file | sed 's,.*ca,ca,g`;
print "$lineTodelete is corrupted";
chomp($lineTodelete);
#print "echo -n '$lineTodelete' | sed 's,/,\\,g";
#$line = `echo -n '$lineTodelete' | sed 's,/,\\\\\\\\\\,g' | sed 's,\\,.,g`;

$line = `echo -n '$lineTodelete' | sed 's,\\.\\.$,.,g`;
#quotemeta($lineTodelete);
#print $line;
#print "`sed 's,.*$line,g' -i file`";
print "`sed 's,$line,g' -i ExecFiles.list`";
`sed 's,$line,g' -i ExecFiles.list`;
$results="java -jar $ENV{"JARS"}/org.jacoco.cli-0.8.7-nodeps.jar merge \cat ExecFiles.list\ --destfile merged.exec`;
close(PASSN);
print "$? \n";
}
#print "`sed '/^\\$/d' file`";

```

Figure 27: Backend Merging File

- merged file will be archive_path/merged.json

Generate test: line coverage file per test case

- Generate a file that will state each file and line affected by this test case you should do the following
- Loop on all directories that contain coverage .json file and run the script as follows

5.2.5. Results of running a full Regression on Coverage Analyzer using Regression Launcher Tool

The coverage data processing was performed on 521 Passing designs and failing designs with total ~620 raw coverage file (for each backend and frontend)

Make Done:	0	Pass:	521	68.73%
Make Errors:	0	Fail:	234	30.87%
Total:	758	KFail:	0	0%
Pending:	3	T/O:	0	0%
Running:	0	Miss:	0	0%
Re-running:	3	N/A:	0	0%
		N/R:	0	0%

Figure 28: Regression suite launcher tool Results

com.mentor.dvt.coverageanalyzer.logback	18%	0%	4	5	6	7	2	3	0	1		
com.mentor.dvt.shared.plugin	12%	7%	14	18	56	66	7	11	2	3		
com.mentor.dvt.coverageanalyzer.license.mgls	5%	3%	158	167	347	368	80	89	7	9		
com.mentor.dvt.coverageanalyzer.analysis.datamodel	0%	0%	161	161	331	331	65	65	1	1		
default	0%	0%	55	55	121	121	40	40	5	5		
com.mentor.dvt.coverageanalyzer.integration.halos	0%	0%	26	26	92	92	14	14	3	3		
com.mentor.dvt.coverageanalyzer.visitor	0%	0%	13	13	33	33	11	11	1	1		
com.mentor.dvt.coverageanalyzer.web.interceptor	0%	n/a	1	1	1	1	1	1	1	1		
Total	40,935 of 158,345	74%	6,640 of 17,134	61%	6,320	16,392	8,972	33,569	1,739	7,663	105	712

Figure 29: Coverage Analyzer - Full Regression Backend Coverage Report

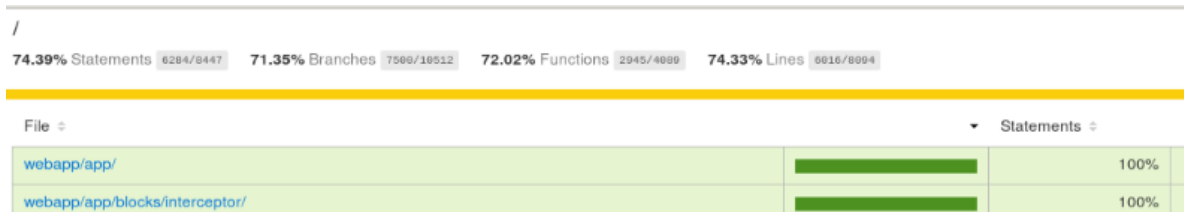


Figure 30: Coverage Analyzer - Full Regression Frontend Coverage Report

5.3. Final Package

The Test Automation Team in Siemens EDA currently:

- Using our steps to build with coverage, and archive coverage results weekly for the whole regression
- Running set of scripts to merge coverage data and generate frontend and backend reports
- Running script to generate summaries for each run (archived with its date)

```

20220508
backend line : 0
frontend line : 12.68
frontend function : 11.79
average line = 6.34
average function = 11.79
~
~
    
```

Figure 31: Results of Merging Script

- Opening an html page which provides:
 - The summary of the most recent run
 - The ability to view the detailed frontend and backend coverage
 - The ability to navigate to the coverage report of a previous run
 - A list of all the previous archived results



Figure 32: HTML Navigator/Report

Feedback from Siemens Test Automation Team

The test automation team started presenting our work and applying it to different VIQ web applications other than Coverage Analyzer.



Figure 33: VIQ Code Coverage Report

The test automation team also started analysing the reduced test case list that resulted from the test reduction tool mentioned above.

5.4. Challenges in Deployment Process

In this section we will discuss some of the challenges that the coverage test automation team faced when using our tool on their applications.

1. Cannot work with fully Dev build

Our tool needs development build as it contains a high level of debugging, which is needed to catch bugs. However, the customer will need to use a production build as debugging isn't needed, and also it's more abstract than the dev build. And they need maximum optimization from the build, this forms a contradiction in interests.

Furthermore, Dev build can't be used to run VIQ regression because all of the SQL files are saved based on production VIQ build, and production SQL files can't be used with dev build. To overcome this issue, in the building steps instead of setting dev profile we added prod profile but still execute buildnonprod task as shown in the figure. So by doing this we can create dev build, instrument the build by setting profile = prod, the build will behave the same as the prod build and we can run the test cases normally and collect code coverage for both the frontend and backend.

```

! ext {
!   logbackLogLevel = "INFO"
! }
! dependencies {
! }
!
! def profiles = 'prod'
! project.ext.profile=profiles
! if (project.hasProperty('no-Liquibase')) {
!
!   37 task buildnonprodcov(type: NpmTask){
!   38   workingDir file("${project.projectDir}")
!   39   args = ['run', 'buildnonprodcov']
!   40 }
!   41
!   42
!   43 task Instrument(type: NpmTask , dependsOn: ['buildnonprodcov']){
!   44   workingDir file("${project.projectDir}")
!   45   args = ['run', 'instrument']
!   46 }
! }
! }

6 ext {
7 !   logbackLogLevel = "DEBUG"
8
9
10 ! def profiles = 'dev'
11 project.ext.profile=profiles
12 if (project.hasProperty('no-Liquibase')) {

```

Figure 34: Buildnonprodcov task

2. Missing files in frontend that were not reported

Some required files were included as library files (in node modules) and not as source files, therefore when collecting coverage these files were missing from the reports

3. Storage problem

Each test case coverage reports are about 62 MB to 38 GB for one regression run

4. All files under shared directory are now included in one file and not in separate files as in the actual source codes

5.5. Formatting the results for usage in applications

After inspecting the HTML reports that were generated, we started collecting the useful coverage data and formulating this data into files to be used in different applications. We created a parsing script to extract information from the HTML reports into text files, each test case has a new file that contains source file name as well as the numbers of lines (i.e. the number of the line in the source code) that were hit by this test case.

These parsed files can be used as input in various applications since it's easier to extract the data of each test case from them directly. One of the applications that can use the parsed files as input, is reducing the size of the regression test suite, which will be discussed in the following section.

6. Applications of Code Coverage

6.1. Introduction to Code Coverage Applications

Software quality is an important issue that all developers of software systems want to achieve. It currently attracts a lot of attention since software is everywhere and affects our lives on a daily basis. Software testing is the main factor in enhancing and increasing the quality of software.

Regression testing necessitates running a large program on a large number of test cases, which can be costly in both human and machine time. Software costs may be reduced if the regression testing process could be improved. The goal of researchers using test-suite selection strategies is to reduce costs. This is why they strive to find test-suite subsets that provide the same level of software coverage as the original test-suite. As a result, a variety of approaches for dealing with test suits have been investigated, including minimization, selection, and prioritising. Minimizing or reducing the number of tests to execute is the goal of test suite minimization.

Now that we understand the importance of controlling the size of a test case regression, we can take a look at the different metrics upon which we can make an optimum decision of whether or not the test-suite is compact and efficient. Some of the various metrics are as follows:

Customer requirements:

In the customer requirements based selection techniques test cases are divided based on the factors decided on the requirements of customers documented during the phase of requirements gathering.

Cost effective:

The test cases are classified on the basis of the cost factor in this approach. The cost can be the cost of requirement gathering, cost of regression testing, cost of execution and validating test cases, the cost of analyses to select and support a test case, cost of classification of test cases, cost of the running time or any other implicit cost, e.g. test environment (hardware), competence or other cost pending factors in the development or production cycle.

History based:

The test cases are classified based upon the history of the test case itself which means priority of test case depends upon its previous execution time, rate of finding failures and other performance metrics.

Churn:

Testing can also be classified based on churn, e.g. changes. Meaning that you prioritize the test cases affected by the latest code change. Depending on your architecture, programming language choice and many other development factors e.g. how you associate and connect your tests with the code.

Fault-based:

By constantly collecting statistics on every execution of the software, information from e.g. customers, changes, and pass-fail history of the test case, classification can be based on the fault history, including severity or occurrence.

Coverage based:

Owing to the fact that code coverage is one of the most important parameter to calculate in any software testing, as well as being the output of our code coverage tool we will be using it as the metric in our approaches for test reduction, it can be used along with other metrics or as a standalone as we will explain later on.

Based on coverage the classification of the test cases are on the quantity of the source code of a program that has been exercised during testing. In this approach the test cases having the capability of testing a larger part of the code are classified. We can either use percentages only without having any knowledge of the lines within the source code, or by also including the exact lines that were covered. This means that this metric can be used for both white box testing and black box testing.

Coverage-based test suite reduction and prioritization techniques optimize test cases based on the achieved coverage of different aspects (e.g. source code or model) of System Under Test (SUT). The code coverage is used to measure the degree of SUT's code exercised by the generated test suite. Furthermore, it provides feedback about the strategy that should be used to enhance the achieved coverage. The coverage criteria (e.g. statement, branch, or path) act as a stopping point to decide whether the SUT is sufficiently tested or not.

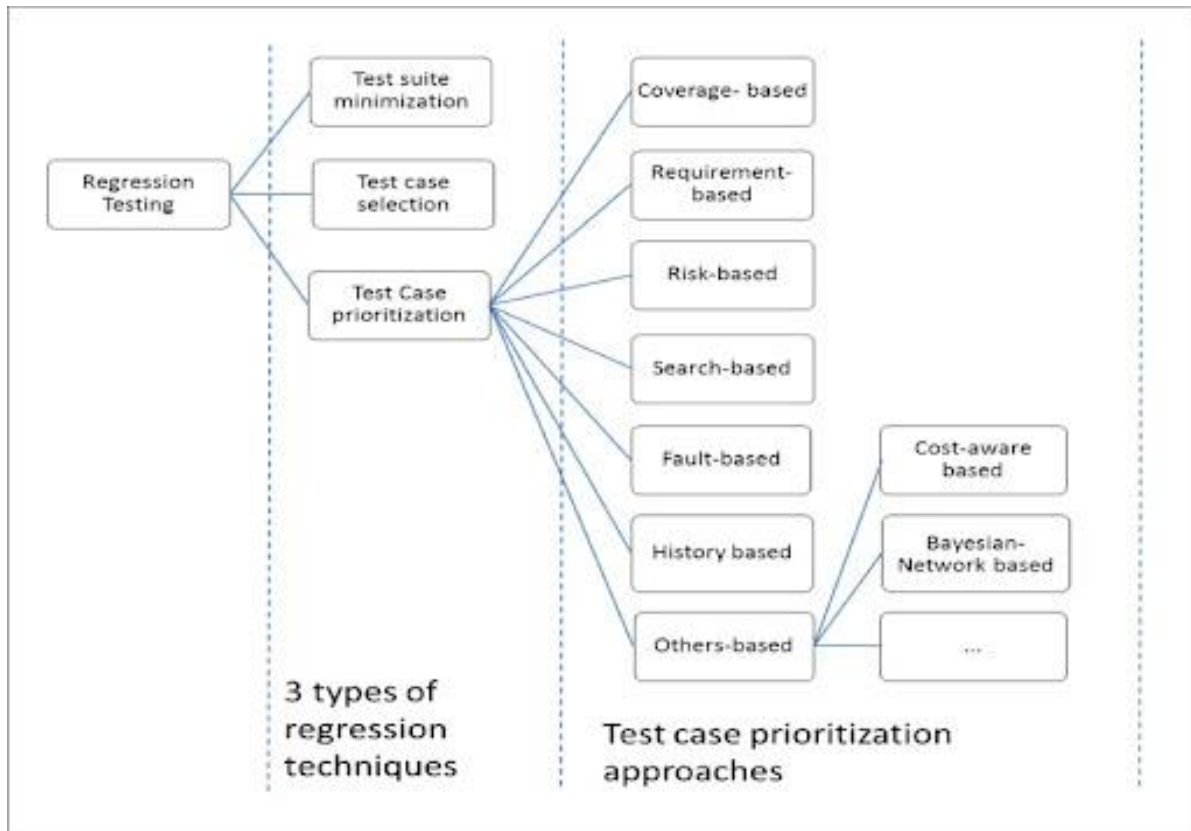


Figure 35: Applications Using Testing Metrics

6.2. Applications Using Code Coverage Results

In phase 1 of our project, we successfully obtained the code coverage data using selenium test cases. For our next steps we started researching on how we can use this data to enhance the development and testing productivity of any software application. Most of the applications revolved around how to optimize a test suite either by prioritizing the test cases, reducing the test cases, or even exposing the blocks of code that are unused, inside the source code or the test cases themselves. In the following section, we will be discussing the major applications that we found, and which can be useful for software testers/developers.

6.2.1. Selection and Prioritization

Test case prioritisation, as the name implies, is the process of prioritising test cases in a test suite based on a variety of parameters. Code coverage, risk/critical modules, functionality, features, and so on are all possible factors. The test suite increases in size when the software itself increases, this also leads to more efforts in order to maintain the test suite. Test case prioritization is important in order to detect bugs in software as early as possible so that important test cases can be executed first.

Types of Test Case Prioritization

- **General Prioritization:**

Test cases that will be relevant for future changed versions of the product are prioritised in this type of prioritising. It does not require any information on the program's adjustments.

- **Version – Specific Prioritization:**

Test cases can also be prioritised so that they are only useful on specific versions of the product. This type of prioritisation necessitates knowledge of program changes.

Prioritization Techniques

Coverage – based Test Case Prioritization:

This type of prioritization is based on code coverage i.e. test cases are prioritized on basis of their code coverage.

- **Total Statement Coverage Prioritization**

In this technique, total number of statements covered by test case is used as factor to prioritize test cases. For example, test case covering 10 statements will be given higher priority than test case covering 5 statements.

- **Additional Statement Coverage Prioritization**

This technique involves iteratively selecting test case with maximum statement coverage, then selecting test case which covers statements that were left uncovered by previous test case. This process is repeated till all statements have been covered.

- **Total Branch Coverage Prioritization**

Using total branch coverage as factor for ordering test cases, prioritization can be achieved. Here, branch coverage refers to coverage of each possible outcome of condition.

- **Additional Branch Coverage Prioritization**

Similar to additional statement coverage technique, it first selects text case with maximum branch coverage and then iteratively selects test case which covers branch outcomes that were left uncovered by previous test case.

- **Total Fault-Exposing-Potential Prioritization**

Fault-exposing-potential (FEP) refers to ability of test case to expose fault. Statement and Branch Coverage Techniques do not take into account fact that some bugs can be more easily detected than others and also that some test cases have more potential to detect bugs than others. FEP depends on :

1. Whether test cases cover faulty statements or not.
2. Probability that faulty statement will cause test case to fail.

Risk – based Prioritization:

This technique uses risk analysis to identify potential problem areas which if failed, could lead to bad consequences. Therefore, test cases are prioritized keeping in mind potential problem areas. In risk analysis, following steps are performed:

- List potential problems.
- Assigning probability of occurrence for each problem.
- Calculating severity of impact for each problem.

After performing above steps, risk analysis table is formed to present results. The table consists of columns like Problem ID, Potential problem identified, Severity of Impact, Risk exposure, etc.

Requirements – based Prioritization:

Some requirements are more important than others or are more critical in nature, hence test cases for such requirements should be prioritized first. The following factors can be considered while prioritizing test cases based on requirements:

- **Customer assigned priority**
the customer assigns weight to requirements according to his need or understanding of requirements of product.
- **Developer perceived implementation complexity**
priority is assigned by developer on basis of efforts or time that would be required to implement that requirement.
- **Requirement volatility**
this factor determines frequency of change of requirement.

- **Fault proneness of requirements**

priority is assigned based on how error-prone requirement has been in previous versions of software.

Metric for measuring Effectiveness of Prioritized Test Suite:

For measuring how effective prioritized test suite is, we can use metric called APFD (Average Percentage of Faults Detected). APFD value can range from 0 to 100. The higher APFD value, faster faults detection rate. So simply put, APFD indicates of how quickly test suite can identify faults or bugs in software. If test suite can detect faults quickly, then it is considered to be more effective and reliable.

6.2.2. Exposure of Unused Code

Code coverage helps testers guide the testing by numerically and graphically visualizing the aspects of code that have been tested and the ones that aren't working correctly. We usually think increasing test cases can only increase that code coverage, but you can increase the coverage percentage of the code by removing unnecessary code.

In some cases, code coverage report can help you to find code which isn't used anymore. For example, private methods, which aren't called anywhere. There is nothing pleasant about wasting time on reading a dead code and trying to understand why it is needed. Code like this should be removed promptly. If you think you may need this code in future - still remove it. You can restore it from version control system if needed.

By exposing the parts of your code that are dead, you will effectively enhance the performance as well as speed of page loading in your web application.

6.2.3. Test Case Size Minimization

Another less obvious example of dead code - dead code in tests. You can have a test method in which you're looping over a list of some objects and make asserts on each of them. If the list for some reason turns out to be empty, the test will pass although none of the asserts actually occur. This kind of bugs is easy to discover with code coverage report because loop body will be shown as not covered. Software testers can easily reduce the size of a test case manually by knowing the code coverage data of each test case.

6.2.4. Test Suite Reduction (TSR)

The test suite reduction aims at identifying and removing all the redundant test cases; therefore, we minimize the number of tests from the test suite. Test suite reduction approaches also speed up

regression testing by removing redundant test cases. Traditional research on test-suite reduction is rather diverse but shares three properties:

- a) Requirements are defined by a coverage criterion such as statement coverage.
- b) The reduced test suite has to satisfy all or almost all the requirements as the original test suite.
- c) The quality of the reduced test suites is measured.

With that being said, each of the above applications is proven to enhance the efficiency and effectiveness of the testing process. In our project scope, we decided to focus only on the test suite reduction. In the following sections we will be discussing the implementation and results of the several approaches we took by applying several machine learning algorithms in order to reduce the test suite without affecting the coverage results greatly.

6.3. Test Suite Reduction Implementation Using Code Coverage & Machine Learning

6.3.1. Introduction

Test suite reduction techniques aims at reducing the test suite size by removing the redundant test cases from original test suite based on certain coverage requirement. In the context of open source development or software evolution, developers often face test suites which have been developed with no criteria and which may need to be adjusted or refined to ensure its dependability, or even reduced to meet the runtime limits of the test suite regression. It is important to provide both methodological and tool support to help people understand the limitations of test suites and their possible redundancies, so as to be able to change them in a cost effective manner. To address this problem in the case of black-box or white-box testing, we propose two methodologies based on machine learning that have shown promising results regarding the test suite size as well as the coverage data.

Test cases are abstracted under the form of category and choice combinations, as defined in Category- Partition. These choice combinations characterize a test case in terms of input and execution environment properties. A machine-learning algorithm is then used to learn about relationships between inputs/environment conditions and outputs as they are exercised by the test suite. This allows the tester to precisely understand the capabilities and weaknesses of the test suite.

Choosing the Suitable Algorithms

Before talking about the two machine learning algorithms that we went with in our project, we will discuss briefly about other algorithms and techniques that could also be used for the same purpose in the following table.

Table 1: Machine Learning Algorithms for Test Suite Reduction^[33]

Algorithm	Technique	Advantages	Disadvantages
Genetic Algorithm	Builds the initial population based on test history, it calculates the fitness value using coverage and cost (customized metric). Then the fitness function is used to evaluate the generated population to choose the best candidates, then the crossover and mutation process are taking place.	Reduce the number of test cases and also decreases total running time.	Need to be examined on the fault detection capability and other criteria
Fuzzy Logic	Allows each feature to belong to more than one cluster with different membership degrees (between 0 and 1) and fuzzy boundaries between clusters. In fuzzy clustering, each point has a degree of belonging to clusters, rather than belonging completely to one cluster only.	A safe technique and reduce the regression testing size and execution time	Need more experiments and studies
Greedy algorithm	Greedy algorithm is used for test suite reduction also called Set Covering Technique. It starts by determining test cases which can satisfy all the requirements. If the test case does not satisfy requirements then the algorithm repeatedly eliminate redundant test cases then update the test suite and	Provide significant reduction in the number of test cases	Involve random selection of test case in a tie situation.

	the remaining requirements that are uncovered.		
Clustering	Divide the test cases into clusters according to the similarity in profiling using data mining approach	Produce smaller representative sets of test cases	Less fault detection ability also it's a statistical method
Program slicing	This technique is used to check a program over a specific property and to build a slice set, which is a set of statements effect to determine a statement; in many cases it is the output statement of a program, based on input values. Two algorithms are used: the first one generates a program called differences, it captures the difference between certified and modified program, where certified is the previously tested program without changes and modified is the program with modification. The second algorithm uses existing test cases to test components new in modified, also it uses the test cases for which modified and certified program produced the same outputs. The idea is to avoid the cost of using new test cases and to avoid rerunning test cases that produce the same output.	Decrease the number of required test cases and consequently the cost and time of testing will be decreased.	Need to be examined on the fault detection capability and larger generated data and high complexity

Hybrid algorithm	Combine genetic algorithms and greedy algorithms	Provide significant reduction in the number of test cases and multi-objective optimization	High complexity
-------------------------	--	--	-----------------

The first algorithm we chose was **K-mean clustering**. It's the simplest clustering algorithm within its category, as well as being the best fit option for our project. For example, one of the other clustering algorithms is the Hierarchical Clustering Algorithm, which clusters the data in a hierarchical fashion unlike in K-mean, which simply clusters them into groups. In our project we did not care about the hierarchy of the test cases, furthermore, the hierarchical algorithm has a higher complexity and works for smaller datasets.

One of the other clustering algorithms that were mentioned in table 3 is Fuzzy logic. It was not chose because although convergence is always guaranteed, the process is very slow and this cannot be used for larger data.

As seen in the above table, some algorithms had a high complexity which made it less appealing to use, or they did not depend on the code coverage data for the reduction process which is a crucial metric for us.

One of the algorithms that we researched on was k-nearest neighbour (KNN) algorithm, we did not work with KNN because it's a supervised classification algorithm where grouping is done based on a prior class information, however, and in our project we did not have such information. When we look at the two possible options remaining in the above table which are Genetic and Greedy, we chose to work with **Greedy** and not Genetic. Our choice was based on the fact that Genetic algorithm needs to work with white-box testing, meaning it needed to access the test case source code in order to modify it to meet a certain reduction requirement.

6.3.2. K – Mean Clustering Algorithm

a) Introduction to Clustering Algorithms

Clustering is essentially an unsupervised learning method. An unsupervised learning method is one in which we draw references from datasets that only contain input data and no labelled responses. It is commonly used as a process to discover meaningful structure, explanatory underlying processes, generative features, and groupings inherent in a set of examples.

Clustering is the process of dividing a population or set of data points into groups so that data points in the same group are more similar to other data points in the same group and dissimilar to data points in other groups. It is essentially a collection of objects based on their similarity and dissimilarity.

For example, the data points in the graph below clustered together can be classified into one single group. We can distinguish the clusters, and we can identify that there are 3 clusters in the below picture.

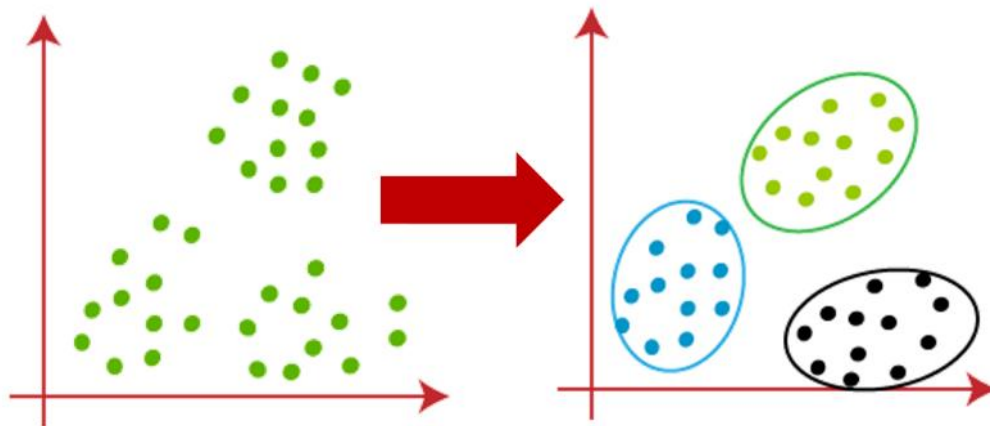


Figure 36: Clustering Illustration

Clustering is critical because it determines the intrinsic grouping of the unlabelled data present. There are no requirements for good clustering. It is up to the user to determine what criteria they will use to satisfy their needs. There are different types of clustering algorithms that handle all kinds of unique data.

Density-based

Data is grouped in density-based clustering by areas of high concentrations of data points surrounded by areas of low concentrations of data points. The algorithm basically finds places that

are dense with data points and labels them as clusters. Because these clustering algorithms do not attempt to assign outliers to clusters, they are ignored.

Distribution-based

A distribution-based clustering approach considers all data points to be members of a cluster based on the probability that they belong to a given cluster. It works like this: there is a centre-point, and the farther a data point is from the centre, the less likely it is to be part of that cluster.

Centroid-based

It's a little picky about the initial parameters you give it, but it's quick and efficient. These algorithms separate data points based on the presence of multiple centroids in the data. A cluster is assigned to each data point based on its squared distance from the centroid. This is the most common clustering method.

Hierarchical-based

On hierarchical data, hierarchical-based clustering is commonly used. It creates a tree of clusters to organise everything from the top down. This type of clustering is more restrictive than the others, but it is ideal for certain types of data sets.

K-means clustering

It is the most popular clustering algorithm. It is the simplest unsupervised learning algorithm and is centroid-based. The goal of this algorithm is to reduce the variance of data points within a cluster. It's also how most people become acquainted with unsupervised machine learning. Because it iterates over all of the data points, K-means is best used on smaller data sets. That means it will take longer to classify data points if the data set contains a large number of them.

How it Works:

First, Initialize K random centroids. You could pick K random data points and make those your starting points. Otherwise, you pick K random values for each variable. For every data point, look at which centroid is nearest to it. Using some sort of measurement like Euclidean or Cosine distance. Assign the data point to the nearest centroid. For every centroid, move the centroid to the average of the points assigned to that centroid. Repeat the last three steps until the centroid assignment no longer changes. Works Best on Numeric Data Since the k-means algorithm computes the distance between two points, you can't really do that with categorical (low, medium, high) variables. A simple workaround for multiple categorical variables is to calculate the percent of times each variable matches in comparison to the cluster centroid.

Advantages of K-means:

- It is very simple to implement.
- It is scalable to a huge data set and also faster to large datasets.
- It adapts the new examples very frequently.
- Generalization of clusters for different shapes and sizes.

b) Methodology

Given a program such as our web application Employee Manager or Coverage Analyzer from VIQ, a set of test cases is defined to test the program traces including instructions, lines, methods, branches, classes and the cyclomatic complexity. We use our tool to get the code coverage results from those test cases. In order to reduce the number of generated test cases according to their coverage, K-mean clustering is applied.

The software that we used to apply K mean clustering is called SPSS^[35]. SPSS is a powerful statistical software platform. It offers a user-friendly interface and a robust set of features that lets you quickly extract useful insights from your data. It is used by market researchers, health researchers, survey companies, marketing organizations, data miners and others. It applies k mean clustering to generate a decision metric such as distance or density.

The following figure shows the overall flow of our approach, it also includes the names of the software that were used to perform each step.

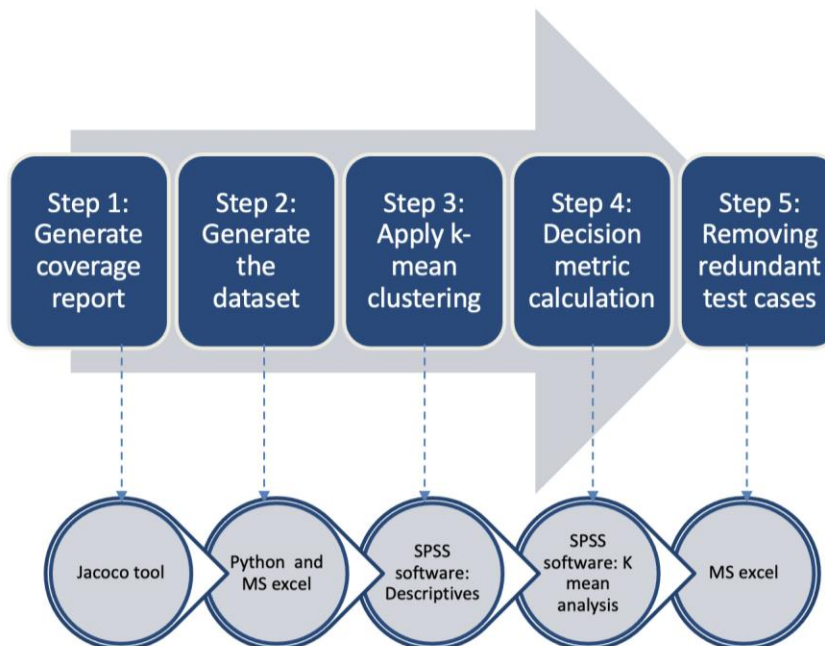


Figure 37: K-mean Process Flow

Step 1: Getting Code Coverage

Using our tool, we calculate the code coverage results for Coverage Analyzer backend and produce an HTML report containing all coverage data needed, it includes also the total values for each coverage result as seen in figure 38.

JaCoCo Coverage Report

Element	Missed Instructions	Cov.	Missed Branches	Cov.	Missed Cxty	Missed Lines	Missed Methods	Missed Classes						
com.mentor.dvt.coverageanalyzer.service.impl	52,011	10,648	16%	6,714	1,043	13%	4,945	5,711	9,989	12,227	1,269	1,718	15	82
com.mentor.dvt.coverageanalyzer.model	13,982	3,281	19%	1,794	272	13%	2,064	2,422	3,357	4,161	1,074	1,363	51	81
com.mentor.dvt.coverageanalyzer.jni.mapper	9,902	3,935	28%	1,209	301	19%	812	1,000	1,946	2,744	120	245	2	21
com.mentor.dvt.coverageanalyzer.service.dto	8,053	1,061	11%	767		7%	1,626	1,841	2,424	2,760	1,231	1,427	75	91
com.mentor.dvt.coverageanalyzer.service.mapper	7,449	1,206	13%	789	117	12%	583	711	1,674	1,930	162	258	18	88
com.mentor.dvt.coverageanalyzer.web.rest	2,913		14%	157		1%	252	304	548	654	172	224	2	29
com.mentor.dvt.coverageanalyzer.covergroups.service	2,401		1%	231		0%	177	185	449	461	59	67	0	5
com.mentor.dvt.shared.web.rest.util	2,230		12%	378		3%	250	270	451	504	52	70	7	12
com.mentor.dvt.coverageanalyzer.covergroups.analysis	1,880		0%	194		0%	163	163	360	360	66	66	6	6
com.mentor.dvt.coverageanalyzer.analysis.expressions	1,804	2,424	57%	324	256	44%	336	553	393	904	132	263	1	11
com.mentor.dvt.coverageanalyzer.domain	1,635		12%	121		0%	297	325	506	577	236	264	1	12
com.mentor.dvt.coverageanalyzer.domain.enumeration	1,623	2,336	59%	91		13%	109	155	193	335	48	89	9	25
com.mentor.dvt.coverageanalyzer.covergroups.exclusions	1,461		0%	80		0%	159	159	296	296	119	119	8	8
com.mentor.dvt.coverageanalyzer.license.mgls	1,459		5%	149		3%	158	167	347	368	80	89	7	9
com.mentor.dvt.coverageanalyzer.analysis.datamodel	1,420		0%	192		0%	161	161	331	331	65	65	1	1
com.mentor.dvt.coverageanalyzer.covergroups.domain	1,354		0%	132		0%	157	157	300	300	91	91	8	8
com.mentor.dvt.coverageanalyzer.interceptor	1,286		7%	293		2%	179	185	240	266	30	35	3	5
com.mentor.dvt.coverageanalyzer.service	931		6%	104		0%	66	68	180	184	14	16	0	2
com.mentor.dvt.coverageanalyzer.covergroups.service.dto	887		6%	110		0%	161	172	245	265	106	117	7	8
com.mentor.dvt.tpa.service	868		7%	128		3%	88	97	203	222	23	31	0	2
com.mentor.dvt.coverageanalyzer.license.salt	735		38%	95		24%	86	117	152	266	24	54	0	4
com.mentor.dvt.coverageanalyzer.constant	706		0%			n/a	7	7	122	122	7	7	6	6
com.mentor.dvt.coverageanalyzer.util	600	777	56%	70		38%	73	116	156	326	22	59	0	10
com.mentor.dvt.coverageanalyzer.exception	588		23%			28%	105	131	169	227	98	124	45	51
com.mentor.dvt.coverageanalyzer.shell	565		9%	65		9%	67	80	134	175	31	44	4	10
com.mentor.dvt.tpa.model			0%			0%	94	94	134	134	64	64	4	4
com.mentor.dvt.coverageanalyzer.web.resolver			93%			75%	1	5	1	12	0	3	0	1
com.mentor.dvt.coverageanalyzer.web.interceptor			0%			n/a	1	1	1	1	1	1	1	1
Total	125,504	of 158,345	20%	14,797	of 17,134	13%	13,925	16,392	26,848	33,569	5,830	7,663	327	712

Created with [JaCoCo](#) 0.8.7.202105040129

Figure 38: VIQ Backend Coverage Report to Extract Data

We generated an HTML report for **538 test cases** which is a full directory. These HTML files will be the source of data that we will create our dataset from.

Step 2: Creating the Dataset

The second step is the most important in our approach because it is the framework for the following phases. In order to build the dataset we need to select the most important and effective attributes for the test cases. The average cyclomatic complexity and the code coverage are the most two effective attributes in test case selection, so our dataset will contain the complexity and the coverage for each test case.

Cyclomatic complexity of a code section is the quantitative measure of the number of linearly independent paths in it. It is a software metric used to indicate the complexity of a program. It is computed using the Control Flow Graph of the program. The nodes in the graph indicate the smallest group of commands of a program, and a directed edge in it connects the two nodes.

Cyclomatic complexity is measured by counting the splitting nodes, in our case, our tool already calculates the cyclomatic complexity for the backend and it was not needed to calculate it manually. Cyclomatic complexity can make sure that every path have been tested at least once. And help to focus more on uncovered paths.

To begin building our dataset, we extracted the total values of data from the HTML report into an EXCEL file, then we computed the hits for each of the instructions, lines, methods, classes, branches, as the report contained the missed and the total of each attribute, and the missed cyclomatic complexity.

We created a python script that will extract the total values only from the HTML files. To write such script, we opened any HTML file and viewed its page source.

```

1 <?xml version="1.0" encoding="UTF-8"?><!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd"><html xmlns="http://www.w3.org/1999/xhtml" lang="en"><head>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8"/><link rel="stylesheet" href="jacoco-
resources/report.css" type="text/css"/><link rel="shortcut icon" href="jacoco-resources/report.gif"
type="image/gif"/><title>JaCoCo Coverage Report</title><script type="text/javascript" src="jacoco-
resources/report.js"></script></head><body onload="initialSort(['breadcrumb', 'coveragetable'])"><div
class="breadcrumb" id="breadcrumb"><span class="info"><a href="jacoco-sessions.html"
class="el_session">Sessions</a></span><span class="el_report">JaCoCo Coverage Report</span></div><h1>JaCoCo
Coverage Report</h1><table class="coverage" cellspacing="0" id="coveragetable"><thead><tr><td class="sortable"
id="a" onclick="toggleSort(this)">Element</td><td class="down sortable bar" id="b"
onclick="toggleSort(this)">Missed Instructions</td><td class="sortable ctr2" id="c"
onclick="toggleSort(this)">Cov.</td><td class="sortable bar" id="d" onclick="toggleSort(this)">Missed
Branches</td><td class="sortable ctr2" id="e" onclick="toggleSort(this)">Cov.</td><td class="sortable ctr1"
id="f" onclick="toggleSort(this)">Missed</td><td class="sortable ctr2" id="g"
onclick="toggleSort(this)">Cxt</td><td class="sortable ctr1" id="h" onclick="toggleSort(this)">Missed</td><td
class="sortable ctr2" id="i" onclick="toggleSort(this)">Lines</td><td class="sortable ctr1" id="j"
onclick="toggleSort(this)">Missed</td><td class="sortable ctr2" id="k" onclick="toggleSort(this)">Methods</td><td
class="sortable ctr1" id="l" onclick="toggleSort(this)">Missed</td><td class="sortable ctr2" id="m"
onclick="toggleSort(this)">Classes</td></tr></thead><tfoot><tr><td><b>Total</b></td><td class="bar">125,504 of
158,345</td><td class="ctr2">20%</td><td class="bar">14,797 of 17,134</td><td class="ctr2">13%</td><td
class="ctr1">13,925</td><td class="ctr2">16,392</td><td class="ctr1">26,848</td><td class="ctr2">33,569</td><td
class="ctr1">5,830</td><td class="ctr2">7,663</td><td class="ctr1">327</td><td class="ctr2">712</td></tr></tfoot>
</tbody><tr><td id="a37"><a href="com.mentor.dvt.coverageanalyzer.service.impl/index.html"

```

Figure 39: Coverage Report Source Page to Extract Total Results

We used the following function in the parsing script to extract the required coverage numbers

```

def parsefile (file_html):
    html = open(file_html,"r")
    reading = html.read()
    content = str(reading)
    info = re.findall(r"Total.*?</tfoot>",>content)
    info_rep = info[0].replace(',','')
    coverageN = re.findall(r"\d+",>info_rep)
    return coverage

```

Then, we write these numbers in an excel sheet by subtracting the missed number of lines, instructions, etc. from the total values to get the number of hits. The cyclomatic complexity is

taken as it is, indicating the missed complexity, which means the less complexity, the better the test case is. This dataset excel sheet is the input to the next step.

for i **in** files:

list = parsefile(i)

if (list):

coverageN.append(list)

worksheet.write(c+1,0,os.path.basename(os.path.dirname(i)))

worksheet.write(c+1,1, int(coverageN[c][9]))

worksheet.write(c+1,2, int(coverageN[c][1])-int(coverageN[c][0]))

worksheet.write(c+1,3, int(coverageN[c][15])-int(coverageN[c][13]))

worksheet.write(c+1,4, int(coverageN[c][19])-int(coverageN[c][17]))

worksheet.write(c+1,5, int(coverageN[c][23])-int(coverageN[c][21]))

worksheet.write(c+1,6, int(coverageN[c][5])-int(coverageN[c][4]))

	A	B	C	D	E	F	G	H	I	J
1	Testcase	Cxty	Inst	Line	Meth	Classes	Branch			
2	breadCrur	13925	32841	6721	1833	385	2337			
3	breadCrur	13715	34779	7199	2149	414	2070			
4	breadCrur	13849	34002	6957	1860	385	2446			
5	breadCrur	13787	34716	7101	1876	386	2569			
6	branchesS	14540	26051	5293	1530	362	1535			
7	branchesS	14488	27033	5471	1528	368	1595			
8	branchesS	14280	29764	6063	1670	376	1917			
9	branchesS	14211	30814	6288	1733	384	1926			
10	codeEdito	13787	34762	7103	1878	385	2574			
11	codeEdito	13926	33750	6870	1829	383	2388			
12	codeEdito	13791	34754	7077	1855	383	2611			
13	codeEdito	13864	34239	6967	1844	384	2499			
14	comment:	14146	30894	6286	1745	379	2057			
15	condition:	14020	33005	6737	1816	383	2249			
16	condition:	14210	30422	6168	1720	379	1959			
17	condition:	13925	33885	6903	1838	384	2419			
18	condition:	14326	28853	5891	1694	376	1751			
19	expressio	14130	31244	6332	1745	379	2091			
20	expressio	14333	29065	5921	1669	377	1787			

Figure 40: Extracted Data in EXCEL File Format

Step 3: Applying K-mean Clustering

In this step, we will apply K clustering machine learning algorithm using the previous data. We import the excel sheet to a software called SPSS.

Firstly, we perform descriptive analysis to calculate the z scores of the attributes which is needed SPSS to apply the k mean analysis

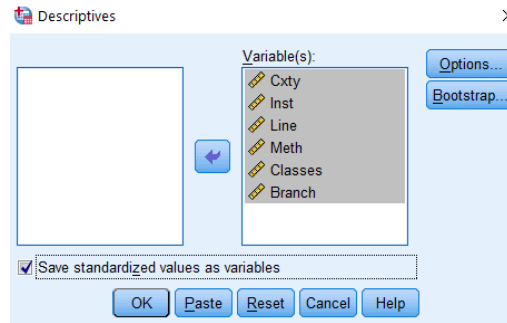


Figure 41: Descriptive Analysis to Compute Z-scores for K-mean Analysis in SPSS

ZCxty	ZInst	ZLine	ZMeth	ZClasses	ZBranch
-58432	53639	.51382	.37087	.32830	.84436
-.86259	.72964	.74082	.95076	.64694	.52320
-.68503	.65216	.62590	.42042	.32830	.97547
-.76719	.72335	.69428	.44978	.33929	1.12342
.23061	-.14068	-.16431	-.18516	.07559	-.12033
.16171	-.04276	-.07978	-.18883	.14152	-.04815
-.11391	.22956	.20135	.07175	.22942	.33916
-.20534	.33426	.30820	.18736	.31732	.34999
-.76719	.72794	.69523	.45345	.32830	1.12943
-.58300	.62703	.58458	.36353	.30633	.90570
-.76189	.72714	.68288	.41124	.30633	1.17394
-.66515	.67579	.63064	.39106	.31732	1.03922
-.29147	.34224	.30725	.20938	.26238	.50756
-.45844	.55274	.52142	.33967	.30633	.73851
-.20667	.29518	.25121	.16351	.26238	.38968
-.58432	.64049	.60025	.38005	.31732	.94299
-.05296	.13872	.11967	.11579	.22942	.13949
-.31268	.37714	.32909	.20938	.26238	.54846

Figure 42: Results of Descriptive Analysis

Secondly, we apply the analysis by choosing the number of clusters and the number of iterations. The clusters centres are randomly selected, and the new centres are recalculated every iteration. We choose 3 clusters based on a previously made research paper^[32] and 12 iterations as they were sufficient for convergence after trials.

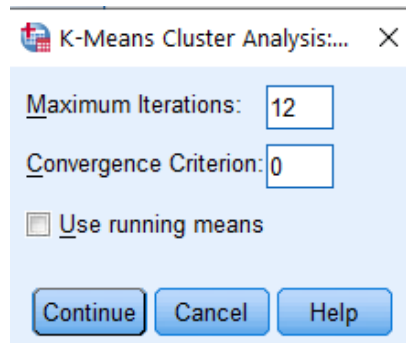


Figure 43: K-mean Cluster Analysis in SPSS

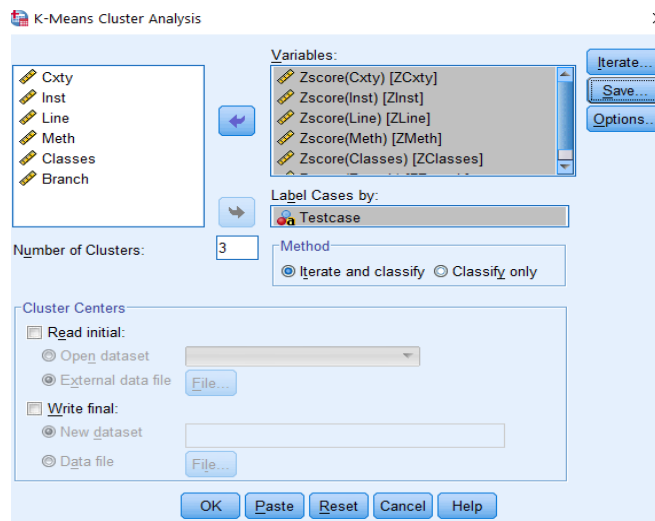


Figure 44: K-mean Cluster Analysis in SPSS

The final cluster centers are based on the values of Zscores for example, test cases which have a large positive Zscore for the cyclomatic complexity are placed in the third clusters as shown in the following figure.

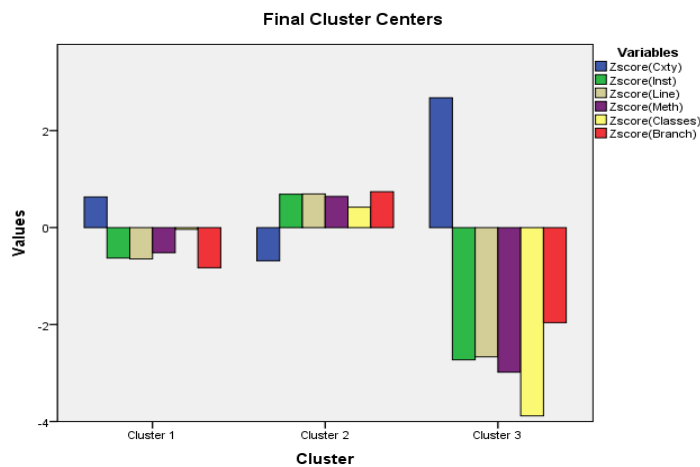


Figure 45: Bar Graph Showing Clustering Criteria

The following figures shows the distance of each test cases to the center of the clusters which will be our decision metric in the next step and the cluster which each test case belongs to.

	Cxty	Inst	Line	Meth	Classes	Branch	QCL_1	Distance	va
	13925	32841	6721	1833	385	2337	2	40308	
	13715	34779	7199	2149	414	2070	2	47673	
	13849	34002	6957	1860	385	2446	2	34473	
	13787	34716	7101	1876	386	2569	2	44161	
	14540	26051	5293	1530	362	1535	1	1.10618	
	14488	27033	5471	1528	368	1595	1	1.26366	
	14280	29764	6063	1670	376	1917	2	1.14926	
	14211	30814	6288	1733	384	1926	2	94327	
	13787	34762	7103	1878	385	2574	2	44766	
	13926	33750	6870	1829	383	2388	2	38260	
	13791	34754	7077	1855	383	2611	2	50838	
	13864	34239	6967	1844	384	2499	2	40869	
	14146	30894	6286	1745	379	2057	2	83902	
	14020	33005	6737	1816	383	2249	2	45820	
	14210	30422	6168	1720	379	1959	2	98620	
	13925	33885	6903	1838	384	2419	2	37834	
	14326	28853	5891	1694	376	1751	2	1.31455	
	14130	31244	6332	1745	379	2091	2	79376	
	14333	29065	5921	1669	377	1787	2	1.30258	
	14389	28203	5738	1618	375	1708	2	1.49771	
	13976	33083	6749	1811	383	2308	2	43839	

Figure 46: Results of K-mean Analysis - Distance from Cluster Center & Cluster Membership

Step 4: Decision Metric

K-mean Algorithm works on partitioning of a given data set into groups or clusters to maximize the intra cluster similarity, each test case within the cluster displays the same behavior.

After clustering the Euclidian distance that measure the distance between any given test case and the centroid is calculated.

$$dist = \sqrt{\sum_{i=1}^n (db_i - c_j)^2}$$

Where C_j is the cluster centre and db_i is the data point of each test case

Along the distance from the centre of cluster, we also consider the cyclomatic complexity as a decision metric in the removal of the redundant test cases.

Step 5: Removing redundant test cases

A test cases t_i is considered redundant with t_j if $dist(t_i, centroid) \approx dist(t_j, centroid)$, in which t_i, t_j belong to the same cluster. In this case, pick the test case with minimum Cyclomatic Complexity. Test cases in the same cluster with an approximately equal distance are considered redundant because since they exhibit the same behavior, they are expected to give the same code coverage results.

For example, the following figure shows two cases, first case we have two test cases (test21 and test21_1), they have approximately equal distances. In this case, we remove the test case with higher cyclomatic complexity which is test21.

The second case is having two test cases (test14 and test14_1) with the exact same distance and complexity. In this case, one of the test cases is chosen randomly.

240	test21	13793	35896	7425	2058	398	2293	2	0.27158
241	test21_1	13792	35917	7431	2058	398	2297	2	0.27551
242	test14	13772	35711	7417	2068	399	2280	2	0.27858
243	test14_1	13772	35711	7417	2068	399	2280	2	0.27858
244	test13	13772	35715	7417	2068	399	2282	2	0.27904

Figure 47: Sample of Redundant Test Cases

c) Result Analysis

By applying the reduction, we obtained 250 test cases out of the original 538 test cases. To check the effectiveness of our approach, we recomputed the code coverage percentage with the new set of test cases. The reduced list achieved Total Code Coverage of 71 % while the original list had 73%.

This method is statistical, it doesn't guarantee that important test cases won't be removed as it doesn't depend on a specific set of requirements. It only ensures a reduced list of test cases with an acceptable code coverage percentage compared to the original list. This reduction will reduce the cost and the time required for running the regression.

With that being said, it's to be noted that even though it's a statistical approach, it has a very important advantage which is not needing an access to the source code. Therefore, it's totally black box method.

6.3.3. Greedy Algorithm

a) Introduction to Greedy algorithm

The Greedy algorithm is one of the most common machine learning algorithms. It has many types depending on a given problem, such as selection sort, knapsack problem, Set cover problem, and minimum spanning tree. And each type works in a certain way. However, the general idea is that it builds up a solution piece by piece, always choosing the next piece that offers the most obvious and immediate benefit.

The general steps of Greedy algorithm work by recursively constructing a set of objects from the smallest possible constituent parts applying the following steps:

- To begin with, the solution set (containing answers) is empty.
- At each step, an item is added to the solution set until a solution is reached.
- If the solution set is feasible, the current item is kept.
- Else, the item is rejected and never considered again.

It is quite easy to come up with a greedy algorithm (or even multiple greedy algorithms) for a problem. Analysing the run time for greedy algorithms will generally be much easier than for other techniques (like Divide and conquer). For the Divide and conquer technique, it is not clear whether the technique is fast or slow. This is because at each level of recursion the size of gets smaller and the number of sub-problems increases.

Our problem here is similar to Set cover problem.

In the set cover problem, we are given a universe U of n elements, a collection of subsets of U say $S = \{S_1, S_2, \dots, S_m\}$ we are searching for the minimum subset that covers all the elements of U . It starts forming the new subset by selecting the set that covers the largest number of elements in U which are the requirements. Then, it loops on the other sets and choose the next set that has the most elements which are not covered in the subset.

Example:

$U = \{1, 2, 3, 4, 5\}$
$S = \{S_1, S_2, S_3\}$
$S_1 = \{4, 1, 3\},$
$S_2 = \{2, 5\},$
$S_3 = \{1, 4, 3, 2\},$

The Output of set cover is $\{ S_2, S_3 \}$

b) Methodology

In the previous approach, we only took the number hits from the code coverage report. This approach is a white box one where we want to make use of the information we have on the source code. The input we have is the files generated from HTML reports, which we extracted before, using a parsing script. Each test case has a file that contains the file name of the source code and the lines hit by the test case.

Having this input, we consider the lines numbers as the list of requirements, and we are trying to reach the minimum set of test cases that satisfy this list. Therefore, our problem is similar to the set cover problem. The following format is the input format of the set cover greedy algorithm^[34] which is a predefined open-source C++ algorithm.

Number of requirements Number of sets
A line for each set indicating which requirements from the list are satisfied by this set

For example, if we have 10 requirements and 5 test cases the input format should be like the following

```
10 5
5 7 8 9 10
5 7
1 3 6 10
4 6 7 9 10
1 2 8 10
```

When we run the set cover algorithm the expected output is:

```
Set id 4 Element: 4, 6, 7, 9, 10,
Set id 5 Element: 1, 2, 8,
Set id 1 Element: 5,
Set id 3 Element: 3,
```

Where sets {4, 5, 1, 3}, which represents the test case number, satisfy all the requirements.

Now, we will apply on coverage analyzer App to generate the reduced list. The first step is using the parsed file to generate the requirement list which includes the different line hits for each test case by making a python script.

Then, we we generate the text file including each test case and which requirement it satisfies in the same format as the previous example.

This part generates the requirement list

for i in files:

 contentinfo = openfile(i)

 contentinfo=contentinfo.split(", ")

for j in contentinfo:

 contentinfo[c] = re.findall(r"[*?]",contentinfo[c])

if contentinfo[c] not in Req_list:

 Req_list.append(contentinfo[c])

```

    c=c+1
    print(dc)
    dc=dc+1
    c=0

```

```

["webapp/app/app.animations.ts:4"], ["webapp/app/app.constants:1"], ["webapp/app/app.constants:2"],
["webapp/app/app.constants:3"], ["webapp/app/app.constants:4"], ["webapp/app/app.module.ts:75"],
["webapp/app/app.route.ts:5"], ["webapp/app/polyfills.ts:1"], ["webapp/app/polyfills.ts:2"],
["webapp/app/polyfills.ts:4"], ["webapp/app/assertions-directives/assertions-directives.component.ts:33"],
["webapp/app/assertions-directives/assertions-directives.component.ts:1482"],
["webapp/app/assertions-directives/assertions-directives.module.ts:28"],
["webapp/app/assertions-directives/assertions-directives.route.ts:5"],
["webapp/app/assertions-directives/assertions-directives.service.ts:13"],
["webapp/app/assertions-directives/assertions-directives.service.ts:27"],
["webapp/app/assertions-directives/assertions-directives.service.ts:28"],
["webapp/app/assertions-directives/assertions-directives.service.ts:36"],
["webapp/app/assertions-directives/assertions-directives.service.ts:42"],
["webapp/app/assertions-directives/assertions-directives.service.ts:43"],
["webapp/app/assertions-directives/assertions-directives.service.ts:44"],
["webapp/app/assertions-directives/assertions-directives.service.ts:45"],
["webapp/app/assertions-directives/assertions-directives.service.ts:46"],
["webapp/app/assertions-directives/assertions-directives.service.ts:47"],
["webapp/app/assertions-directives/assertions-directives.service.ts:54"],
["webapp/app/assertions-directives/assertions-directives.service.ts:479"],
["webapp/app/blocks/config/uib-pagination.config.ts:6"],
["webapp/app/blocks/config/uib-pagination.config.ts:13"],
["webapp/app/blocks/interceptor/auth.interceptor.ts:4"],
["webapp/app/blocks/interceptor/auth.interceptor.ts:7"],
["webapp/app/blocks/interceptor/auth.interceptor.ts:15"],
["webapp/app/blocks/interceptor/auth.interceptor.ts:17"],
["webapp/app/blocks/interceptor/http.provider.ts:30"], ["webapp/app/blocks/interceptor/http.provider.ts:37"],

```

Figure 48: Requirement List

And then we generate a test data file which is in the correct format using the previous output

for i in files:

```

contentinfo2 = openfile(i)
contentinfo2=contentinfo2.split(", ")

```

for j in contentinfo2:

```

    contentinfo2[c] = re.findall(r"[*?]",contentinfo2[c])
    index = find_element_in_list(contentinfo2[c], Req_list)
    datafile.write(str(index+1)+' ')
    c=c+1
c=0
datafile.write('/n')

```

The test data file contains the total number of lines hit by all test cases which is 24175 and the total number of test cases in the regression which is 538. And a line for each test case stating the requirements satisfied by it.

```

1 24175 538
2 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41
42 43 44 45 46 47 48 49 50 51 52 53 54 55 56 57 58 59 60 61 62 63 64 65 66 67 68 69 70 71 72 73 74 75 76 77 78 79
80 81 82 83 84 85 86 87 88 89 90 91 92 93 94 95 96 97 98 99 100 101 102 103 104 105 106 107 108 109 110 111 112
113 114 115 116 117 118 119 120 121 122 123 124 125 126 127 128 129 130 131 132 133 134 135 136 137 138 139 140
141 142 143 144 145 146 147 148 149 150 151 152 153 154 155 156 157 158 159 160 161 162 163 164 165 166 167 168
169 170 171 172 173 174 175 176 177 178 179 180 181 182 183 184 185 186 187 188 189 190 191 192 193 194 195 196
197 198 199 200 201 202 203 204 205 206 207 208 209 210 211 212 213 214 215 216 217 218 219 220 221 222 223 224
225 226 227 228 229 230 231 232 233 234 235 236 237 238 239 240 241 242 243 244 245 246 247 248 249 250 251 252
253 254 255 256 257 258 259 260 261 262 263 264 265 266 267 268 269 270 271 272 273 274 275 276 277 278 279 280

```

Figure 49: Input to setcover Greedy Algorithm

The next step is running the set cover algorithm to get the output. We ran the code on Ubuntu to get the executable file. The output will be the minimum set of test cases that satisfy all the requirements as mentioned before. We used the following command

```

make
./setcover testdatafile.txt

```

The following figure shows the output

```

set id 157 Element: 15159,
set id 328 Element: 19951,
set id 286 Element: 19874,
set id 122 Element: 14355,
set id 75 Element: 13521,
set id 388 Element: 20676,
set id 504 Element: 23612,
set id 342 Element: 20178,
set id 115 Element: 14572,
set id 412 Element: 21163,
set id 208 Element: 17921,
set id 40 Element: 10655,
set id 1 Element: 4558,
set id 266 Element: 19744,
set id 367 Element: 20514,
set id 347 Element: 20216,

```

Figure 50: Snippet from the Output of setcover

Finally, we create a python script that maps the ids to the test cases paths

```

def parsefile (text_file):
    reducedFile = open(text_file,"r")
    reading = reducedFile.read()
    content = str(reading)
    indexLine = re.findall(r"set id.*?:",content)
    index = []
    c=0
    for i in indexLine:
        index.append(re.findall(r"\d+",i))
        c = c+1
    return index

```

Now we have the reduced list of test cases with the correct path in the directory that cover the requirements.

```

1 ca/Exclusion/dashBoardNavigation/coverage.json
2 ca/func_coverage/assertions/assertionsExclusions_dashBoard11/coverage.json
3 ca/code_coverage/patternsMatching/patternsExclusion_TreeNode3/coverage.json
4 ca/func_coverage/transitive/transitive3/coverage.json
5 ca/PA/sanityPA/coverage.json
6 ca/Exclusion/view_applied_exclusions/test_2_import_exclusions_appy_view_available_exclusions/coverage.json
7 ca/testAnalysis/TestsRanking/newConfigRankTestCovMetrics/coverage.json
8 ca/testPlanAuthor/excludeCodeCov/coverage.json
9 ca/code_coverage/fsm/fsmExclusion_dashBoard/coverage.json
10 ca/Exclusion/test46/coverage.json
11 ca/func_coverage/holeAnalysis/excludeHole2/coverage.json
12 ca/code_coverage/toggles/togglesExclusion_checkUCDB1/coverage.json
13 ca/Exclusion/import_exclusions/test_9_1/coverage.json
14 ca/summaryDashBoard/globalSearch/searchInstances/coverage.json
15 ca/PA/PA_Design_Dashboard/Include_Exclude_Icons/coverage.json
16 ca/testAnalysis/rankTests2/coverage.json
17 ca/Exclusion/expTreeNode/coverage.json
18 ca/code_coverage/misc/mulInstances_heatMap2/coverage.json
19 ca/Exclusion/VM-10586_broken_heatmap_distribution/coverage.json
20 ca/testPlanAuthor/coverageFilters3/coverage.json
21 ca/PA/Exclusion_Dashboard/Right_Click_Context_Menu/coverage.json
22 ca/func_coverage/holeAnalysis/singleCross2/coverage.json
23 ca/code_coverage/fsm/fsmExclusion_fsm1/coverage.json
24 ca/func_coverage/assertions/assertionsExclusions_checkUCDB/coverage.json
25 ca/testAnalysis/testHitData12/coverage.json
26 ca/Exclusion/import_exclusions/test_14_Dashboard_filter_button/coverage.json
27

```

Figure 51: Snippet from the Reduced List

c) Result Analysis

To sum up our work in this section we compared code coverage results before and after the reduction. The result states that we reduced the test cases list from 538 to **196** and this Reduced list achieved **73%** Backend code coverage which is the same as before reduction. It also achieved **72.44%** frontend code coverage which is very close to the result before reduction (72.5%).

The following figure shows the Backend code coverage after our reduction:

	99%	n/a	1	7	1	122	1	7	0	6
	93%	75%	1	5	1	12	0	3	0	1
	0%	n/a	1	1	1	1	1	1	1	1
.mapper	99%	95%	1	18	0	63	0	7	0	3
	<u>42,508 of 158,345</u> 73%	6,937 of 17,134 59%	6,527	16,392	9,234	33,569	1,754	7,663	105	712

Figure 52: Backend Coverage after setcover Greedy Algorithm

Table 2: Summary of Coverage Results

	Before Reduction	After Reduction
Number of Test Cases	538	196
Backend Code Coverage	73%	73%
Frontend Code Coverage	72.5%	72.44%

The following table compares between the results of K mean clustering and Greedy algorithm:

Table 3: Comparison between Greedy & K mean clustering

	K mean clustering	Greedy
Access to source code	Does not need access	Needs access to source code to generate requirement list
Suite after reduction	Reduced to 250	Reduced to 196
Coverage Percentages	Reduced by 2 %	Maintain same code cover

6.4. Future work and conclusion

The future work regarding the K-mean, we can apply the same methodology to frontend as we have the required code coverage percentages and we only need to calculate the cyclomatic complexity, this can be obtained from open source packages such as ts-complex. To validate the efficiency and effectiveness of our methodology we can test it on different larger applications. In this approach we only included the code coverage percentages so to increase the quality we can add other decision metrics such as lines hit by each test case.

To conclude, today software development and testing are dominated not by the design of new software, but by the rework and maintenance of existing software. Such software maintenance activities may account for as much as two-thirds of the overall cost of software production therefore test reduction, test prioritization, etc. are important applications. Hence, code coverage data can also be a useful metric for those applications not only in determining the code coverage for a specific test suite.

7. Scalability of the Tool

Since our project was mainly tailored for angular frontend and java spring boot backend web applications, we checked the scalability of the solution if it can be further used for other applications.

7.1. Frontend Scalability

For the frontend we found that Istanbul-nyc works with any web application that is using TypeScript/JavaScript, for example React, Vue.js, Aurelia. However, there are some limitations which will be based on the framework itself like finding an option that is similar `--source-map` in other frameworks. For example, React uses the command **npm run build** to build the frontend. The environment variable `GENERATE_SOURCEMAP=true` by default in CRA. That means once you build the generated folder there will be extra ".map" files generated. It will look something like this:

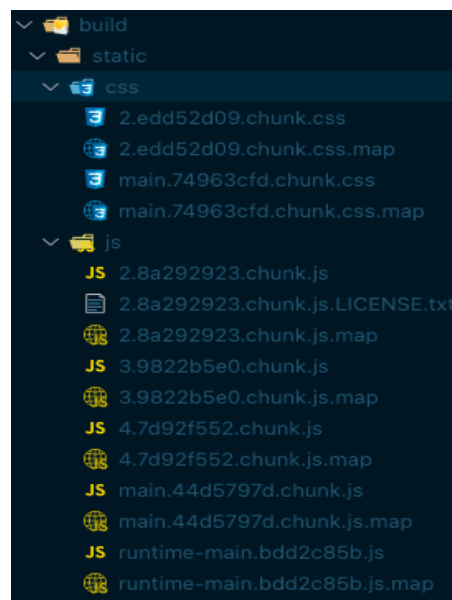


Figure 53: Files included in React

7.2. Backend Scalability

JaCoCo tool we used will work with any web application that is using Java or Java like syntax such as Groovy in the backend. So, Frameworks like Claris FileMaker, OutSystems and G2 Deals should be compatible. Accordingly, the only limitation is if the application's backend uses a language different from Java such as Python.

7.3. Test Automation Tool

Any test automation tool can be used as long as it imitates the user's interaction. The only limitation here is to make sure the code snippet that generates the **coverage.json** file needed for the frontend is written in the same language used for the tests.

7.4. Test Suite Reduction

Any python script can parse the needed input using the code coverage results from our tool, and can then proceed to generate the reduced list based on the used algorithm.

8. Overhead Calculations

There are several types for overhead like:

- Disk space
- Build time
- Memory consumption
- Runtime

8.1. Disk Space

To calculate this overhead, we built the war file once with instrumentation and another time without instrumentation and compared the size. We found that:

- Size without instrumentation: 270 MB
- Size with instrumentation 291 MB

Also, the coverage data occupies 61MB in disk space for each test case. Therefore, the overhead in disk space is minor.

8.2. Build Time

To calculate this, we built the war file once with instrumentation and another time without instrumentation and found that:

- Build time without instrumentation 2 min. 34 sec.
- Build time with instrumentation 3 min. 34 sec.

Therefore, the overhead of the build time is also minor. Also, the overhead of the build time is not crucial as we only build the war file once.

8.3. Memory Consumption

Table 4: Memory Consumption Comparison

Type	Memory	Time
With Coverage	14736112K	455 seconds
Without Coverage	14716760K	342 seconds

This difference in memory is 19.352 MB which is negligible compared to the 14 GB used

8.4. Runtime

This is the main overhead that concerns us as this will affect each and every run. In addition, time is the main metric knowing that memory is not an issue as they have very powerful machines with large capacities.

We ran the test cases multiple times with the instrumented war file and other times with non-instrumented war file and calculated the difference in run times. We found that with instrumentation the run time is doubled.

The first thought that comes to mind is that this overhead is not acceptable, but in code coverage industry we can see that this overhead is actually very satisfying. For example, LCOV, which is the other code coverage tool that Mentor uses, gives an overhead of X5 to X10 of the runtime.

Conclusion

In this project we demonstrated the importance of software testing and its applications by measuring code coverage using Selenium black box testing on a web application. We measured the code coverage using Istanbul framework in the frontend and JaCoCo in the backend. After validating our results, we started deploying on VIQ Coverage Analyzer from Siemens, accordingly, any needed adjustments in our tool were made to accommodate the environment of VIQ.

We successfully measured the code coverage results for Coverage Analyzer web app, and we validated these results with test automation team. The next step was creating an HTML page that summarized the coverage results for both frontend and backend, as well as allowing the user to navigate the results.

We also demonstrated how we can use the coverage data to reduce large test suites using K-mean clustering algorithm as well as Greedy algorithm. We applied these approaches on Coverage Analyzer, which resulted in reducing the test suite to less than half its original size, while maintaining approximately the same coverage percentages. We researched for the possibility of finding other packages that give the same service but we didn't find any. Therefore, our package is exclusive in regards what it does and its results.

References

1. Spring Initializr. (n.d.). Retrieved September 15, 2021, from <https://start.spring.io/>
2. JavaScript with syntax for types. TypeScript. (n.d.). Retrieved September 15, 2021, from <https://www.typescriptlang.org/>
3. Angular. (n.d.). Retrieved September 15, 2021, from <https://angular.io/>
4. Angular. (n.d.). Retrieved September 15, 2021, from <https://angular.io/guide/ngmodules>
5. Node.js. (n.d.). About. Node.js. Retrieved September 15, 2021 from <https://nodejs.org/en/about/>
6. API testing client that flows with you. HTTPie. (n.d.). Retrieved September 22, 2021 from <https://httpie.io/>
7. Selenium. (n.d.). Retrieved October 1, 2021, from <https://www.selenium.dev/>
8. Patro, S. (2020, December 30). ngWebDriver - a way to automate angular apps in selenium using java. Qavalidation. Retrieved October 1, 2021, from <https://qavalidation.com/2017/10/ngwebdriver-way-automateangular-apps-selenium-using-java.html/>
9. Selvanathan, K. (2020, May 19). ngWebdriver overview - QAFox. QAFox | The Easiest Tutorial Site on Software Testing. Retrieved October 2, 2021, from <https://www.qafox.com/ngwebdriver-overview/>
10. *What is Cors (cross-origin resource sharing)? tutorial & examples: Web security academy.* What is CORS (cross-origin resource sharing)? Tutorial & Examples | Web Security Academy. (n.d.). Retrieved October 14, 2021, from <https://portswigger.net/web-security/cors>
11. NPM. npm. (n.d.). Retrieved November 5, 2021, from <https://www.npmjs.com/>
12. Istanbuljs. (n.d.). *Istanbuljs/NYC: The Istanbul Command Line Interface.* GitHub. Retrieved November 12, 2021, from <https://github.com/istanbuljs/nyc>
13. Remap-istanbul. npm. (n.d.). Retrieved December 5, 2021, from <https://www.npmjs.com/package/remap-istanbul>
14. What is instrumentation in nyc istanbul? (2019, September 24). Stack Overflow. Retrieved November 12, 2021, from <https://stackoverflow.com/questions/58075076/what-is-instrumentation-in-nyc-istanbul>

15. Priya, Jaya, Jaya, Lucky, Thakar, R., Swathi, Sunitha, Krishna, Negma, Zahrani, S., Kumar, P., S, P., Artem, Sfdsdfsdfdfdfs, Luba, Monkeypants, M., Wadyalkar, S., MEntor, Q. A., Aschwanden, P., ... Skok, S. (2022, May 5). What is regression testing? definition, tools, method, and example. Software Testing Help. Retrieved Retrieved June 17, 2022, from <https://www.softwaretestinghelp.com/regression-testing-tools-and-methods/>
16. *GCC, the GNU compiler collection*. GCC, the GNU Compiler Collection - GNU Project. (n.d.). Retrieved June 17, 2022, from <https://gcc.gnu.org/>
17. Lcov. Codecov. (2021, July 15). Retrieved June 17, 2022, from <https://about.codecov.io/tool/lcov/>
18. Porter, B., Zyl, J. van, & Lamy, O. (n.d.). *Welcome to Apache Maven*. Maven. Retrieved June 17, 2022, from <https://maven.apache.org/>
19. Casey, J. (2005, June 24). Introduction to maven plugin development. Maven. Retrieved June 18, 2022, from <https://maven.apache.org/guides/introduction/introduction-to-plugins.html>
20. Codeception. (n.d.). Retrieved June 17, 2022, from <https://codeception.com/docs/11-Codecoverage>
21. Protractor. (n.d.). Retrieved June 17, 2022, from <https://protractor.angular.io/#:~:text=Protractor%20is%20made%20specifically%20for,setup%20effort%20on%20your%20part>
22. *Oracle Linux 6: Porting guide*. Moved. (2021, March 31). Retrieved June 17, 2022, from <https://docs.oracle.com/en/operating-systems/oracle-linux/6/porting/ch02s05s01.html>
23. *Emma code coverage - intellij IDEs plugin: Marketplace*. JetBrains Marketplace. (n.d.). Retrieved January 3, 2022, from <https://plugins.jetbrains.com/plugin/103-emma-code-coverage>
24. *Gradle Build Tool*. Gradle. (n.d.). Retrieved February 19, 2022, from from <https://gradle.org/>
25. JavaScript ES5. (n.d.). Retrieved June 17, 2022, from https://www.w3schools.com/js/js_es5.asp
26. *Webpack*. webpack. (n.d.). Retrieved Retrieved November 28, 2021, from from <https://webpack.js.org/>
27. What will happen if sourcemap is set as false in Angular. (2019, February 26). Stack Overflow. <https://stackoverflow.com/questions/54879588/what-will-happen-if-sourcemap-is-set-asfalse-in-angular>

28. Getting Code Coverage for e2e tests run on a Java codebase. (2021, April 13). Ankeetmaini.Dev. Retrieved December 3, 2021, <https://ankeetmaini.dev/posts/getting-e2e-tests-coverage-for-java/>
29. Raika, J. (2019, September 9). JaCoCo End-to-End Code Coverage at Runtime. Dzone.Com. Retrieved December 3, 2021, from <https://dzone.com/articles/code-coverage-report-generator-for-java-projects-a>
30. JaCoCo- Command Line Interface. Jacoco.Org. Retrieved December 3, 2021, from <https://www.jacoco.org/jacoco/trunk/doc/cli.html>
31. Java code coverage with Jacoco. Merge exec files collected from different application versions. (2019, July 4).Stack Overflow. Retrieved December 3, 2021, from <https://stackoverflow.com/questions/56891525/java-code-coverage-with-jacoco-merge-exec-files-collected-from-different-applic>
32. Saifan, Ahmad & Alsukhni, Emad & Alawneh, Hanadi & Sbaih, Ayat. (2016). Test Case Reduction Using Data Mining Technique. International Journal of Software Innovation. 4. 56-70. 10.4018/IJSI.2016100104.
33. Alian, Marwah & Suleiman, Dima & Shaout, Adnan. (2016). Test Case Reduction Techniques - Survey. International Journal of Advanced Computer Science and Applications. 7. 264-275. 10.14569/IJACSA.2016.070537.
34. Martin-Steinegger. (n.d.). *Martin-Steinegger/Setcover: Linear (time,space) greedy set cover implementation*. GitHub. Retrieved from <https://github.com/martin-steinegger/setcover>
35. SPSS software. IBM. (n.d.). Retrieved June 18, 2022, from <https://www.ibm.com/analytics/spss-statistics-software>
36. What is postman? Postman API Platform. (n.d.). Retrieved June 18, 2022, from <https://www.postman.com/product/what-is-postman/>

Appendix

Appendix A: Code

Employee manager web application code snippets

Employee resources – request mapping

```
package tech.getarrays.employeeManager;

import org.springframework.http.HttpStatus;
import org.springframework.http.ResponseEntity;
import org.springframework.transaction.annotation.Transactional;
import org.springframework.web.bind.annotation.*;
import tech.getarrays.employeeManager.model.Employee;
import tech.getarrays.employeeManager.service.EmployeeService;

import java.io.Console;
import java.util.List;

@RestController
@RequestMapping("/employee")
public class EmployeeResource {
    private final EmployeeService employeeService;

    public EmployeeResource(EmployeeService employeeService) {
        this.employeeService = employeeService;
    }

    @GetMapping("/all")
    public ResponseEntity<List<Employee>> getAllEmployees (){
        List<Employee> employees = employeeService.findAllEmployees();
        return new ResponseEntity<>(employees, HttpStatus.OK);
    }

    @GetMapping("/find/{id}")
    public ResponseEntity<Employee> getEmployeeById (@PathVariable("id") Long id){
        Employee employee = employeeService.findEmployeeById(id);
        return new ResponseEntity<>(employee, HttpStatus.OK);
    }

    @PostMapping("/add")
    public ResponseEntity<Employee> addEmployee(@RequestBody Employee employee){
        Employee newEmployee = employeeService.addEmployee(employee);
        return new ResponseEntity<>(newEmployee, HttpStatus.CREATED);
    }

    @PutMapping("/update")
    public ResponseEntity<Employee> updateEmployee(@RequestBody Employee employee){
        Employee updateEmployee = employeeService.updateEmployee(employee);
```

```
        return new ResponseEntity<>(updateEmployee, HttpStatus.OK);
    }
    @Transactional
    @DeleteMapping("/delete/{id}")
    public ResponseEntity<?> deleteEmployee(@PathVariable("id") Long id) {
        employeeService.deleteEmployee(id);
        System.out.println(id);
        return new ResponseEntity<>(HttpStatus.OK);
    }
}
```

Employee model

```
package tech.getarrays.employeeManager.model;

import javax.persistence.*;
import java.io.Serializable;
@Entity
public class Employee implements Serializable {
    @Id
    @GeneratedValue(strategy = GenerationType.AUTO)
    @Column(nullable = false, updatable = false)
    private long id;
    private String name;
    private String email;
    private String jobTitle;
    private String phone;
    private String imageUrl;
    @Column(nullable = false, updatable = false)
    private String employeeCode;
    public Employee() {}
    public Employee(String name, String email, String jobTitle, String phone, String imageUrl, String
employeeCode){
        this.name = name;
        this.email = email;
        this.jobTitle = jobTitle;
        this.phone = phone;
        this.imageUrl = imageUrl;
        this.employeeCode = employeeCode;
    }
    public long getId(){
        return this.id;
    }
    public void setId(long id){
        this.id = id;
    }

    public String getName() {
```

```
    return name;
}
public void setName(String name){
    this.name = name;
}

public String getEmail() {
    return email;
}

public void setEmail(String email) {
    this.email = email;
}

public String getEmployeeCode() {
    return employeeCode;
}

public void setEmployeeCode(String employeeCode) {
    this.employeeCode = employeeCode;
}

public String getImageUrl() {
    return imageUrl;
}

public void setImageUrl(String imageUrl) {
    this.imageUrl = imageUrl;
}

public String getJobTitle() {
    return jobTitle;
}

public void setJobTitle(String jobTitle) {
    this.jobTitle = jobTitle;
}

public String getPhone() {
    return phone;
}

public void setPhone(String phone) {
    this.phone = phone;
}
@Override
public String toString()
{
    return "Employee{" +
```



```
        "id="+ id +
        ", name='"+ name +\'"+
        ", email='"+ email +\'"+
        ", jobTitle='"+ jobTitle +\'"+
        ", phone='"+ phone +\'"+
        ", imageUrl='"+ imageUrl +\'"+
        '});
    }
}
```

Communication with the database – Employee services

```
package tech.getarrays.employeeManager.service;
```

```
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.beans.factory.parsing.EmptyReaderEventListener;
import org.springframework.stereotype.Service;
import tech.getarrays.employeeManager.exception.UserNotFoundException;
import tech.getarrays.employeeManager.model.Employee;
import tech.getarrays.employeeManager.repo.EmployeeRepo;
```

```
import java.util.List;
import java.util.UUID;
```

```
@Service
public class EmployeeService {
    private final EmployeeRepo employeeRepo;
    @Autowired
    public EmployeeService(EmployeeRepo employeeRepo) {
        this.employeeRepo = employeeRepo;
    }
    public Employee addEmployee(Employee employee){
        employee.setEmployeeCode(UUID.randomUUID().toString());
        return employeeRepo.save(employee);
    }
    public List<Employee> findAllEmployees() {
        return employeeRepo.findAll();
    }
    public Employee updateEmployee(Employee employee) {
        return employeeRepo.save(employee);
    }
    public Employee findEmployeeById(Long id){
```

```

        return employeeRepo.findEmployeeById(id).
            orElseThrow(() -> new UserNotFoundException("user by id "+id+" was not found"));
    }
    public void deleteEmployee(Long id) {
        employeeRepo.deleteEmployeeById(id);
    }
}

```

Frontend: app component's functions handling the user actions

```

import { HttpResponse } from '@angular/common/http';
import { EmitterVisitorContext } from '@angular/compiler';
import { Component, OnInit } from '@angular/core';
import { NgForm } from '@angular/forms';
import { Employee } from './employee';
import { EmployeeService } from './employee.service';

```

```

@Component({
  selector: 'app-root',
  templateUrl: './app.component.html', // One page
  styleUrls: ['./app.component.css']
})
export class AppComponent implements OnInit {
  employees! : Employee[] ;
  editEmployee?:Employee;
  deleteEmployee?:Employee;
  employeesToBeSearched! : Employee[] ;
  key! : string;

  constructor(private employeeService: EmployeeService) {
  }
  ngOnInit(){
    this.getEmployees();
  }

  public getEmployees(): void {
    this.employeeService.getEmployees().subscribe(
      (response: Employee[]) => {
        this.employees = response;
        this.employeesToBeSearched = this.employees;
      },
      (error : HttpResponse) => {
        alert(error.message);
      }
    );
  }
}

```

```

    }
  );
}
public onAddEmployee(addForm: NgForm):void {
  document.getElementById("add-employee-form")?.click();
  this.employeeService.addEmployee(addForm.value).subscribe(
    (response:Employee)=>{
      console.log(response);
      this.getEmployees();
      this.employeesToBeSearched = this.employees;
      addForm.reset();
    },
    (error: HttpResponse) => {
      alert(error.message);
      addForm.reset();
    }
  );
}
public onUpdateEmployee(employee: Employee):void {
  this.employeeService.updateEmployee(employee).subscribe(
    (response:Employee)=>{
      console.log(response);
      this.getEmployees();
      this.employeesToBeSearched = this.employees;
    },
    (error: HttpResponse) => {
      alert(error.message);
    }
  );
}
public onDeleteEmployee(employeeId?: number):void {
  this.employeeService.deleteEmployee(employeeId).subscribe(
    (response:void)=>{
      console.log(response);
      this.getEmployees();
      this.employeesToBeSearched = this.employees;
    },
    (error: HttpResponse) => {
      alert(error.message);
    }
  );
}
public searchEmployees(key: string): void {
  console.log(key); // hit
  const results: Employee[] = []; //hit
  this.key = key; // hit
  for (const employee of this.employeesToBeSearched ) { // hit
    if (employee.name.toLowerCase().indexOf(key.toLowerCase()) !== -1 // hit
    || employee.email.toLowerCase().indexOf(key.toLowerCase()) !== -1

```

```

    || employee.phone.toLowerCase().indexOf(key.toLowerCase()) !== -1
    || employee.jobTitle.toLowerCase().indexOf(key.toLowerCase()) !== -1) {
      results.push(employee); // hit
    }
  }
  this.employees = results; // hit
  if (!key) { // hit
    this.getEmployees(); // hit
  }
}
public onOpenModal(employee?: Employee, mode?: string): void {
  const container = document.getElementById('main-container');
  const button = document.createElement('button');
  button.type = 'button';
  button.style.display = 'none';
  button.setAttribute('data-toggle', 'modal');
  if (mode === 'add') {
    button.setAttribute('data-target', '#addEmployeeModal');
  }
  if (mode === 'edit') {
    this.editEmployee = employee;
    button.setAttribute('data-target', '#updateEmployeeModal');
  }
  if (mode === 'delete') {
    this.deleteEmployee = employee;
    button.setAttribute('data-target', '#deleteEmployeeModal');
  }
  container?.appendChild(button);
  button.click();
}
}

```

Frontend: Services that construct the http requests

```

import { Injectable } from '@angular/core';
import { Observable } from 'rxjs';
import { HttpClient, HttpClientModule } from '@angular/common/http';
import { Employee } from './employee';
import { environment } from 'src/environments/environment';
@Injectable({
  providedIn: 'root'
})
export class EmployeeService {
  private apiUrl= environment.apiUrl;

  constructor(private http: HttpClient) { }
  public getEmployees (): Observable<Employee[]> {
    return this.http.get<Employee[]>(`${this.apiUrl}/employee/all`);
  }
}

```

```

    }

    public addEmployee (employee: Employee): Observable<Employee> {
    return this.http.post<Employee>(`${this.apiUrl}/employee/add`,employee);
    }

    public updateEmployee (employee: Employee): Observable<Employee> {
    return this.http.put<Employee>(`${this.apiUrl}/employee/update`,employee);
    }

    public deleteEmployee (id?: number): Observable<void> {
    return this.http.delete<void>(`${this.apiUrl}/employee/delete/${id}`);
    }
}

```

Frontend: HTML components

```

<nav class="navbar navbar-expand-lg navbar-dark bg-dark">
  <a class="navbar-brand" style="color:white;">Employee Manager</a>
  <button class="navbar-toggler" type="button" data-toggle="collapse" data-target="#navbarColor02" aria-
controls="navbarColor02" aria-expanded="false" aria-label="Toggle navigation">
  <span class="navbar-toggler-icon"></span>
</button>
<div class="collapse navbar-collapse" id="navbarColor02">
  <ul class="navbar-nav mr-auto">
    <li class="nav-item active">
      <a class="nav-link" (click)="onOpenModal(undefined, 'add')">Add Employee <span class="sr-
only">(current)</span></a>
    </li>
  </ul>
  <form class="form-inline my-2 my-lg-0">
    <input type="search" (ngModelChange)="searchEmployees(key.value)" #key="ngModel" ngModel
name="key" id="searchName" class="form-control mr-sm-2" placeholder="Search employees..."
required>
  </form>
</div>
</nav>
<div class="container" id="main-container">
  <div class="row">
    <div *ngFor="let employee of employees" class="col-md-6col-xl-3">
      <div class="card m-b-30">
        <div class="card-body row">
          <div class="col-6">
            <a href=""></a>
          </div>
          <div class="col-6 card-title align-self-center mb-0">
            <h5>{{employee?.name}}</h5>
            <p class="m-0">{{employee?.jobTitle}}</p>

```

```

        </div>
    </div>
    <ul class="list-group list-group-flush">
        <li class="list-group-item"><i class="fa fa-envelope float-right"></i>Email : <a
href="#">{{employee?.email}}</a></li>
        <li class="list-group-item"><i class="fa fa-phone float-right"></i>Phone
:{{employee?.phone}}</li>
    </ul>
    <div class="card-body">
        <div class="float-right btn-group btn-group-sm">
            <a (click)="onOpenModal(employee, 'edit')" class="btn btn-primary tooltips" data-
placement="top" data-toggle="tooltip" data-original-title="Edit"><i class="fa fa-pencil"></i> </a>
            <a (click)="onOpenModal(employee, 'delete')" class="btn btn-secondary tooltips" data-
placement="top" data-toggle="tooltip" data-original-title="Delete"><i class="fa fa-times"></i></a>
        </div>
        <ul class="social-links list-inline mb-0">
            <li class="list-inline-item"><a title="" data-placement="top" data-toggle="tooltip"
class="tooltips" href="" data-original-title="Facebook"><i class="fa fa-facebook-f"></i></a></li>
            <li class="list-inline-item"><a title="" data-placement="top" data-toggle="tooltip"
class="tooltips" href="" data-original-title="Twitter"><i class="fa fa-twitter"></i></a></li>
            <li class="list-inline-item"><a title="" data-placement="top" data-toggle="tooltip"
class="tooltips" href="" data-original-title="Skype"><i class="fa fa-skype"></i></a></li>
        </ul>
    </div>
</div>
</div>
</div>
</div>
</div>
<!-- Add Employee Modal -->
<div class="modal fade" id="addEmployeeModal" tabindex="-1" role="dialog" aria-
labelledby="addEmployeeModalLabel" aria-hidden="true">
    <div class="modal-dialog" role="document">
        <div class="modal-content">
            <div class="modal-header">
                <h5 class="modal-title" id="addEmployeeModalLabel">Add Employee</h5>
                <button type="button" class="close" data-dismiss="modal" aria-label="Close">
                    <span aria-hidden="true">&times;</span>
                </button>
            </div>
            <div class="modal-body">
                <form #addForm="ngForm" (ngSubmit)="onAddEmployee(addForm)">
                    <div class="form-group">
                        <label for="name">Name</label>
                        <input type="text" ngModel name="name" class="form-control" id="name" placeholder="Name"
required>
                    </div>
                    <div class="form-group">
                        <label for="email">Email Address</label>

```

```

        <input type="email" ngModel name="email" class="form-control" id="email" placeholder="Email"
required>
    </div>
    <div class="form-group">
        <label for="phone">Job title</label>
        <input type="text" ngModel name="jobTitle" class="form-control" id="jobTitle" placeholder="Job
title" required>
    </div>
    <div class="form-group">
        <label for="phone">Phone</label>
        <input type="text" ngModel name="phone" class="form-control" id="phone" placeholder="Phone"
required>
    </div>
    <div class="form-group">
        <label for="phone">Image URL</label>
        <input type="text" ngModel name="imageUrl" class="form-control" id="imageUrl"
placeholder="Image URL" required>
    </div>
    <div class="modal-footer">
        <button type="button" id="add-employee-form" class="btn btn-secondary" data-
dismiss="modal">Close</button>
        <button [disabled]="addForm.invalid" type="submit" class="btn btn-primary" >Save
changes</button>
    </div>
</form>
</div>
</div>
</div>
</div>
</div>
</div>
<!-- Edit Modal -->
<div class="modal fade" id="updateEmployeeModal" tabindex="-1" role="dialog" aria-
labelledby="employeeEditModalLabel" aria-hidden="true">
    <div class="modal-dialog" role="document">
        <div class="modal-content">
            <div class="modal-header">
                <h5 class="modal-title" id="updateEmployeeModalLabel">Edit Employee {{editEmployee?.name}}
</h5>
                <button type="button" class="close" data-dismiss="modal" aria-label="Close">
<span aria-hidden="true">&times;</span>
                </button>
            </div>
            <div class="modal-body">
                <form #editForm="ngForm" >
                    <div class="form-group">
                        <label for="name">Name</label>
                        <input type="text" ngModel="{{editEmployee?.name}}" name="name" class="form-control"
id="name" aria-describedby="emailHelp" placeholder="Name">
                    </div>

```

```

    <input type="hidden" ngModel="{{editEmployee?.id}}" name="id" class="form-control" id="id"
placeholder="Email">
    <input type="hidden" ngModel="{{editEmployee?.employeeCode}}" name="userCode" class="form-
control" id="userCode" placeholder="Email">
    <div class="form-group">
        <label for="email">Email Address</label>
        <input type="email" ngModel="{{editEmployee?.email}}" name="email" class="form-control"
id="email" placeholder="Email">
    </div>
    <div class="form-group">
        <label for="phone">Job title</label>
        <input type="text" ngModel="{{editEmployee?.jobTitle}}" name="jobTitle" class="form-control"
id="jobTitle" placeholder="Job title">
    </div>
    <div class="form-group">
        <label for="phone">Phone</label>
        <input type="text" ngModel="{{editEmployee?.phone}}" name="phone" class="form-control"
id="phone" name="phone" placeholder="Phone">
    </div>
    <div class="form-group">
        <label for="phone">Image URL</label>
        <input type="text" ngModel="{{editEmployee?.imageUrl}}" name="imageUrl" class="form-
control" id="imageUrl" placeholder="Image URL">
    </div>
    <div class="modal-footer">
        <button type="button" id="" data-dismiss="modal" class="btn btn-secondary">Close</button>
        <button (click)="onUpdateEmployee(editForm.value)" data-dismiss="modal" class="btn btn-
primary" >Save changes</button>
    </div>
</form>
</div>
</div>
</div>
</div>
</div>
<!-- Delete Modal -->
<div class="modal fade" id="deleteEmployeeModal" tabindex="-1" role="dialog" aria-
labelledby="deleteModelLabel" aria-hidden="true">
    <div class="modal-dialog" role="document">
        <div class="modal-content">
            <div class="modal-header">
                <h5 class="modal-title" id="deleteModelLabel">Delete Employee</h5>
                <button type="button" class="close" data-dismiss="modal" aria-label="Close">
<span aria-hidden="true">&times;</span>
                </button>
            </div>
            <div class="modal-body">
                <p>Are you sure you want to delete employee {{deleteEmployee?.name}}</p>
            <div class="modal-footer">
                <button type="button" class="btn btn-secondary" data-dismiss="modal">No</button>

```



```

        <button (click)="onDeleteEmployee(deleteEmployee?.id)" class="btn btn-danger" data-
dismiss="modal">Yes</button>
    </div>
</div>
</div>
</div>
</div>
<!-- Notification for no employees -->
<div *ngIf="employees?.length == 0" class="col-lg-12 col-md-12 col-xl-12">
    <div class="alert alert-info" role="alert">
        <h4 class="alert-heading">NO EMPLOYEES!</h4>
        <p>No Employees were found.</p>
    </div>
</div>

```

Selenium test cases code

```

package level2;
import java.nio.file.Files;
import java.nio.file.Paths;
import java.util.List;
import java.util.concurrent.TimeUnit;

import org.openqa.selenium.*;
import org.openqa.selenium.By;
import org.openqa.selenium.Keys;
import org.openqa.selenium.WebDriver;
import org.openqa.selenium.firefox.FirefoxDriver;
import org.openqa.selenium.JavascriptExecutor;
import org.openqa.selenium.interactions.Action;
import org.openqa.selenium.interactions.Actions;
import org.openqa.selenium.interactions.internal.MouseAction.Button;

//import com.google.common.io.Files;
import com.google.gson.Gson;
import com.google.gson.GsonBuilder;
import com.paulhammant.ngwebdriver.*;
public class Test3 {
    public static void main(String[] args) {
        try {
            int passedTestsCount=0;
            int failedTestsCount=0;
            // flush all // Noisy //
            System.setProperty("webdriver.gecko.driver","C:\\geckodriver.exe");
            WebDriver driver = new FirefoxDriver();
            JavascriptExecutor jsDriver = (JavascriptExecutor) driver;

            NgWebDriver ngdriver= new NgWebDriver (jsDriver);

```

```

driver.manage().deleteAllCookies();
driver.manage().window().maximize();
driver.manage().timeouts().implicitlyWait(30, TimeUnit.SECONDS);
driver.manage().timeouts().pageLoadTimeout(30, TimeUnit.SECONDS);
driver.get("http://localhost:3000/");
ngdriver.waitForAngularRequestsToFinish();
Thread.sleep(1000);
System.out.print("===== Source code as
follows =====\n");
System.out.print( driver.getPageSource()+"\n"); // --> File (splitting 10 functions)
System.out.print("===== Source code ended
===== \n");
System.out.print("===== Element Source code
as follows =====\n");
System.out.print( driver.findElement(By.className("nav-link")).getAttribute("outerHTML")+"\n");
System.out.print("===== Source code ended
===== \n");
System.out.print("=====Add test
started===== \n");
System.out.print( driver.findElement(By.className("navbar")).getAttribute("innerHTML"));
driver.findElement(By.className("nav-link")).click(); // loop 10 functions --> function.Hit=true ..
function.Hit=false --> true
driver.findElement(By.id("name")).sendKeys("Loay Samy");
Thread.sleep(500);
driver.findElement(By.id("email")).sendKeys("Loaysamy13@yahoo.com");
Thread.sleep(500);
driver.findElement(By.id("jobTitle")).sendKeys("Hunter");
Thread.sleep(500);
driver.findElement(By.id("phone")).sendKeys("0123456789");
Thread.sleep(500);
driver.findElement(By.id("imageUrl")).sendKeys("../assets/images/Loay.jpg");
Thread.sleep(500);
driver.findElement(By.xpath("//button[normalize-space()='Save changes']")).click();
Thread.sleep(500);

if(driver.findElement(By.xpath("//h5[text() = 'Loay Samy']"))!=null)
{
System.out.print("Test Passed \n");
passedTestsCount++;
}else {
System.out.print("Test failed \n");
failedTestsCount++;
}
System.out.print("=====Add test
ended===== \n");
System.out.print("=====Edit test
started===== \n");
Thread.sleep(500);

```

```

driver.findElements(By.xpath("//i[@class='fa fa-pencil']")).get(5).click();
Thread.sleep(500);
driver.findElement(By.xpath("//div[@id='updateEmployeeModal']/input[@id='name']")).clear();
Thread.sleep(500);
driver.findElement(By.xpath("//div[@id='updateEmployeeModal']/input[@id='name']")).sendKeys("Loay
Sam");
Thread.sleep(500);
driver.findElement(By.xpath("//div[@id='updateEmployeeModal']/button[normalize-space()='Save
changes']")).click();
Thread.sleep(500);
if(driver.findElement(By.xpath("//h5[text() = 'Loay Sam']"))!=null)
{
    System.out.print("Test Passed \n");
    passedTestsCount++;
}
else {
    System.out.print("Test failed \n");
    failedTestsCount++;
}
System.out.print("=====Edit test
ended=====\n");

System.out.print("=====delete test
started=====\n");
Thread.sleep(500);
driver.findElements(By.xpath("//i[@class='fa fa-times']")).get(5).click();
Thread.sleep(500);
driver.findElement(By.xpath("//div[@id='deleteEmployeeModal']/button[normalize-
space()='Yes']")).click();
Thread.sleep(500);
if(driver.findElements(By.xpath("//i[@class='fa fa-pencil']")).size()==5)
{
    System.out.print("Test Passed \n");
    passedTestsCount++;
}
else {
    System.out.print("Test failed \n");
    failedTestsCount++;
}
System.out.print("=====delete test
ended=====\n");

System.out.print("=====Search tests
started=====\n");

driver.findElement(By.id("searchName")).sendKeys("L");
Thread.sleep(500);
if(driver.findElements(By.xpath("//i[@class='fa fa-pencil']")).size()==3)
{
    System.out.print("Test Passed \n");
    passedTestsCount++;
}
else {

```

```

System.out.print("Test failed \n");
failedTestsCount++;
}
driver.findElement(By.id("searchName")).sendKeys("o");
Thread.sleep(500);
if(driver.findElements(By.xpath("//i[@class='fa fa-pencil']")).size()==1)
{
System.out.print("Test Passed \n");
passedTestsCount++;
}
else {
System.out.print("Test failed \n");
failedTestsCount++;
}
driver.findElement(By.id("searchName")).sendKeys("s");
Thread.sleep(500);
if(driver.findElement(By.xpath("//h4[text() = 'NO EMPLOYEES!']"))!=null)
{
System.out.print("Test Passed \n");
passedTestsCount++;
}
else {
System.out.print("Test failed \n");
failedTestsCount++;
}
driver.findElement(By.id("searchName")).sendKeys(Keys.BACK_SPACE);
Thread.sleep(500);
driver.findElement(By.id("searchName")).sendKeys(Keys.BACK_SPACE);
Thread.sleep(500);
if(driver.findElements(By.xpath("//i[@class='fa fa-pencil']")).size()==3)
{
System.out.print("Test Passed \n");
passedTestsCount++;
}
else {
System.out.print("Test failed \n");
failedTestsCount++;
}
driver.findElement(By.id("searchName")).sendKeys(Keys.BACK_SPACE);
Thread.sleep(500);
if(driver.findElements(By.xpath("//i[@class='fa fa-pencil']")).size()==5)
{
System.out.print("Test Passed \n");
passedTestsCount++;
}
else {
System.out.print("Test failed \n");
failedTestsCount++;
}
}
System.out.print("=====Search tests
ended=====\n");

System.out.print(passedTestsCount+" tests passed\n");

```

```

    System.out.print(failedTestsCount+" tests failed\n");
    Object str = jsDriver.executeScript("return window._coverage_");
    GsonBuilder builder = new GsonBuilder();
    Gson gson = builder.create();
    String Coverage = gson.toJson(str);
    Files.write(Paths.get("C:\\Users\\loay
samy\\AngularApp\\employeemanagerapp\\.nyc_output\\coverage.json"),Coverage.getBytes());
    // driver.close();

} catch (Exception e) {
    // TODO Auto-generated catch block
    e.printStackTrace();
}

}

}
}

```

Deployment on VIQ Coverage Analyzer

Merge frontend coverage

```

#open(PASSN,'>', "./file");
print"Merging frontend started\n";
$results=/home/lelmasry/Development/nodeJS/node-v10.16.3-linux-x64/bin/istanbul-merge --out
merged.json `cat Jsonfiles.list`;
print "$? \n";
while($? ne "0"){rm file;open(PASSN,'>', "./file");
print PASSN $results;
$lineTodelete = `tail -1 file | sed 's,.*ca,ca,g`;
print "$lineTodelete is corrupted";
chomp($lineTodelete);
#print "echo -n '$lineTodelete' | sed 's,/,\\,g";#$line = `echo -n '$lineTodelete' | sed 's,/,\\\\\\\\\\,g' | sed 's,\\,.,g`;
$line = `echo -n $lineTodelete | sed 's,\\,\\,g`;
#quotemeta($lineTodelete);#print $line;#print "`sed 's,.*$line,,g' -i file`;print"`sed 's,$line,,g' -i Jsonfiles.list`;
sed 's,$line,,g' -i Jsonfiles.list;
$results=/home/lelmasry/Development/nodeJS/node-v10.16.3-linux-x64/bin/istanbul-merge --out
merged.json `cat Jsonfiles.list`;close(PASSN);
print "$? \n";
}#print"`sed '/^\\$/d' file`;

```

Merge BackEnd Coverage

```
#open(PASSN,>', "./file");
print "Merging backend started\n";
$results=java -jar $ENV{"JARS"}/org.jacoco.cli-0.8.7-nodeps.jar merge \`cat ExecFiles.list\` --destfile
merged.exec`;
print "$? \n";
while($? ne "0"){ rm file; open(PASSN,>', "./file");
print PASSN $results;
$lineTodelete = tail -1 file | sed 's,.*ca,ca,g^';
print "$lineTodelete is corrupted";
chomp($lineTodelete);#print "echo -n '$lineTodelete' | sed 's,/,\\,g''";#line = `echo -n '$lineTodelete' | sed 's,/,\\\\\\\\\\\\,g'
| sed 's,\\,.,g^';
$line = echo -n '$lineTodelete' | sed 's,\\,.,g^';
#quotemeta($lineTodelete);#print $line;#print "`sed 's,.*$line,,g' -i file`";#print "`sed 's,$line,,g' -i ExecFiles.list`";
sed 's,$line,,g' -i ExecFiles.list`;
$results=java -jar $ENV{"JARS"}/org.jacoco.cli-0.8.7-nodeps.jar merge \`cat ExecFiles.list\` --destfile
merged.exec`; close(PASSN);
print "$? \n";
}print "`sed '/^$d' file`";
```

Wrapper

```
foreach e (cat Jsonfiles.list | sed 's,/coverage.json,,g') ##Jsonfiles_red.list
cd $e;
echo $e;

/zin/tools/python/3.6.3/bin/python /home/lelmasry/parse_all.py "./FrontendCoverage" "./BackendCoverage";
|
cd -;
end
```

Parsing script to generate ln_cov.info file which contains both frontend and backend

```
import sys
import os
import re
from pathlib import Path

def parsefile_front (filename):
    html = open(filename,"r")
    content = str(html.read())
    #getting lines
```

```

lines = re.findall(r"<span class=\"cline-any cline.*?span>",content)
coverage=[] index=0 #getting hits
for i in lines:
    if ("-yes" in i):
        hits = re.findall(r"\b\d+\b",i)
        coverage.append({"index":index+1,"hits":int(hits[0])}) index+=1
return coverage

```

```

def parsefile_back (filename):
    html = open(filename,"r")
    reading = html.read()
    content = str(reading)
    #getting lines
    lines = re.findall(r"<span class=\"fc\".*?>",content)
    coverage=[]
    for i in lines:
        index = re.findall(r"\d+",i)
        coverage.append(int(index[0]))
    return coverage

```

```

def generate_file_list(pathlist,option):
    filelist=[]
    if (option=="front"):
        for i in pathlist_front:
            coverage=parsefile_front(i)
            if (coverage):
                filelist.append(i) else:
        for i in pathlist_back:
            coverage=parsefile_back(i)
            if (coverage):
                filelist.append(i)
    return filelist

```

```

def write_front(filelist,info):
    for j in filelist:
        filename = re.findall(r".*?.ts",str(j))
        name = re.sub(r'FrontendCoverage/', "", filename[0])
        coverage = parsefile_front(j)
        for i in coverage:
            if (j==filelist[-1] and i==coverage[-1]):
                info.write("\n"+name+": "+str(i["index"])+"\n": "+str(i["hits"]))
            else:
                info.write("\n"+name+": "+str(i["index"])+"\n": "+str(i["hits"]), ")
    return

```

```

def write_back(filelist,info):
    for j in filelist:
        filename = re.findall(r".*?.java",str(j))

```

```

    name = re.sub(r'BackendCoverage/', "", filename[0])
    coverage = parsefile_back(j)
    for i in coverage:
        info.write("\n"+name+": "+str(i)+"\n": "+"1")    return
path_dir_front = sys.argv[1]
path_dir_back = sys.argv[2]
pathlist_front = Path(path_dir_front).glob('*/.ts.html')
pathlist_back = Path(path_dir_back).glob('*/.java.html')

filelist_front = generate_file_list(pathlist_front,"front")
filelist_back = generate_file_list(pathlist_back,"back")

info = open("ln_cov.info","w")
info.write("{}")write_front(filelist_front,info)write_back(filelist_back,info)info.write("{}")info.close()

```

Test suite reduction applications

kmeanParse:

```

import sys
import os
import re
from operator import itemgetter
from pathlib import Path
import xlswriter

path_dir=sys.argv[1]
excelfile=sys.argv[2]

def parsefile (file_html):
    html = open(file_html,"r")
    reading = html.read()
    content = str(reading)
    info = re.findall(r"Total.*?</tfoot>",content)
    info_rep = info[0].replace(',','')
    95overage = re.findall(r"\d+",info_rep)
    return 95overage

files = Path(path_dir).glob('**/*index.html')

workbook = xlswriter.Workbook(excelfile)
worksheet = workbook.add_worksheet()

worksheet.write('A1', 'Testcase')

```



```
worksheet.write('B1','Cxyt')
worksheet.write('C1','Inst')
worksheet.write('D1','Line')
worksheet.write('E1','Meth')
worksheet.write('F1','Classes')
worksheet.write('G1','Branch')
```

```
96coverage=[]
c=0
for i in files:
    list = parsefile(i)
    if (list):
        96coverage.append(list)

worksheet.write(c+1,0,os.path.basename(os.path.dirname(i)))
worksheet.write(c+1,1, int(96coverage[c][9]))
worksheet.write(c+1,2, int(96coverage[c][11])-int(96coverage[c][0]))
worksheet.write(c+1,3, int(96coverage[c][15])-int(96coverage[c][13]))
worksheet.write(c+1,4, int(96coverage[c][19])-int(96coverage[c][17]))
worksheet.write(c+1,5, int(96coverage[c][23])-int(96coverage[c][21]))
worksheet.write(c+1,6, int(96coverage[c][5])-int(96coverage[c][4]))

c=c+1

workbook.close()
```

GreedyParse:

```
import sys
import os
import re
from operator import itemgetter
from pathlib import Path

path_dir=sys.argv[1]

def openfile (infofile):
    info = open(infofile,"r")
    reading = info.read()
    content = str(reading)
    return content

def find_element_in_list(element, list_element):
    index_element = list_element.index(element)
    return index_element
```

```
files = Path(path_dir).glob('**/*In_cov.info')

Req_list=[]
c=0
dc=1
for i in files:
    contentinfo = openfile(i)
    contentinfo=contentinfo.split(", ")
    for j in contentinfo:
        contentinfo[c] = re.findall(r"[*?]",contentinfo[c])
        if contentinfo[c] not in Req_list:
            Req_list.append(contentinfo[c])
        c=c+1
    print(dc)
    dc=dc+1
    c=0
output=open("output.txt","w")
output.write(str(Req_list))
output.close
datafile=open("testdatafile.txt","w")

datafile.write(str(len(Req_list))+ ' '+str(dc-1))
datafile.write('\n')

c=0
files = Path(r'C:/Users/anod/Desktop/GP/list/ca').glob('**/*In_cov.info')

for i in files:
    contentinfo2 = openfile(i)
    contentinfo2=contentinfo2.split(", ")

    for j in contentinfo2:
        contentinfo2[c] = re.findall(r"[*?]",contentinfo2[c])
        index = find_element_in_list(contentinfo2[c], Req_list)
        datafile.write(str(index+1)+' ')
        c=c+1
    c=0
    datafile.write('\n')

datafile.close
```

GreedyReducedList:

```
import sys
import os
import re
```

```
from operator import itemgetter
from pathlib import Path
import xlswriter

path_dir=sys.argv[1]
reducedList=sys.argv[2]

# text_file is the reduced list that came from setcover (testfiledata)
def parsefile (text_file):
    reducedFile = open(text_file,"r")
    reading = reducedFile.read()
    content = str(reading)
    indexLine = re.findall(r"set id.*?:",content)
    index = []
    c=0
    for i in indexLine:
        index.append(re.findall(r"\d+",i))
        c = c+1
    return index

files = Path(path_dir).glob('**/*In_cov.info')

totalTestcases = []
c = 0
for i in files:
    totalTestcases.append(os.path.dirname(i))

index = parsefile(reducedList)

reducedFile = open("greedyReducedList.txt","w")
for j in index:
    reducedFile.write(totalTestcases[int(j[0])+"\coverage.json")
    reducedFile.write("\n")

reducedFile.close
```

Appendix B: Licenses

Istanbul

ISC License

Copyright (c) 2015, Contributors

Permission to use, copy, modify, and/or distribute this software for any purpose with or without fee is hereby granted, provided that the above copyright notice and this permission notice appear in all copies.

THE SOFTWARE IS PROVIDED "AS IS" AND THE AUTHOR DISCLAIMS ALL WARRANTIES

WITH REGARD TO THIS SOFTWARE INCLUDING ALL IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS. IN NO EVENT SHALL THE AUTHOR BE LIABLE FOR ANY SPECIAL, DIRECT, INDIRECT, OR CONSEQUENTIAL DAMAGES OR ANY DAMAGES WHATSOEVER RESULTING FROM LOSS OF USE, DATA OR PROFITS,

WHETHER IN AN ACTION OF CONTRACT, NEGLIGENCE OR OTHER TORTIOUS ACTION,

ARISING OUT OF OR IN CONNECTION WITH THE USE OR PERFORMANCE OF THIS SOFTWARE.

JaCoCo

Copyright © 2009, 2022 Mountainminds GmbH & Co. KG and Contributors

The JaCoCo Java Code Coverage Library and all included documentation is made available by Mountainminds GmbH & Co. KG, Munich. Except indicated below, the Content is provided to you under the terms and conditions of the Eclipse Public License Version 2.0 ("EPL"). A copy of the EPL is [provided](#) with this Content and is also available at <https://www.eclipse.org/legal/epl-2.0/>.

Set-cover Greedy algorithm

The MIT License (MIT)

Copyright (c) 2013 Martin Steinegger

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.