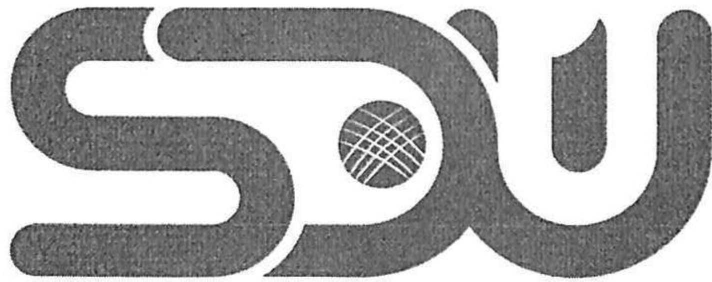
Ministry of Education and Science of the Republic of Kazakhstan
Suleyman Demirel University



Temirlan Shaikenov

Development of test automation with distributed systems

THESIS

Presented in Partial Fulfillment for the Degree
of Master of Science in Computer Science
(degree code: 7M06102)
Department of Computer Sciences
Faculty of Engineering and Natural Sciences

Supervisor: Larissa Kiziyeva

Kaskelen, 2022

Suleyman Demirel University

Faculty of Engineering and Natural Sciences

Department of Computer Science

Dean of Faculty

Associate Professor, PhD Zhamanov A.

«20» 06 2022

Topic of the thesis

Development of test automation with distributed systems

Thesis submitted as part of the requirements for the award of the MSc in "7M06102 - Computer Science", SDU, 2020-2022

Head of Department

Assos.prof. Cemil Turan

Academic Supervisor

Larissa Kiziyeva

Master's student

Temirlan Shaikenov

Kaskelen, 2022

Abstract

There are numerous techniques for developing test automation in today's quality assurance automation area. Different frameworks, applications, and programming languages can be used to build it. Automation tests can be done in parallel or in series. Many test automation developers are unsure which technique to use while developing their software. We are interested in developing test automation with distributed systems that perform automated tests on numerous machines and browsers in parallel. The creation of test automation with distributed systems is a crucial feature of this effort, as is the design of an effective architecture with the integration of various applications. We used the Selenium grid framework to spread automated tests across numerous servers that were containerized in Docker. In Jenkins Continuous Integration (CI) and Continuous Delivery (CD) environment, we run all processes with maven. As a result, we created development test automation with distributed systems that perform automation tests in parallel, speeding up testing and allowing it to run on several browsers and operating systems at the same time.

Аңдатпа

Бүгінгі сапаны қамтамасыз етуді автоматтандыру аймағында сынақ автоматикасын дамытудың көптеген әдістері бар. Оны құру үшін әртүрлі фреймворктарды, қолданбаларды және бағдарламалау тілдерін пайдалануға болады. Автоматтандыру сынақтарын параллельді немесе тізбектей орындауға болады. Көптеген тестілеуді автоматтандыруды әзірлеушілер бағдарламалық жасақтамасын әзірлеу кезінде қандай әдісті қолдану керектігін білмейді. Біз параллельді түрде көптеген машиналар мен браузерлерде автоматтандырылған сынақтарды орындайтын бөлінген жүйелермен тестілеуді автоматтандыруды дамытуға мүдделіміз. Үлестірмелі жүйелермен тестілеуді автоматтандыруды құру осы күш-жігердің шешуші ерекшелігі болып табылады, сонымен қатар әртүрлі қолданбаларды біріктіру арқылы тиімді архитектураны жобалау. Біз Selenium тор құрылымын Docker-те контейнерленген көптеген серверлер бойынша автоматтандырылған сынақтарды тарату үшін қолдандық. Jenkins Continuous Integration (CI) және Continuous Delivery (CD) ортасында біз барлық процестерді maven көмегімен іске қосамыз. Нәтижесінде біз автоматтандыру сынақтарын параллель орындайтын, тестілеуді жылдамдатып, оның бірнеше браузерлер мен операциялық жүйелерде бір уақытта жұмыс істеуіне мүмкіндік беретін үлестірілген жүйелермен әзірлеу сынақтарын автоматтандыруды жасадық.

Аннотация

В современной области автоматизации обеспечения качества существует множество методов разработки автоматизации тестирования. Для его создания могут использоваться различные фреймворки, приложения и языки программирования. Автоматические тесты могут выполняться параллельно или последовательно. Многие разработчики автоматизации тестирования не уверены, какую технику использовать при разработке своего программного обеспечения. Мы заинтересованы в разработке автоматизации тестирования с помощью распределенных систем, которые выполняют автоматические тесты на множестве компьютеров и браузеров параллельно. Важнейшей особенностью этих усилий является создание автоматизации тестирования с помощью распределенных систем, а также проектирование эффективной архитектуры с интеграцией различных приложений. Мы использовали грид-фреймворк Selenium для распространения автоматических тестов на множество серверов, которые были контейнеризованы в Docker. В среде Jenkins Continuous Integration (CI) и Continuous Delivery (CD) мы запускаем все процессы с помощью maven. В результате мы создали автоматизацию тестирования разработки с распределенными системами, которые параллельно выполняют тесты автоматизации, ускоряя тестирование и позволяя запускать его в нескольких браузерах и операционных системах одновременно.

Acknowledgements

Thanks to my thesis supervisor Darmen Kariboz and Doctor Of Science Lyazzat Atymtayeva for constant support and useful discussion. Thanks to the external reviewer PhD Shara Toibaeva for very useful feedback that helped to significantly improve the current work.

Contents

1 **Introduction** **8**
1.1 Quality assurance 8
1.2 Literature Review 9
1.3 Questions and objectives 10
1.4 Thesis Outline 11

2 **METHODS** **12**
2.1 Methodological approach 12
2.2 Methods of data collection 12
2.3 Data analysis methods 13
2.4 Docker 13
2.5 Selenium Grid 15
2.6 Maven 17
2.7 TestNG 23
2.8 Automated tests 26
2.9 Jenkins 45
2.10 Architecture 48

3 **RESULTS AND DISCUSSION** **49**
3.1 Quantity of containerized machines 49
3.2 Time of work 52
3.3 Series and distributed mode 53

4 **Conclusion** **59**
4.1 Distributed mode 59
4.2 Integrated applications 59
4.3 Dependency 59

4.4 Automated test cases 60
4.5 Future work 60

A Appendix A **61**
A.1 Integration 61

B Appendix B **62**
B.1 Time execution automated test 62

References **67**

Nomenclature

CI/CD Continuous Integration/Continuous Delivery

GB gigabyte

IDE Integrated Development Environment

QA Quality Assurance

RAM random access memory

ROM read-only memory

1. Introduction

1.1 Quality assurance

Quality assurance (QA) is rapidly expanding, and there are numerous techniques for designing a test automation environment. Using Java, Python, JavaScript, and other computer languages, we may create automated tests. We also have a variety of frameworks, applications, and techniques for various programming languages. The overall goal of this study is to demonstrate how we can construct test automation[5] with distributed systems. This method will allow us to execute automation tests in parallel mode[1][13], allowing us to run tests on numerous machines with the desired operating system and version. If one machine fails, it has no bearing on the other machines. The significance of this topic is that it demonstrates the creation of test automation with distributed systems by designing an effective architecture with the integration of many applications. This will assist test automation developers in creating a reliable and effective test automation environment.

Jenkins Continuous Integration (CI) and Continuous Delivery (CD), Docker containers, Docker images, Selenium grid, Eclipse integrated development environment (IDE) , Maven, test runner TestNG, and browsers are all used to accomplish this. Jenkins is a CI/CD server that automates the development process by developing, testing, and deploying software. Docker is an OS-level virtualization platform that uses applications packaged in containers. A Selenium grid is a server that runs tests on numerous machines simultaneously. In addition, the Selenium grid has a hub and employees called Grid nodes. In Selenium Grid, the Hub node controls the Grid nodes. Maven makes it simple to create Java projects. TestNG is a test runner that facilitates the execution of automation tests.

1.2 Literature Review

We have a number of research papers in the automation testing area that illustrate how to develop test automation in series mode. Some publications describe how to apply automation with various frameworks based on computer languages, while others discuss the benefits and drawbacks of various test automation methods. This study focuses on development test automation using distributed systems, which aids in the parallel execution of stable tests. We did not find a similar topic.

We published "The best framework for QA automation testing: Advantages and disadvantages of Robot framework."[21] article in Suleyman Demirel University Bulletin: Natural and Technical Sciences. Where we can find seven criteria's which showed positive and negative sides of this framework in testing area. One of the criteria's was parallel execution where we used distributed mode with Selenium grid. This criteria showed positive effect where we saved time for execution automated test cases. In this article we did not show how we made it, we just took results. Also, we did not work in dependency of things which we need in running automated test cases.

Also, we published "DEVELOPMENT OF TEST AUTOMATION WITH DISTRIBUTED"[20] article in XIX International Multidisciplinary Conference "Innovations and Tendencies of State-of-Art Science" Mijnbestseller Nederland, Rotterdam, Nederland. Where we show how we can develop test automation with distributed systems but we did not provide detail information which help in development. We did not make investigation in test automation with distributed systems area.

Article "A framework for automated testing of automation systems"[2] showed series mode of running automated test cases. Where Test-First Development framework improved automation systems development and product quality. Author had test case generation, execution and reporting which support testing processes more efficiently. This approach standard test automated work where we waste time for execution in series mode.

Article "A Maintainability Spreadsheet-Driven Regression Test Automation Framework"[15], author showed how we can use Spreadsheet-Driven Regression test automation framework without script development. Also, we can see that

this framework showing series mode execution of automated test cases. This method took more time than distributed because test developer should control spreadsheet, this is manual test work and takes more time than automated test work.

Article "Software Integration Test Report Analysis Automation Using Python" [16] we can find how Python help in reading test report documents with integration testing at Varroc is performed using VectorCAST tool that generates test reports in HTML format. This method focused in parameters finding, if we generated new format test report document with new parameters, this method will fail. For making adaptation for different test report documents we need time. In test environment we work for checking errors in application. If we took more time in analyzing test reports, we will develop or check manual less test cases.

In overall, we published two articles where we have parallel execution but we found some gaps which should be improved in this thesis work. Also, other articles showed series mode execution where the main factor is execution time. In this thesis work we showed how we can save execution time of running automated test cases with distributed systems.

1.3 Questions and objectives

For making strong research question we answered for six criteria's:

-Focused, We focused in developing test automation with distributed systems which help in making stable test environment and saving execution time.

-Researchable, for this research we used primary and secondary sources.

-Feasible, we can answered within the timeframe and practical constraints.

-Specific, we can provide full information for answering.

-Complex, for answering we made complex operation steps.

-Relevant, this research relevant to our field of study.

Research question:

-How development test automation with distributed systems affect execution time with different factors?

Objectives:

-Develop test automation with distributed systems for showing architecture.

-Run automated test cases in series and distributed mode.

-Collect data from test reports.

-Find dependency of different factors and execution time in test automation with distributed systems.

1.4 Thesis Outline

The remainder of this paper is structured as follows: The first chapter is Introduction chapter. In the section 2 we will have methods for this topic. Section 3 presents results and discussion of this framework and Section 4 gives us Conclusion. Finally, Section 5 presents references and last section gives abstract.

2. METHODS

2.1 Methodological approach

For finding factors which affect on execution time in test automation distributed system, we need to develop architecture of distributes automation test and analyze data from test reports which generated from test runner and test framework.

We collected data in quantitative type because our factors measured in number. We used primary data because we collected data from our test reports which it gives original data. This research is experimental, we intervene in a process and measure the outcome.

This approach is the most suitable for answering our research question because factors which affect on execution time measured in numbers and from experiments we can make results. In test automation field is a standard methodology.

2.2 Methods of data collection

We had quantitative method because we collected results from experiments which run in test automation with distributed systems from test report documents. Also, we took from laptop characteristics. In experiment we developed test automation with distributed systems test environment where we run in Java automated test cases. From test runner and test framework we generated test reports with results of execution time, quantity test cases, laptop characteristics, remote containerized machines. Reports generated in html format where we can see results of automated test execution.

2.3 Data analysis methods

We chose statistical analysis for analyzing data from experiments. We collected data from test reports which run in distributed automated tests environment.

2.4 Docker

We must get all applications from the official website in order to perform development test automation with distributed systems. To begin with, we use Docker containers, which allow you to package and start up programs with all of their prerequisites, including libraries, dependencies, and databases. We should host many virtual machines as Grid nodes that link to the hub node when configuring the Selenium grid.[4] We must also download the Selenium server jar, which we must install on each system where we perform Selenium grid tests. It takes a long time for testers to configure in this manner, and Docker containers assist us to alleviate this problem. We install images of the hub and browser nodes into Docker containers to integrate Selenium grid with Docker containers. For this study, we used the Selenium hub image, Selenium node-chrome image, and Selenium node-firefox image to execute tests.[22] You may get these images through the official Docker Hub website or from the console using the following commands:

-docker pull selenium/hub

-docker pull selenium/node-chrome

-docker pull selenium/node-firefox

We may check using the terminal command (see Figure 2.1) after installing all the images.

```
C:\Users\Temirlan>docker images
REPOSITORY               TAG      IMAGE ID       CREATED
selenium/node-firefox    latest   1d08547e0727   2 weeks ago
selenium/node-chrome     latest   fd32b604b25e   2 weeks ago
selenium/hub             latest   13ec453a88ef   2 weeks ago
```

Figure 2.1: Docker images

Docker networking must be configured to recognize hubs and nodes by their container names in order to run Docker images.[19] We type docker network create

name in the terminal. Grid is the name of our Docker network. In Figure 2.2, we need to set the port, Docker network, and name for launching the hub image.

```
C:\Users\Temirlan>docker run -d -p 4442-4444:4442-4444 --net grid
 --name selenium-hub selenium/hub
```

Figure 2.2: Run Selenium hub images

In Figure 2.3, we need to specify port, Docker network, name of Selenium hub image, publish and subscribe ports, and memory capacity for executing browser nodes image.

```
C:\Users\Temirlan>docker run -d -p 4447:5900 --net grid -e
SE_EVENT_BUS_HOST=selenium-hub --shm-size="2g" -e SE_EVENT_
BUS_PUBLISH_PORT=4442 -e SE_EVENT_BUS_SUBSCRIBE_PORT=4443 s
elenium/node-chrome
```

Figure 2.3: Run Selenium node images

Figure 2.4 show the main hub image node as well as three browser image nodes.

```
C:\Users\Temirlan>docker ps -a
CONTAINER ID   IMAGE                    COMMAND
               NAMES
71ae23423f46   selenium/node-firefox    "/opt/bin/entry_poin…"
               tender_shirley
ae489be5690    selenium/node-chrome     "/opt/bin/entry_poin…"
               hopeful_swirles
9062ae236c33   selenium/node-chrome     "/opt/bin/entry_poin…"
               condescending_lalande
3b90dd76bc91   selenium/hub             "/opt/bin/entry_poin…"
 4444/tcp      selenium-hub
```

Figure 2.4: Docker running images

Figure 2.5 show the images status and ports.

```
STATUS            PORTS
Up 12 seconds     0.0.0.0:4447->5900/tcp
Up 5 minutes      0.0.0.0:4446->5900/tcp
Up 5 minutes      0.0.0.0:4445->5900/tcp
Up 5 minutes      0.0.0.0:4442-4444->4442
```

Figure 2.5: Docker running images status and ports

2.5 Selenium Grid

We go to the default local website "http://localhost:4444" to make that everything is working, and we see three remote nodes in Figure 2.6.

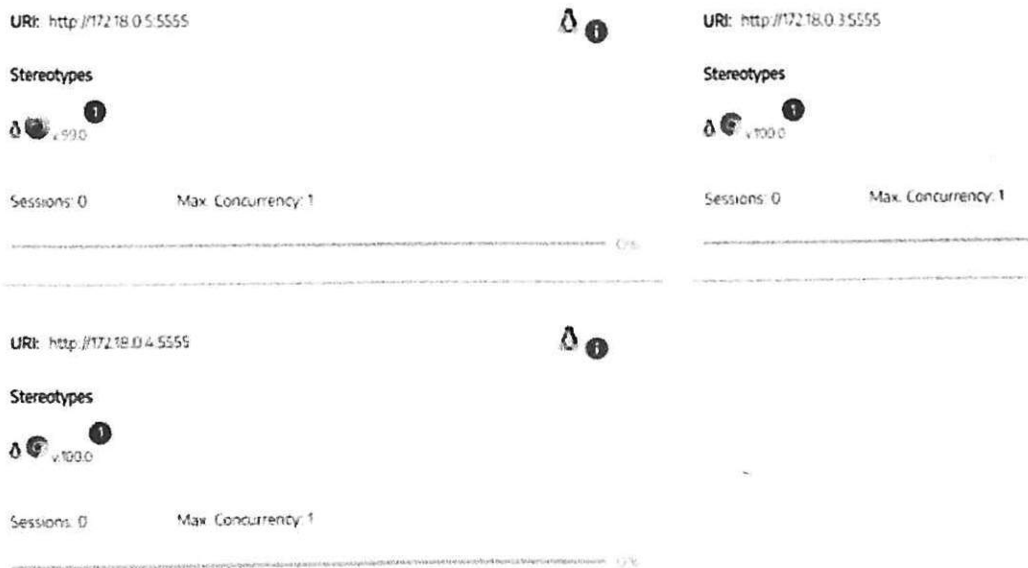


Figure 2.6: Selenium grid local website

In this approach, Docker containers assist us in establishing remote computers on which we can execute automation tests using the Selenium grid in Figure 2.7.

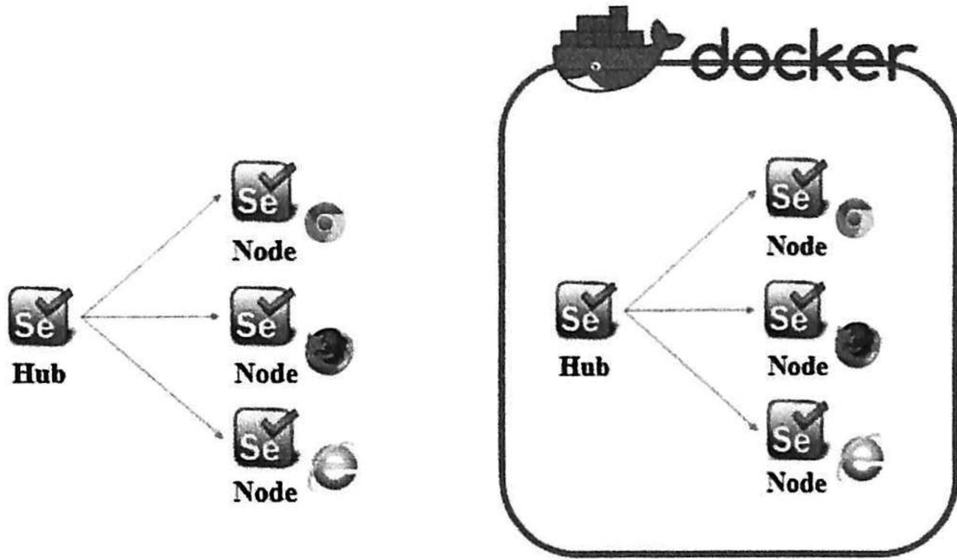


Figure 2.7: Selenium grid without and with Docker containers

2.6 Maven

We created Maven Project in Eclipse IDE in Figure2.8.

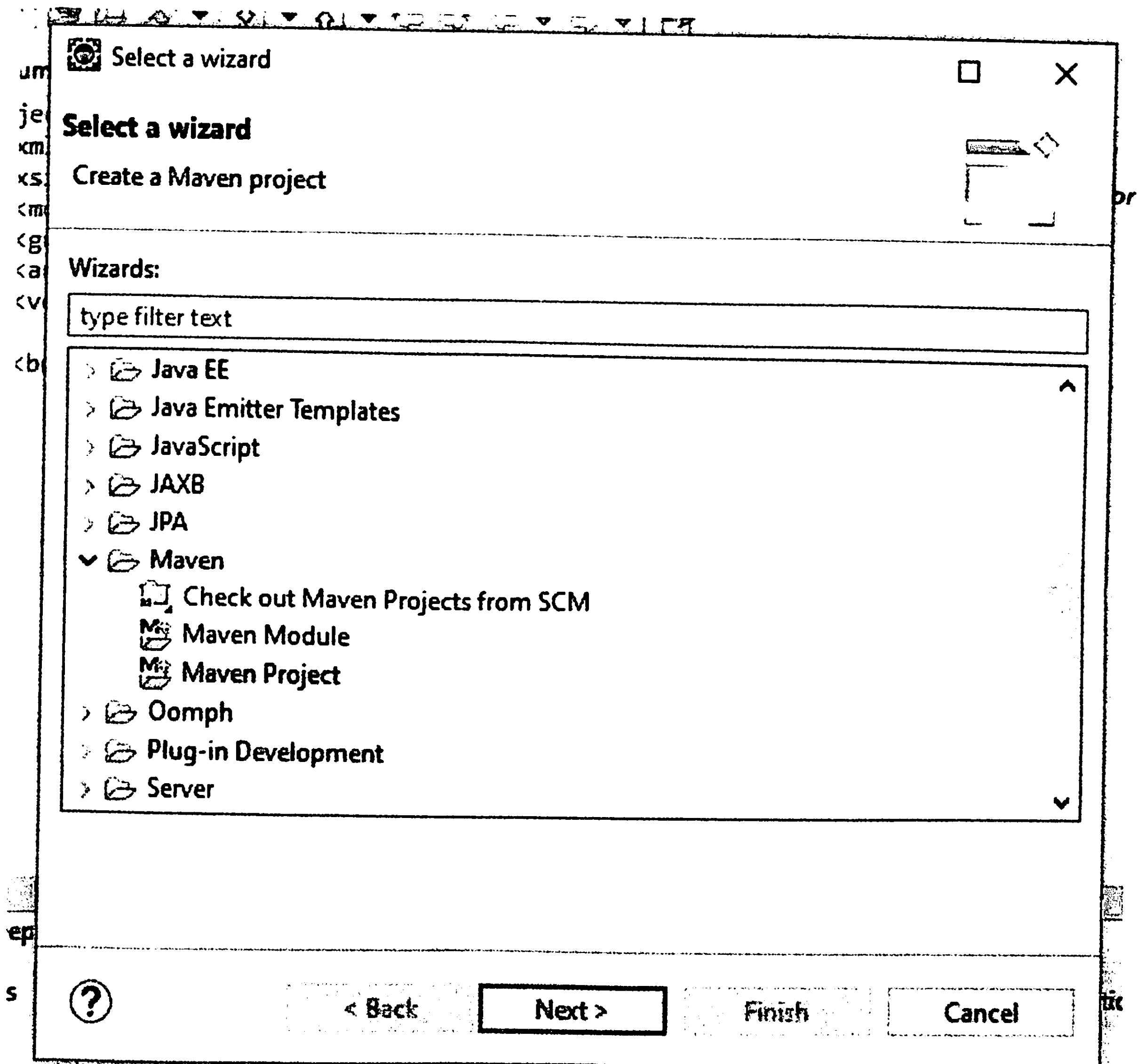


Figure 2.8: Maven Project

When Maven Project created, in the left side we have Package Explorer where we have all structure of Maven project in Figure2.9.

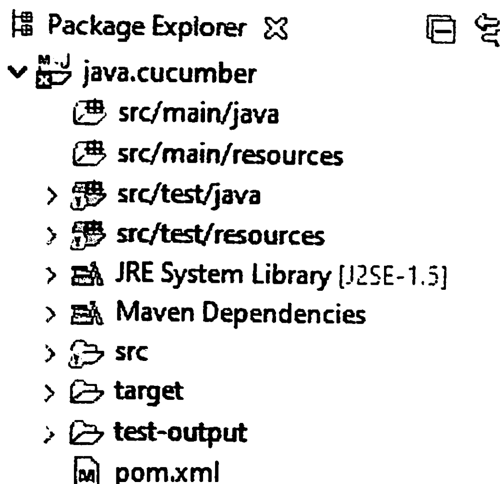


Figure 2.9: Package Explorer of Maven Project

We found last stable version of Selenium java for maven pom dependency from website "https://mvnrepository.com/" in Figure2.10.

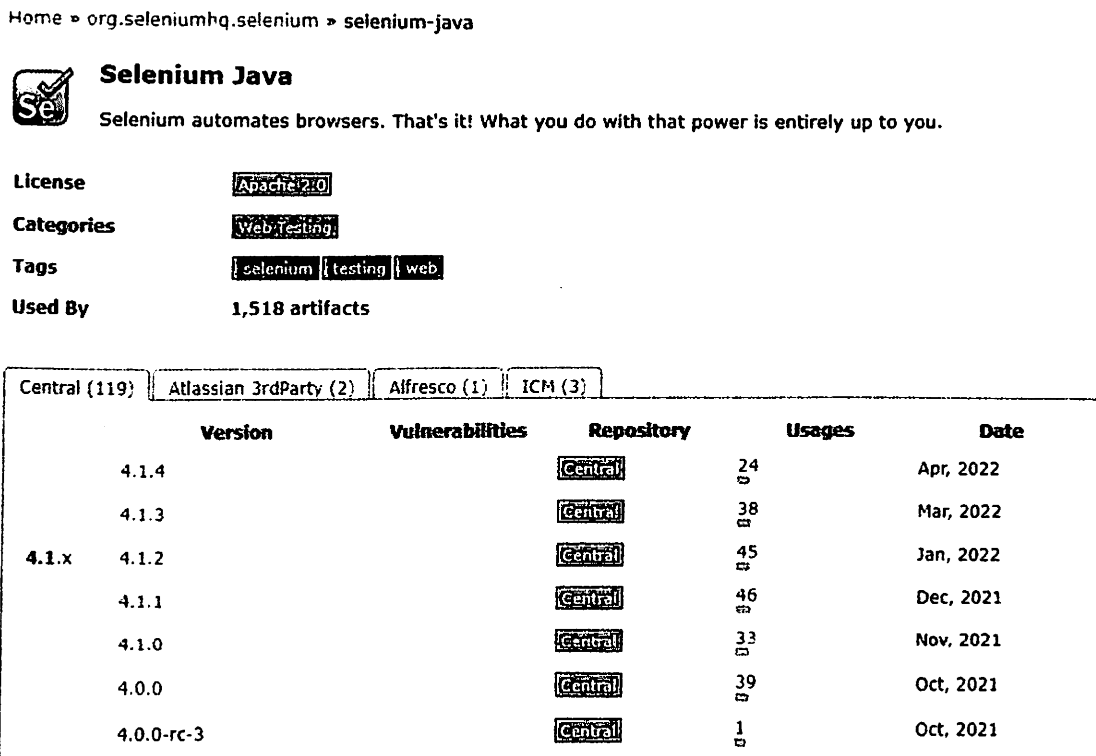


Figure 2.10: Selenium Java pom dependency

We chose Selenium Java version 4.0.0 in Figure2.11.

Selenium Java » 4.0.0

Selenium automates browsers. That's it! What you do with that power is entirely up to you.

License	Apache 2.0
Categories	Web Testing
HomePage	https://selenium.dev/
Date	(Oct 13, 2021)
Files	pom (4 KB) jar (740 bytes) View All
Repositories	Central
Used By	**1,518 artifacts**

Note: There is a new version for this artifact

New Version	4.1.4

Maven | Gradle | Gradle (Short) | Gradle (Kotlin) | SBT | Ivy | Grape | Leiningen | Buildr

```
<!-- https://mvnrepository.com/artifact/org.seleniumhq.selenium/selenium-java -->
<dependency>
    <groupId>org.seleniumhq.selenium</groupId>
    <artifactId>selenium-java</artifactId>
    <version>4.0.0</version>
</dependency>
```

Figure 2.11: Selenium Java version 4

Also, we chose TestNG test runner for maven pom dependency in Figure2.12.

Home » org.testng » testng

TestNG

TestNG

Testing framework for Java

License	Apache 2.0
Categories	Testing Frameworks
Tags	testing testng
Used By	**10,547 artifacts**

Central (79) | Spring Lib Release (1) | Spring Plugins (18) | Redhat GA (1) | Redhat EA (2) | FenixEdu (1) | JCenter (1) | Gradle Releases (1) | Kyligence (1) | Mulesoft (1) | ICM (4)

	Version	Vulnerabilities	Repository	Usages	Date
7.6.x	7.6.0		Central	6	May, 2022
7.5.x	7.5		Central	337	Jan, 2022
7.4.x	7.4.0		Central	582	Feb, 2021
7.3.x	7.3.0		Central	545	Aug, 2020
7.1.x	7.1.0		Central	737	Dec, 2019
	7.0.0		Central	390	Aug, 2019

Figure 2.12: TestNG pom dependency

We chose TestNG version 6.8 in Figure2.13.

Home » org.testng » testng » 6.8

TestNG » 6.8

Testing framework for Java

License	Apache 2.0
Categories	Testing Frameworks
HomePage	http://testng.org
Date	(Sep 10, 2012)
Repositories	Central IBiblio Sonatype
Used By	10,547 artifacts
Vulnerabilities	Vulnerabilities from dependencies: CVE-2021-36374 CVE-2021-36373 CVE-2020-1945 View 3 more ...

Note: There is a new version for this artifact

New Version 7.6.0

Maven | Gradle | Gradle (Short) | Gradle (Kotlin) | SBT | Ivy | Grape | Leiningen | Buildr

```
<!-- https://mvnrepository.com/artifact/org.testng/testng -->
<dependency>
    <groupId>org.testng</groupId>
    <artifactId>testng</artifactId>
    <version>6.8</version>
    <scope>test</scope>
</dependency>
```

Figure 2.13: TestNG version 6.8

The next step is to create automation tests that will execute in a distributed environment.[3] In Figure2.14, we set up a maven Java project and used the default pom.xml file to add TestNG and Selenium for browser integration.

```
<dependency>
    <groupId>org.seleniumhq.selenium</groupId>
    <artifactId>selenium-java</artifactId>
    <version>4.0.0</version>
</dependency>

<dependency>
    <groupId>org.testng</groupId>
    <artifactId>testng</artifactId>
    <version>6.8</version>
    <scope>test</scope>
</dependency>
```

Figure 2.14: Selenium and TestNG in pom.xml file

We must also configure this project to run via the maven command.[10] In Figure2.15, we add additional configuration to pom.xml and specify that tests be executed using the TestNG test runner.

```
        <groupId>org.apache.maven.plugins</groupId>
        <artifactId>maven-compiler-plugin</artifactId>
        <version>3.10.1</version>
        <configuration>
            <source>1.8</source>
            <target>1.8</target>
        </configuration>
    </plugin>

    <plugin>
        <groupId>org.apache.maven.plugins</groupId>
        <artifactId>maven-surefire-plugin</artifactId>
        <version>3.0.0-M6</version>
        <dependencies>
          <dependency>
            <groupId>org.apache.maven.surefire</groupId>
            <artifactId>surefire-testng</artifactId>
            <version>3.0.0-M6</version>
          </dependency>
        </dependencies>
        <configuration>
            <suiteXmlFiles>
                <suiteXmlFile>testng.xml</suiteXmlFile>
            </suiteXmlFiles>
```

Figure 2.15: Maven with test runner TestNG in pom.xml file

2.7 TestNG

In Figure2.16, we need to build a testng.xml file in which we configured the parallel test runner.[23]

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE suite SYSTEM "https://testng.org/testng-1.0.dtd">
<suite parallel="classes" name="Suite">
  <test name="Test">
    <classes>
      <class name="steps.TestNGParallel"/>
      <class name="steps.TestNGParallel2"/>
    </classes>
  </test> <!-- Test -->
</suite> <!-- Suite -->
```

Figure 2.16: Testng.xml file

We configured to run 5 automated test cases in distributed mode with help of TestNG runner in Figure2.17.

```
1  <?xml version="1.0" encoding="UTF-8"?>
2  <!DOCTYPE suite SYSTEM "https://testng.org/testng-1.0.dtd">
3  <suite parallel="classes" name="Suite">
4    <test name="Test">
5      <classes>
6        <class name="steps.TestNGParallel"/>
7        <class name="steps.TestNGParallel2"/>
8        <class name="steps.TestNGParallel3"/>
9        <class name="steps.TestNGParallel4"/>
10       <class name="steps.TestNGParallel5"/>
11     </classes>
12   </test> <!-- Test -->
13 </suite> <!-- Suite -->
14
```

Figure 2.17: Testng 5 automated test cases

Also, we configured to run 10 automated test cases in distributed mode with help of TestNG runner in Figure2.18.

```
1  <?xml version="1.0" encoding="UTF-8"?>
2  <!DOCTYPE suite SYSTEM "https://testng.org/testng-1.0.dtd">
3  <suite parallel="classes" name="Suite">
4    <test name="Test">
5      <classes>
6        <class name="steps.TestNGParallel"/>
7        <class name="steps.TestNGParallel2"/>
8        <class name="steps.TestNGParallel3"/>
9        <class name="steps.TestNGParallel4"/>
10       <class name="steps.TestNGParallel5"/>
11       <class name="steps.TestNGParallel6"/>
12       <class name="steps.TestNGParallel7"/>
13       <class name="steps.TestNGParallel8"/>
14       <class name="steps.TestNGParallel9"/>
15       <class name="steps.TestNGParallel10"/>
16     </classes>
17   </test> <!-- Test -->
18 </suite> <!-- Suite -->
19
```

Figure 2.18: Testng 10 automated test cases

And we configured to run 15 automated test cases in distributed mode with help of TestNG runner in Figure2.19. Then we run 20 test cases, 30 test cases and 50 test cases.

```
1  <?xml version="1.0" encoding="UTF-8"?>
2  <!DOCTYPE suite SYSTEM "https://testng.org/testng-1.0.dtd">
3  <suite parallel="classes" name="Suite">
4    <test name="Test">
5      <classes>
6        <class name="steps.TestNGParallel"/>
7        <class name="steps.TestNGParallel2"/>
8        <class name="steps.TestNGParallel3"/>
9        <class name="steps.TestNGParallel4"/>
10       <class name="steps.TestNGParallel5"/>
11       <class name="steps.TestNGParallel6"/>
12       <class name="steps.TestNGParallel7"/>
13       <class name="steps.TestNGParallel8"/>
14       <class name="steps.TestNGParallel9"/>
15       <class name="steps.TestNGParallel10"/>
16       <class name="steps.TestNGParallel11"/>
17       <class name="steps.TestNGParallel12"/>
18       <class name="steps.TestNGParallel13"/>
19       <class name="steps.TestNGParallel14"/>
20       <class name="steps.TestNGParallel15"/>
21     </classes>
22   </test> <!-- Test -->
23 </suite> <!-- Suite -->
24
```

Figure 2.19: Testng 15 automated test cases

2.8 Automated tests

We chose directory in "src/test/java/" and created folder "steps" where we created Java classes for automation tests in Figure2.20.

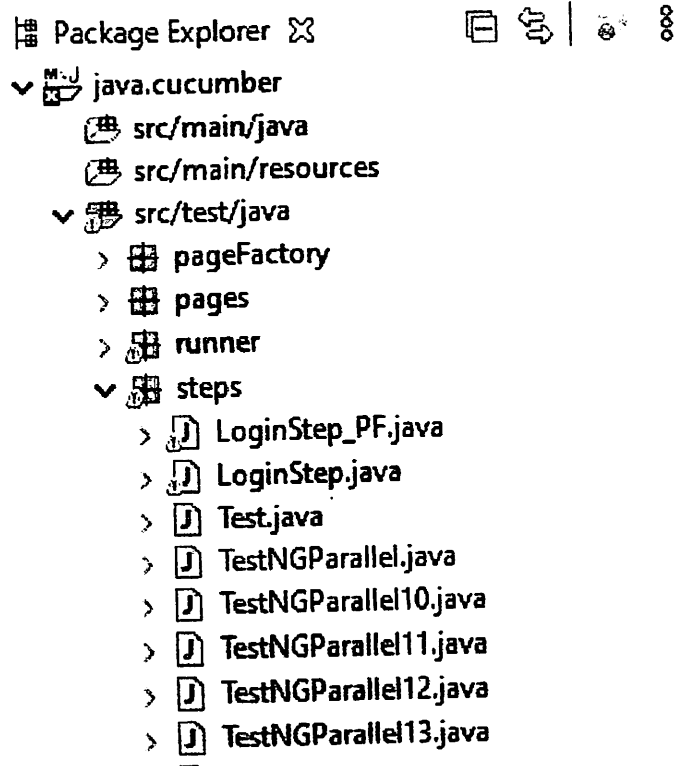


Figure 2.20: Automation test directory

In Java class file we have information what class have in Figure2.21.

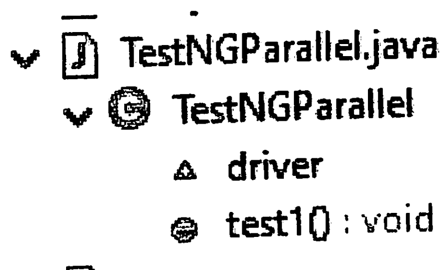


Figure 2.21: Class information

For creating class we make right click in mouse by folder, and choose new, then choose class, and we have class page where we configure this class in Figure2.22.

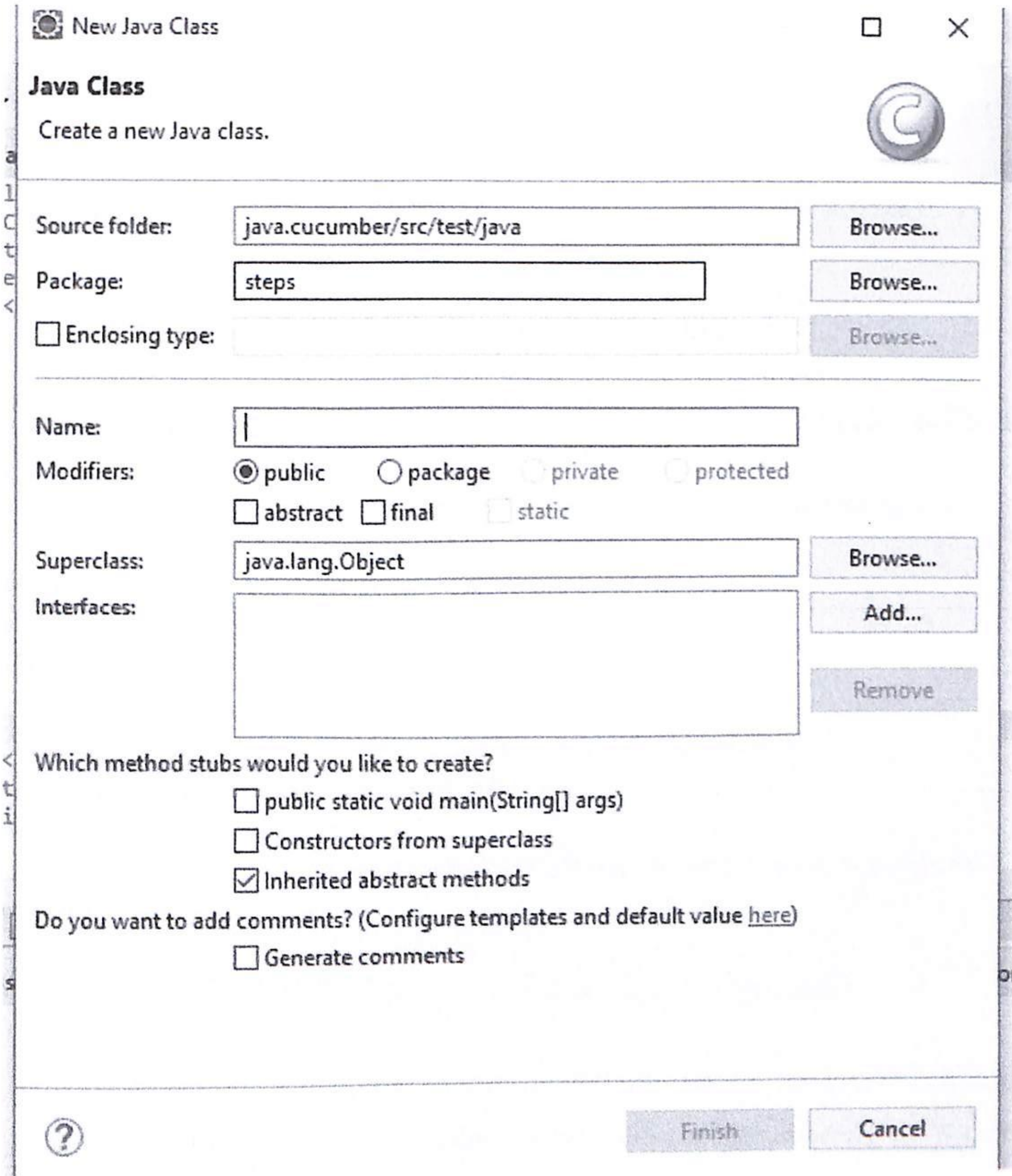


Figure 2.22: Class page

In automation test we used ChromeOptions object for give some arguments for Chrome browser. These arguments help us to configure Chrome browser in Figure2.23.

```
ChromeOptions options = new ChromeOptions()
options.addArgument("start-maximized");
ChromeDriver driver = new ChromeDriver(options);
```

Figure 2.23: ChromeOptions object

ChromeOptions object have many arguments, useful arguments in Figure2.24.

Below are the list of available and most commonly used arguments for ChromeOptions class

- **start-maximized**: Opens Chrome in maximize mode
- **incognito:** Opens Chrome in incognito mode
- **headless:** Opens Chrome in headless mode
- **disable-extensions**: Disables existing extensions on Chrome browser
- **disable-popup-blocking**: Disables pop-ups displayed on Chrome browser
- **make-default-browser:** Makes Chrome default browser
- **version**: Prints chrome browser version
- **disable-infobars:** Prevents Chrome from displaying the notification 'Chrome is being controlled by automated software

Figure 2.24: ChromeOptions object arguments

The ChromeOptions Class is a Selenium WebDriver[11] notion for changing the Chrome driver's different characteristics. For configuring Chrome driver sessions, the Chrome options class is usually used in conjunction with Desired Capabilities. It allows you to do things like open Chrome in maximized mode, disable existing extensions, and disable pop-ups, among other things.

Also, we used Remote WebDriver for connection Selenium Hub where we sent request for running automation test cases in Figure2.25.

```
ChromeOptions chromeOptions = new ChromeOptions();
chromeOptions.setCapability("browserVersion", "67");
chromeOptions.setCapability("platformName", "Windows XP");
WebDriver driver = new RemoteWebDriver(new URL("http://www.example.com"), chromeOptions);
driver.get("http://www.google.com");
driver.quit();
```

Figure 2.25: Remote WebDriver for Chrome

For FireFox browser we used Remote WebDriver in Figure2.26.

```
FirefoxOptions firefoxOptions = new FirefoxOptions();
WebDriver driver = new RemoteWebDriver(new URL("http://www.example.com"), firefoxOptions);
driver.get("http://www.google.com");
driver.quit();
```

Figure 2.26: Remote WebDriver for FireFox

We added to pom.xml file required system properties for Remote WebDriver in Figure2.27.

```
<dependency>
    <groupId>io.opentelemetry</groupId>
    <artifactId>opentelemetry-exporter-jaeger</artifactId>
    <version>1.0.0</version>
  </dependency>
  <dependency>
    <groupId>io.grpc</groupId>
    <artifactId>grpc-netty</artifactId>
    <version>1.35.0</version>
  </dependency>
```

Figure 2.27: Remote WebDriver pom required system properties

For adding required system properties for Remote WebDriver in Figure2.28.

```
System.setProperty("otel.traces.exporter", "jaeger");
System.setProperty("otel.exporter.jaeger.endpoint", "http://localhost:14250");
System.setProperty("otel.resource.attributes", "service.name=selenium-java-client");

ImmutableCapabilities capabilities = new ImmutableCapabilities("browserName", "chrome");

WebDriver driver = new RemoteWebDriver(new URL("http://www.example.com"), capabilities);

driver.get("http://www.google.com");

driver.quit();
```

Figure 2.28: Remote WebDriver required system properties

We used pattern Page Object for good construction of project and remove duplicates in code in Figure2.29.

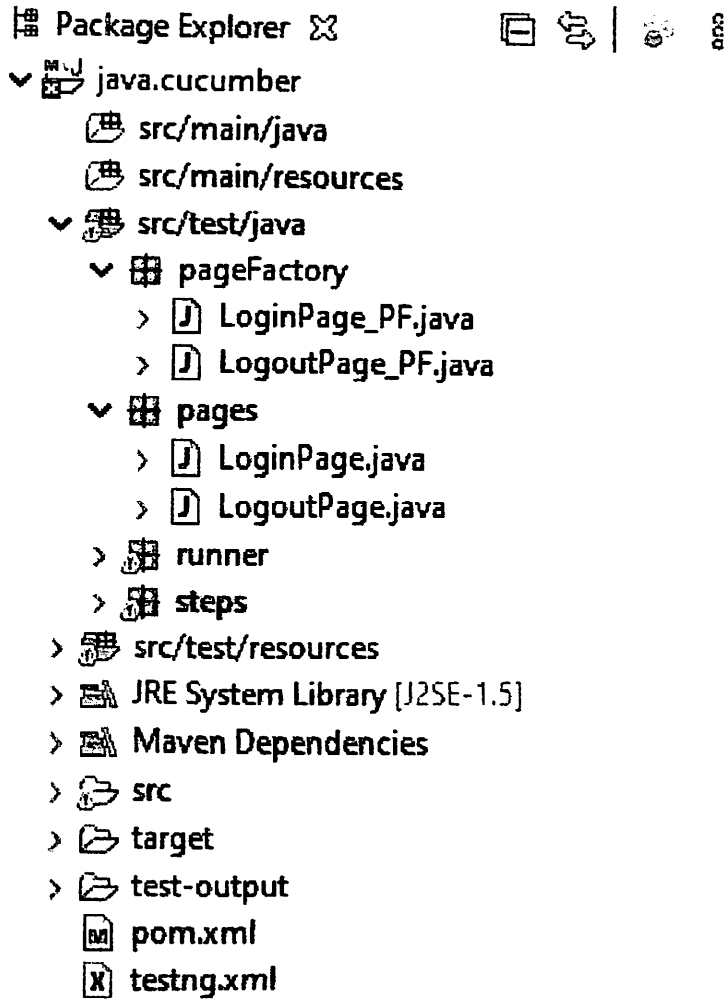


Figure 2.29: Page Object

We used pattern Page Object for page login in Figure2.30.

```
package pages;

+ import org.openqa.selenium.By;

public class LoginPage {
    protected WebDriver driver;

    private By txt_username = By.id("name");
    private By txt_password = By.id("password");
    private By txt_login = By.id("login");

-   public LoginPage(WebDriver driver) {
        this.driver = driver;
    }

-   public void enterUsername(String username) {
        driver.findElement(txt_username).sendKeys(username);
    }

-   public void enterPassword(String password) {
        driver.findElement(txt_password).sendKeys(password);
    }

-   public void clickLogin() {
        driver.findElement(txt_login).click();
    }
```

Figure 2.30: Page Object Login

Also , we used pattern Page Object for page logout in Figure2.31.

```
package pages;

import org.openqa.selenium.By;

public class LogoutPage {
    protected WebDriver driver;

    private By btn_logout = By.id("logout");

    public LogoutPage(WebDriver driver) {
        this.driver = driver;
    }

    public void checkLogout() {
        driver.findElement(btn_logout).isDisplayed();
    }

}
```

Figure 2.31: Page Object Logout

We used Cucumber for making keywords which help for better understanding steps of test case, we have Cucumber structure in Figure2.32.

Figure 2.32: Cucumber structure

Cucumber[6] for making keywords in Figure2.33.

```
#@SmokeFeature
#Feature: Test login page

# @SmokeTest
  #Scenario: Check login page
    #Given user is on login page
    #When user enter name and password
    #And click on login button
    #Then check user home page

  #Scenario Outline: Check login page
    #Given user is on login page
    #When user enter <name> and <password>
    #And click on login button
    #Then check user home page
#
    #Examples:
      #| name    | password |
      #| nameOne | 12345    |
      #| nameTwo | 12345    |
```

Figure 2.33: Cucumber keywords

We used build in keywords like: Feature, Scenario, Given, When, And, Then, Examples. Also, we can construct yourself keywords which help to understand steps of test case.

We used Java System Library in Figure2.34.

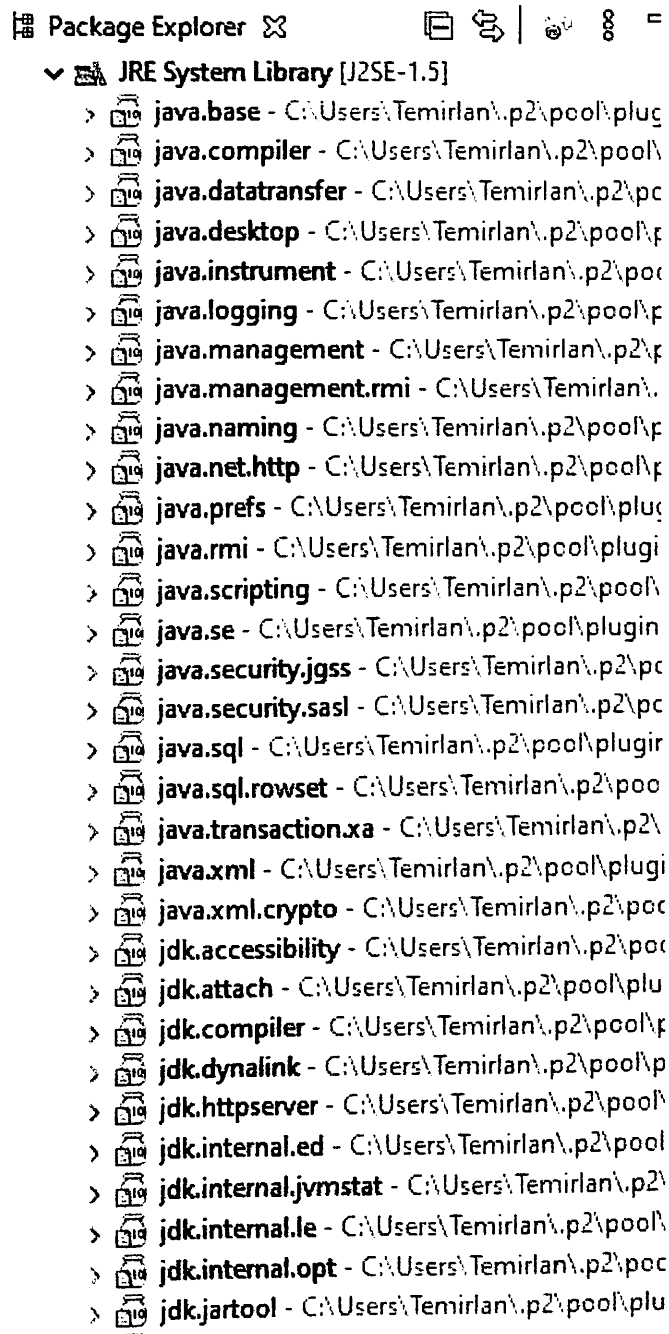


Figure 2.34: Java System Library

Maven Dependencies Library in Figure2.35.

Maven Dependencies
- apiguardian-api-1.1.2.jar - C:\Users\Ter
- async-http-client-2.12.3.jar - C:\Users\
- async-http-client-netty-utils-2.12.3.jar
- auto-common-1.0.jar - C:\Users\Temirl
- auto-service-1.0.jar - C:\Users\Temirlan
- auto-service-annotations-1.0.jar - C:\U
- bsh-2.0b4.jar - C:\Users\Temirlan\.m2\
- byte-buddy-1.11.19.jar - C:\Users\Temi
- checker-qual-3.12.0.jar - C:\Users\Temi
- commons-exec-1.3.jar - C:\Users\Temi
- create-meta-6.0.1.jar - C:\Users\Temirl
- cucumber-core-7.0.0.jar - C:\Users\Ten
- cucumber-expressions-13.0.1.jar - C:\U
- cucumber-gherkin-7.0.0.jar - C:\Users\
- cucumber-gherkin-messages-7.0.0.jar
- cucumber-java-7.0.0.jar - C:\Users\Ter
- cucumber-junit-7.0.0.jar - C:\Users\Ten
- cucumber-plugin-7.0.0.jar - C:\Users\T
- datatable-7.0.0.jar - C:\Users\Temirlan\
- docstring-7.0.0.jar - C:\Users\Temirlan\
- error_prone_annotations-2.7.1.jar - C:\U
- failureaccess-1.0.1.jar - C:\Users\Temirl
- guava-31.0.1-jre.jar - C:\Users\Temirlan
- hamcrest-core-1.3.jar - C:\Users\Temirl
- html-formatter-17.0.0.jar - C:\Users\Te
- j2objc-annotations-1.3.jar - C:\Users\Te
- jakarta.activation-1.2.2.jar - C:\Users\Te
- jcommander-1.27.jar - C:\Users\Temirl
- jsr305-3.0.2.jar - C:\Users\Temirlan\.m2

Figure 2.35: Maven Dependencies Library

In Selenium we worked with different color in Figure2.36.

```
private final Color HEX_COLOUR = Color.fromString("#2F7ED8");
private final Color RGB_COLOUR = Color.fromString("rgb(255, 255, 255)");
private final Color RGB_COLOUR = Color.fromString("rgb(40%, 20%, 40%)");
private final Color RGBA_COLOUR = Color.fromString("rgba(255, 255, 255, 0.5)");
private final Color RGBA_COLOUR = Color.fromString("rgba(40%, 20%, 40%, 0.5)");
private final Color HSL_COLOUR = Color.fromString("hsl(100, 0%, 50%)");
private final Color HSLA_COLOUR = Color.fromString("hsla(100, 0%, 50%, 0.5)");
```

Figure 2.36: Selenium Color

In Selenium we made keyboard actions like "Enter" in Figure2.37.

```
import org.openqa.selenium.By;
import org.openqa.selenium.Keys;
import org.openqa.selenium.WebDriver;
import org.openqa.selenium.firefox.FirefoxDriver;

public class HelloSelenium {
  public static void main(String[] args) {
    WebDriver driver = new FirefoxDriver();
    try {
      // Navigate to Url
      driver.get("https://google.com");

      // Enter text "q" and perform keyboard action "Enter"
      driver.findElement(By.name("q")).sendKeys("q" + Keys.ENTER);
    } finally {
      driver.quit();
    }
  }
}
```

Figure 2.37: Selenium keyboard actions

Also we have action class where we can make different clicks in mouse, drag on drop and etc.

In driver object we have method find element by id, class, path, name and etc. in Figure2.38. This is help us for communication with browser elements.

```
public class findElementsFromElement {
    public static void main(String[] args) {
        WebDriver driver = new ChromeDriver();
        try {
            driver.get("https://example.com");

            // Get element with tag name 'div'
            WebElement element = driver.findElement(By.tagName("div"));

            // Get all the elements available with tag name 'p'
            List<WebElement> elements = element.findElements(By.tagName("p"));
            for (WebElement e : elements) {
                System.out.println(e.getText());
            }
        } finally {
            driver.quit();
        }
    }
}
```

Figure 2.38: Driver find element

We had deselect option in Selenium in Figure2.39.

```
// Deselect an <option> based upon the <select> element's
selectObject.deselectByIndex(1);

// Deselect an <option> based upon its value attribute
selectObject.deselectByValue("value1");

// Deselect an <option> based upon its text
selectObject.deselectByVisibleText("Bread");

// Deselect all selected <option> elements
selectObject.deselectAll();
```

Figure 2.39: Deselect

For all locators[9] you can see in Figure2.40.

Locator	Description
class name	Locates elements whose class name contains the search value (compou
css selector	Locates elements matching a CSS selector
id	Locates elements whose ID attribute matches the search value
name	Locates elements whose NAME attribute matches the search value
link text	Locates anchor elements whose visible text matches the search value
partial link text	Locates anchor elements whose visible text contains the search value. I selected.
tag name	Locates elements whose tag name matches the search value
xpath	Locates elements matching an XPath expression

Figure 2.40: Locators

Useful locators are xpath, css selector and id.

Example for stable automated test case in Java with Selenium framework in Figure2.41.

```
public class FirstScriptTest {
    public WebDriver driver;

    @Test
    public void eightComponents() {
        driver = new ChromeDriver();

        driver.get("https://google.com");

        String title = driver.getTitle();
        Assertions.assertEquals("Google", title);

        driver.manage().timeouts().implicitlyWait(Duration.ofMillis(500));

        WebElement searchBox = driver.findElement(By.name("q"));
        WebElement searchButton = driver.findElement(By.name("btnK"));

        searchBox.sendKeys("Selenium");
        searchButton.click();

        searchBox = driver.findElement(By.name("q"));
        String value = searchBox.getAttribute("value");
        Assertions.assertEquals("Selenium", value);

        driver.quit();
    }
```

Figure 2.41: Automated test case

In new version of Selenium we had changes in set capability, this is changed to options. If we want to configured browser, we will use options for browsers in Figure2.42.

Before

Java JavaScript CSharp Ruby Python

```
DesiredCapabilities caps = DesiredCapabilities.firefox();
caps.setCapability("platform", "Windows 10");
caps.setCapability("version", "92");
caps.setCapability("build", myTestBuild);
caps.setCapability("name", myTestName);
WebDriver driver = new RemoteWebDriver(new URL(cloudUrl), caps);
```

After

Java JavaScript CSharp Ruby Python

```
FirefoxOptions browserOptions = new FirefoxOptions();
browserOptions.setPlatformName("Windows 10");
browserOptions.setBrowserVersion("92");
Map<String, Object> cloudOptions = new HashMap<>();
cloudOptions.put("build", myTestBuild);
cloudOptions.put("name", myTestName);
browserOptions.setCapability("cloud:options", cloudOptions);
WebDriver driver = new RemoteWebDriver(new URL(cloudUrl), browserOptions);
```

Figure 2.42: Selenium options

For waiting we used wait object for waiting some operations in Figure2.43.

```
// Waiting 30 seconds for an element to be present on the page, checking
// for its presence once every 5 seconds.
Wait<WebDriver> wait = new FluentWait<WebDriver>(driver)
  .withTimeout(Duration.ofSeconds(30))
  .pollingEvery(Duration.ofSeconds(5))
  .ignoring(NoSuchElementException.class);

WebElement foo = wait.until(new Function<WebDriver, WebElement>() {
  public WebElement apply(WebDriver driver) {
    return driver.findElement(By.id("foo"));
  }
});
```

Figure 2.43: Wait

Waiting[12] time help us to make stable test cases because website can load a long time. We have two types of waiting. First type when we wait exactly time and second type when we wait when element loaded.

We also need to construct a Java[14] class where we can write and run automated tests. The Selenium grid hub will be used for these tests. We use the remote web driver to connect to a remote server. The Selenium grid hub deployed these automation tests by free browser nodes once we sent commands requests to the hub node. We can select the browser to use in automation tests, as well as provide additional browser arguments using chrome options. We must first construct a chrome options object, from which we can use all of its methods. Then we use the Selenium grid URL and browser parameters with the Remote Web Driver. We can integrate and make something with the browser for testing[7] in Figure2.44.

```
package steps;

import java.net.MalformedURLException;

Run All
public class TestNGParallel {
    RemoteWebDriver driver;
    @Test
    Run | Debug
    public void test1() throws MalformedURLException {
        System.out.println("Test 1");
        ChromeOptions chromeOptions = new ChromeOptions();
        chromeOptions.addArguments("--disable-dev-shm-usage");
        driver = new RemoteWebDriver(new URL("http://192.168.56.1:4444/wd/hub"), chromeOptions
        driver.get("https://google.com");
        driver.manage().window().maximize();
        System.out.println("Chrome");
        System.out.println(driver.getTitle());
        driver.close();
        driver.quit();
    }
}
```

Figure 2.44: Automation test in Java

2.9 Jenkins

First of all, we downloaded jenkins.war file, then we run this file from command line in Figure2.45.

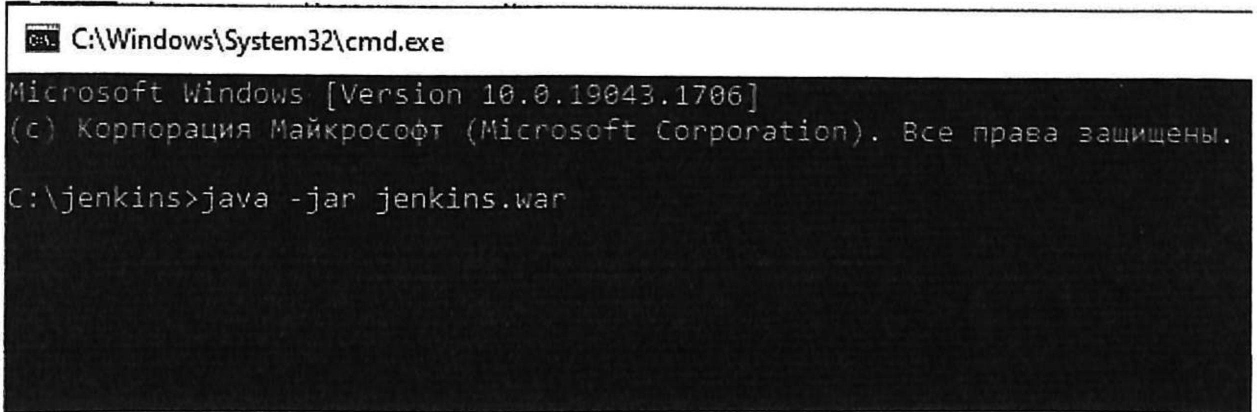


Figure 2.45: Jenkins start

When all of the automation tests are complete, Jenkins is used to execute the tests remotely from the server.[8] We configure the integration by adding the project directory and the maven command mvn clean test. Figure2.46 shows how we perform automation tests in Jenkins in a distributed manner. Jenkins may be configured to work both locally and globally. We proceed to the local URL "http://localhost:8080" to start Jenkins local server.

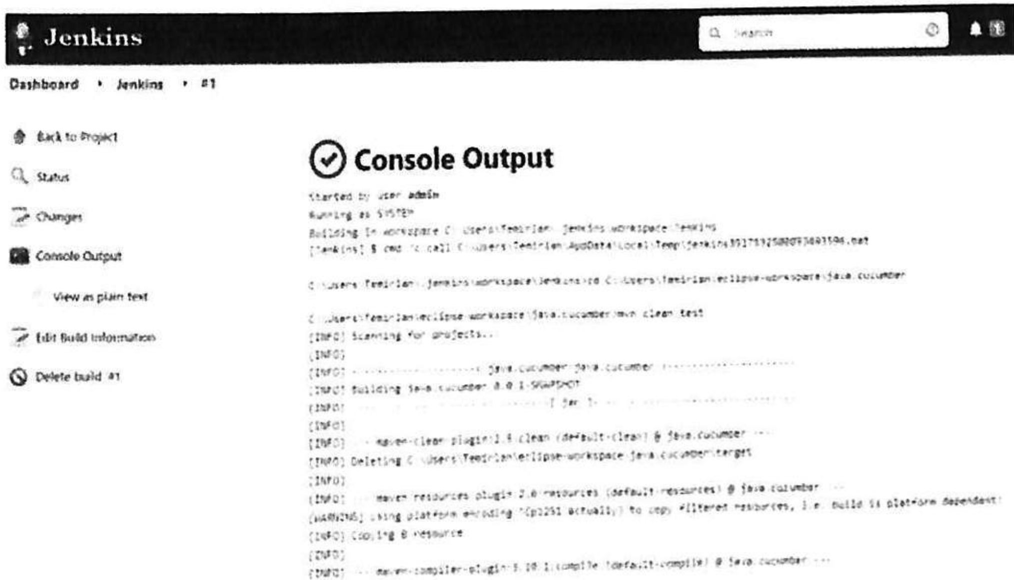


Figure 2.46: Jenkins local server

Jenkins started work and server upped on localhost with port 8080 in Figure2.47.

```
C:\jenkins>java -jar jenkins.war
Running from: C:\jenkins\jenkins.war
webroot: $user.home/.jenkins
2022-05-26 18:14:52.233+0000 [id=1]      INFO    org.eclipse.jetty
to org.eclipse.jetty.util.log.JavaUtilLog
2022-05-26 18:14:52.828+0000 [id=1]      INFO    winstone.Logger#
2022-05-26 18:14:53.116+0000 [id=1]      WARNING o.e.j.s.handler.C
2022-05-26 18:14:53.298+0000 [id=1]      INFO    org.eclipse.jetty
2021-06-30T11:07:22.254Z; git: 526006ecfa3af7f1a27ef3a288e2bef7e
2022-05-26 18:14:57.926+0000 [id=1]      INFO    o.e.j.w.StandardD
did not find org.eclipse.jetty.jsp.JettyJspServlet
2022-05-26 18:14:58.244+0000 [id=1]      INFO    o.e.j.s.s.Default
rName=node0
2022-05-26 18:14:58.244+0000 [id=1]      INFO    o.e.j.s.s.Default
g defaults
2022-05-26 18:14:58.259+0000 [id=1]      INFO    o.e.j.server.sess
 600000ms
2022-05-26 18:15:01.928+0000 [id=1]      INFO    hudson.WebAppMain
emirlan\.jenkins found at: $user.home/.jenkins
2022-05-26 18:15:04.744+0000 [id=1]      INFO    o.e.j.s.handler.C
32.2,/,file:///C:/Users/Temirlan/.jenkins/war/,AVAILABLE}{C:\User
2022-05-26 18:15:06.082+0000 [id=1]      INFO    o.e.j.server.Abst
{HTTP/1.1, (http/1.1)}{0.0.0.0:8080}
2022-05-26 18:15:06.082+0000 [id=1]      INFO    org.eclipse.jetty
2022-05-26 18:15:06.082+0000 [id=23]     INFO    winstone.Logger#
t=disabled
2022-05-26 18:15:14.115+0000 [id=30]     INFO    jenkins.InitReact
2022-05-26 18:15:27.933+0000 [id=32]     INFO    jenkins.InitReact
```

Figure 2.47: Jenkins run

For running project from Jenkins we configured Jenkins, we wrote path o project and with help maven we run project in Figure2.48.

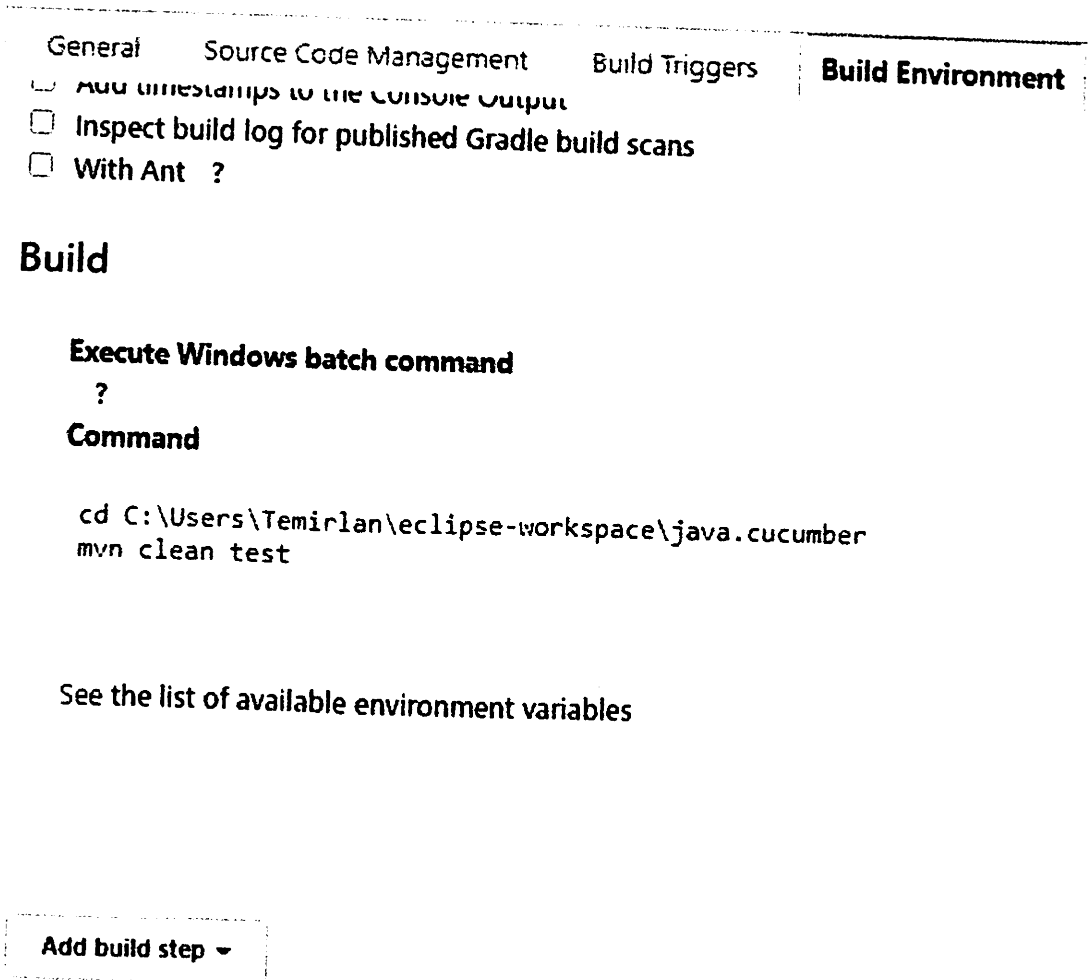


Figure 2.48: Jenkins path for running project

2.10 Architecture

We have architecture view of test automation with distributed system environment in Figure2.49.

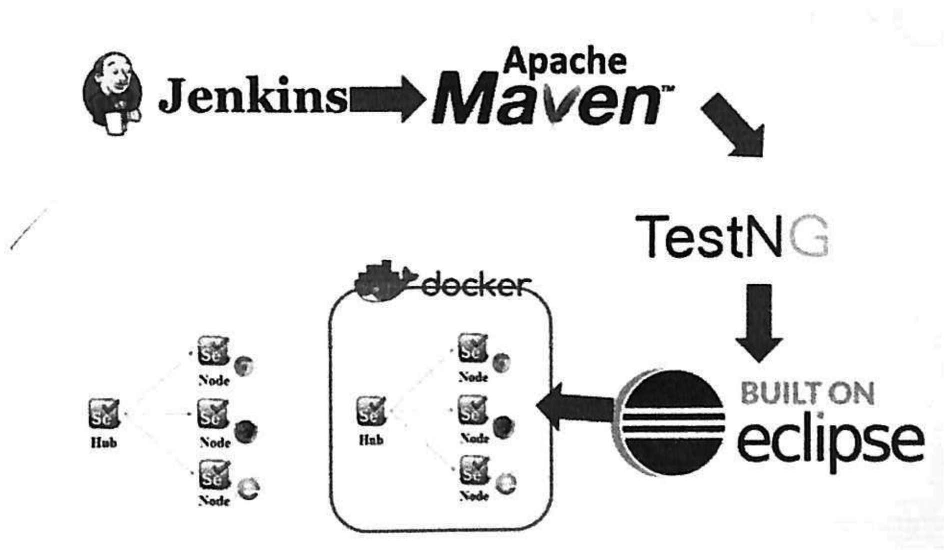


Figure 2.49: Architecture view

3. RESULTS AND DISCUSSION

3.1 Quantity of containerized machines

We have laptop with 4 gigabyte(GB) of random access memory(RAM) and 1 terabyte of read-only memory(ROM), for these characteristics we can run 3 containerized machines. If we run 4 or 5 machines, laptop will work slower than when we have 3 machines in Figure3.1 and in Figure3.2.

Процессор	Intel(R) Core(TM) i5-5200U CPU @ 2.20GHz 2.20 GHz
Оперативная память	4,00 ГБ

Figure 3.1: RAM 4GB

```
C:\Users\Temirlan>docker ps -a
CONTAINER ID   IMAGE                      COMMAND
         NAMES
71ae23423f46   selenium/node-firefox      "/opt/bin/entry_poin…"
         tender_shirley
3e4e89be5690   selenium/node-chrome       "/opt/bin/entry_poin…"
         hopeful_swirles
0062ae236c33   selenium/node-chrome       "/opt/bin/entry_poin…"
         condescending_lalande
5b98dd76bc91   selenium/hub               "/opt/bin/entry_poin…"
4444/tcp   selenium-hub
```

Figure 3.2: Remote machines 3

For laptop with 8 GB of RAM and 1 TB of ROM we can run 7 containerized machines and laptop did not work slow in Figure3.3 and in Figure3.4.

Processor: Intel(R) Core(TM) i7-2600 CPU @ 3.40GHz 3.40 GHz
Installed memory (RAM): 8.00 GB

Figure 3.3: RAM 8GB

```
C:\Users\Temirlan>docker ps -a
CONTAINER ID   IMAGE                     COMMAND
   NAMES
0978dde69805   selenium/node-chrome      "/opt/bin/entry_poin…"
   brave_goodall
b8f5ca5fd87b   selenium/node-chrome      "/opt/bin/entry_poin…"
   nifty_morse
ad2bcf61bae8   selenium/node-chrome      "/opt/bin/entry_poin…"
   competent_jepsen
efa4137d9f12   selenium/node-chrome      "/opt/bin/entry_poin…"
   stupefied_wescoff
71a023423f46   selenium/node-firefox     "/opt/bin/entry_poin…"
   tender_shirley
ca0489be5690   selenium/node-chrome      "/opt/bin/entry_poin…"
   hopeful_swirles
0062ae236c33   selenium/node-chrome      "/opt/bin/entry_poin…"
   condescending_lalande
3b90dd76bc91   selenium/hub              "/opt/bin/entry_poin…"
   selenium-hub
```

Figure 3.4: Remote machines 7

For laptop with 12 GB of RAM and 1 TB of ROM we can run 13 containerized machines and laptop did not work slow in Figure3.5 and in Figure3.6.

Processor: Intel(R) Core(TM) i7-3610QM CPU @ 2.30GHz 2.30 GHz

Installed memory (RAM): 12.0 GB (11.9 GB usable)

Figure 3.5: RAM 12GB

```
C:\Users\Temirlan>docker ps -a
CONTAINER ID   IMAGE                      COMMAND
ORTS      NAMES
ca263da66932   selenium/node-chrome       "/opt/bin/entry_poin…"
          zealous_haslett
b7c28c657ee6   selenium/node-chrome       "/opt/bin/entry_poin…"
          admiring_montalcini
15e3de7c6dcd   selenium/node-chrome       "/opt/bin/entry_poin…"
          flamboyant_gates
243f4f300ae4   selenium/node-chrome       "/opt/bin/entry_poin…"
          blissful_heyrovsky
f4f113bfef12   selenium/node-chrome       "/opt/bin/entry_poin…"
          adoring_hopper
97b499b6cf3a   selenium/node-chrome       "/opt/bin/entry_poin…"
          flamboyant_wiles
0978dde69805   selenium/node-chrome       "/opt/bin/entry_poin…"
          brave_goodall
b8f5ca5fd87b   selenium/node-chrome       "/opt/bin/entry_poin…"
          nifty_morse
ad2bcf61bae8   selenium/node-chrome       "/opt/bin/entry_poin…"
          competent_jepsen
efa4137d9f12   selenium/node-chrome       "/opt/bin/entry_poin…"
          stupefied_wescoff
71a023423f46   selenium/node-firefox      "/opt/bin/entry_poin…"
          tender_shirley
ca0489be5690   selenium/node-chrome       "/opt/bin/entry_poin…"
          hopeful_swirles
0062ae236c33   selenium/node-chrome       "/opt/bin/entry_poin…"
          condescending_lalande
3b90dd76bc91   selenium/hub               "/opt/bin/entry_poin…"
          selenium-hub
```

Figure 3.6: Remote machines 13

3.3 Series and distributed mode

We used distributed technologies to automate development testing, allowing us to run tests in parallel across numerous Machines and browsers. We had a main hub node that controlled other nodes when we used Selenium grid. Selenium grid hub will move to another working node if one or more nodes fail. This method provided us with a separate automation testing environment as well as a distribution region. In series and distributed mode, we compared running time automation tests. We discovered that distributed or parallel mode is faster than sequential or series mode. When we execute automated tests with distributed systems, as shown in Figure3.8, we can save a lot of time.

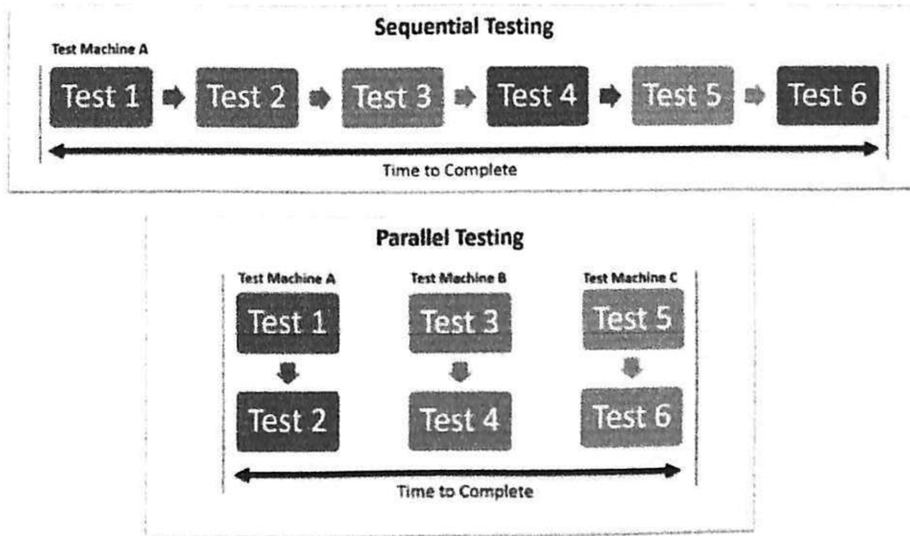


Figure 3.8: Series and distributed mode

50 automation tests were created. One running test took roughly 3 minutes. We multiplied 50 tests by 3 minutes for series mode, and the total duration was 150 minutes. We have three distant containerized Docker machines in distributed mode, and we divided 150 minutes by three machines to get a total time of 50 minutes for all automation tests see Table 3.3. Results of time execution located in Appendix B.

In the result we can say that when we increase RAM also quantity of machines increase with error 1.

Quantity of machines with quantity of RAM see Table 3.1.

Quantity of machines and RAM		
N	N of RAM	N of machines
Series	4	3
Distributed	8	7
Distributed	12	13

We have chart dependencies of RAM and machines in Figure3.7.

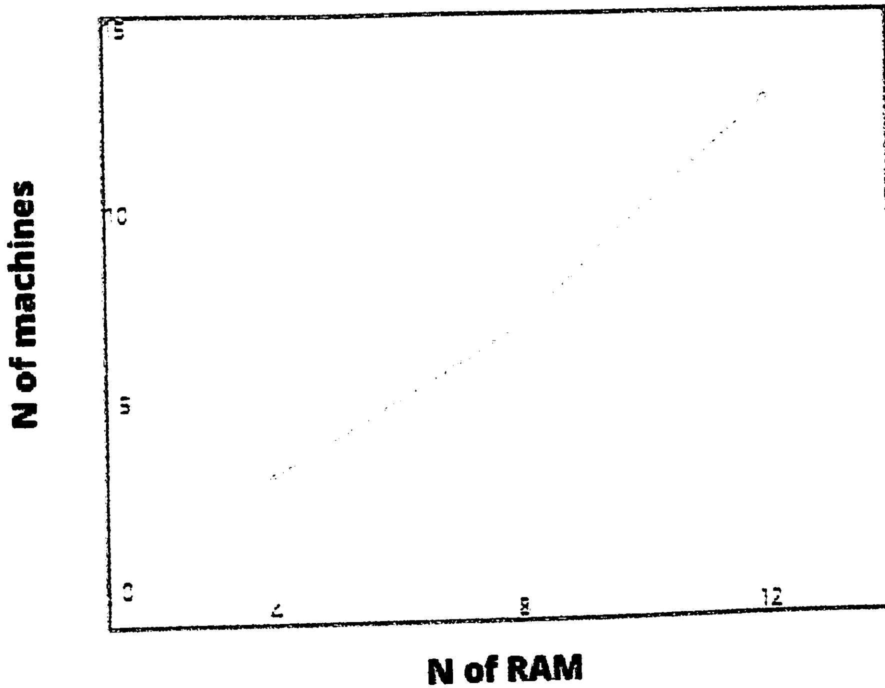


Figure 3.7: Dependencies of RAM and machines

3.2 Time of work

For upping test automation with distributed system environment we spent about two-three weeks. We searched information about this topic, configured applications, read new version of documentations, made integration between applications.

For writing automation test code we spent about 6 days. Code written in Java programming language.

For testing results we spent about 3 days.

Automation tests						
Mode	N of Tests	Time	N of RAM	N of machines	Browser	OS
Series	5	11.5min	4	1	Chrome	Linux
Distributed	5	4.24min	4	3	Chrome	Linux
Series	5	12.25min	4	1	Firefox	Linux
Distributed	5	5.14min	4	3	Firefox	Linux
Series	5	11.76min	4	1	Chrome	Windows
Distributed	5	4.64min	4	3	Chrome	Windows
Series	5	12.46min	4	1	Firefox	Windows
Distributed	5	5.44min	4	3	Firefox	Windows
Series	10	21.5min	4	1	Chrome	Linux
Distributed	10	8.24min	4	3	Chrome	Linux
Series	10	22.25min	4	1	Firefox	Linux
Distributed	10	10.14min	4	3	Firefox	Linux
Series	10	21.76min	4	1	Chrome	Windows
Distributed	10	8.64min	4	3	Chrome	Windows
Series	10	22.46min	4	1	Firefox	Windows
Distributed	10	10.44min	4	3	Firefox	Windows
Series	20	41.5min	4	1	Chrome	Linux
Distributed	20	16.24min	4	3	Chrome	Linux
Series	20	48.25min	4	1	Firefox	Linux
Distributed	20	20.14min	4	3	Firefox	Linux
Series	20	44.76min	4	1	Chrome	Windows
Distributed	20	16.64min	4	3	Chrome	Windows
Series	20	48.46min	4	1	Firefox	Windows
Distributed	20	20.44min	4	3	Firefox	Windows
Series	50	91.5min	4	1	Chrome	Linux
Distributed	50	27.24min	4	3	Chrome	Linux
Series	50	90.25min	4	1	Firefox	Linux
Distributed	50	41.14min	4	3	Firefox	Linux
Series	50	89.76min	4	1	Chrome	Windows
Distributed	50	33.64min	4	3	Chrome	Windows
Series	50	98.46min	4	1	Firefox	Windows
Distributed	50	42.44min	4	3	Firefox	Windows

Automation tests						
Mode	N of Tests	Time	N of RAM	N of machines	Browser	OS
Series	5	11.5min	8	1	Chrome	Linux
Distributed	5	2.04min	8	7	Chrome	Linux
Series	5	12.25min	8	1	Firefox	Linux
Distributed	5	2.14min	8	7	Firefox	Linux
Series	5	11.76min	8	1	Chrome	Windows
Distributed	5	2.54min	8	7	Chrome	Windows
Series	5	12.46min	8	1	Firefox	Windows
Distributed	5	2.44min	8	7	Firefox	Windows
Series	10	21.5min	8	1	Chrome	Linux
Distributed	10	4.24min	8	7	Chrome	Linux
Series	10	22.25min	8	1	Firefox	Linux
Distributed	10	5.14min	8	7	Firefox	Linux
Series	10	21.76min	8	1	Chrome	Windows
Distributed	10	4.64min	8	7	Chrome	Windows
Series	10	22.46min	8	1	Firefox	Windows
Distributed	10	5.44min	8	7	Firefox	Windows
Series	5	11.5min	12	1	Chrome	Linux
Distributed	5	1.04min	12	13	Chrome	Linux
Series	5	12.25min	12	1	Firefox	Linux
Distributed	5	1.14min	12	13	Firefox	Linux
Series	5	11.76min	12	1	Chrome	Windows
Distributed	5	1.54min	12	13	Chrome	Windows
Series	5	12.46min	12	1	Firefox	Windows
Distributed	5	1.44min	12	13	Firefox	Windows

Figure3.9 illustrate dependencies of time and other factors of tests: mode, number of test, number of RAM, number of machines, browser and OS. Also, we wrote code in Python[18] in Figure3.10 showed raw data.[17]

```
import matplotlib.pyplot as plt
import seaborn as sns
sns.set_theme()
sns.relplot(data=df1, x='n_of_machines', y='time',
 hue='mode',size='n_of_RAM',
 style='browser')
plt.show()
```

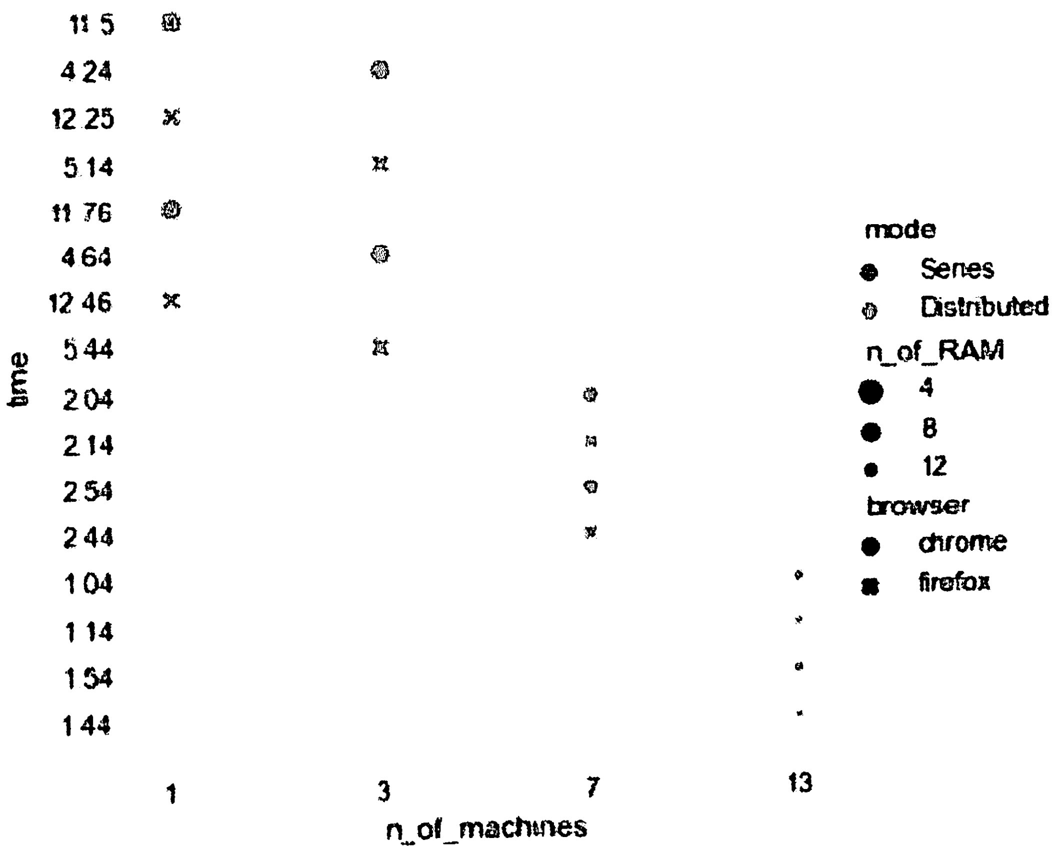


Figure 3.9: Dependencies of time and other factors of tests

```
[4]: import pandas as pd
     raw_data_1 = {
             'mode': ['Series', 'Distributed', 'Series', 'Distributed', 'Series', 'Distributed','Series', 'Distr
             'n_of_test': ['5', '5', '5', '5','5', '5', '5', '5','5', '5', '5', '5','5', '5', '5', '5','5', '5'.
             'time': ['11.5', '4.24', '12.25', '5.14','11.76', '4.64', '12.46', '5.44','11.5', '2.04', '12.25',
             'n_of_RAM': ['4', '4', '4', '4','4', '4', '4', '4','8', '8', '8', '8','8', '8', '8', '8','12', '12
             'n_of_machines': ['1', '3', '1', '3','1', '3', '1', '3','1', '7', '1', '7','1', '7', '1', '7','1',
             'browser': ['chrome', 'chrome', 'firefox', 'firefox', 'chrome', 'chrome', 'firefox', 'firefox','chr
             'os': ['linux', 'linux', 'linux', 'linux', 'windows','windows','windows','windows','linux', 'linux'

     df1 = pd.DataFrame(raw_data_1)
     df1
```

[4]:

	mode	n_of_test	time	n_of_RAM	n_of_machines	browser	os
0	Series	5	11.5	4	1	chrome	linux
1	Distributed	5	4.24	4	3	chrome	linux
2	Series	5	12.25	4	1	firefox	linux
3	Distributed	5	5.14	4	3	firefox	linux
4	Series	5	11.76	4	1	chrome	windows
5	Distributed	5	4.64	4	3	chrome	windows
6	Series	5	12.46	4	1	firefox	windows
7	Distributed	5	5.44	4	3	firefox	windows
8	Series	5	11.5	8	1	chrome	linux
9	Distributed	5	2.04	8	7	chrome	linux

Figure 3.10: Raw Data

For browser we found that Firefox slower than Chrome, the average time of slower = (sum of all one type mode and first browser)/(sum of all one type mode and second browser) = 1.18.

For operating systems we found that Linux faster than Windows, the average time of faster = (sum of all one type mode and first type OS)/(sum of all one type mode and second type OS) = 0.97.

We increased RAM and number of machines these affect time to decrease about two times for 8 RAM and four times for 12 RAM.

We created formulas for finding time:

For Linux OS, 8 RAM and Firefox browser(time) = ((N of test)*1.18*0.97)/2

For Linux OS, 12 RAM and Firefox browser(time) = ((N of test)*1.18*0.97)/4

For Linux OS, 4 RAM and Firefox browser(time) = ((N of test)*1.18*0.97)

We expected this result that development test automation with distributed systems more effective than series mode running. We saved our time for running all automation tests. We can also use numerous workstations to review and run tests in different browsers at the same time.

Previous similar research articles have shown alternative frameworks or how to run in series mode, but we focused on and grasped how to construct automated tests with distributed systems in our study.

We may claim that this distributed approach will be relevant in the future because as the number of automation tests grows, so will the timing. To reduce the timing of running automation tests, we should use distributed mode for running on several machines.

4. Conclusion

4.1 Distributed mode

We created automation tests using distributed systems, which aids automation test developers in creating a successful test automation environment. This is preferable than implementing all test cases on a single machine, as there may be restrictions at some point, and one single machine may not be adequate to run all test cases. All automation tests are conducted on various remote workstations and in various browsers such as Chrome and Firefox. We reduced the time it took to run automation tests and evaluated them in several browsers.

4.2 Integrated applications

In methods section we design automation test with distributed systems integrated with Jenkins CI/CD, Docker containers, Docker images, Selenium grid, Eclipse IDE, Maven, test runner TestNG and browsers.

4.3 Dependency

Also, we made analysis of dependency RAM and quantity of containerized machines. If increased our RAM, we will run more containerized machines in laptop or computer. We answered our research question, we found which factors affect on execution time in distributed automated test area. Time and other factors of tests: mode, number of test, number of RAM, number of machines, browser and OS.

4.4 Automated test cases

We understood how we can write automated code for test cases with using Selenium framework and how to find elements in browser by locators like xpath, css selectors, id, name, tag name and etc. Also, how we can use test runner like TestNG and how configure it. We learned about maven project builder, how this project builder help in integration with Jenkins. We learned how configured Jenkins and integrate with local project. We integrated Selenium grid with Docker containers. We configured hub and browser node images in Docker. We made local ip with port for all containerized machines in test environment.

We wrote many automated test cases for showing how we can develop test automation with distributed systems. We run different quantity of automated test cases in test environment. We compared time of running in distributed mode and series mode.

4.5 Future work

In future work if we have a lot of automation tests, we can use Kubernetes for managing containerized applications at scale because we had Docker containers which containerized Selenium grid hub and browser nodes.

Also, we can make analysis of performance test. This strategy help us in understanding how many quantity of automated test cases we can run in one containerized machine.

A. Appendix A

A.1 Integration

Selenium grid integration in FigureB.10.

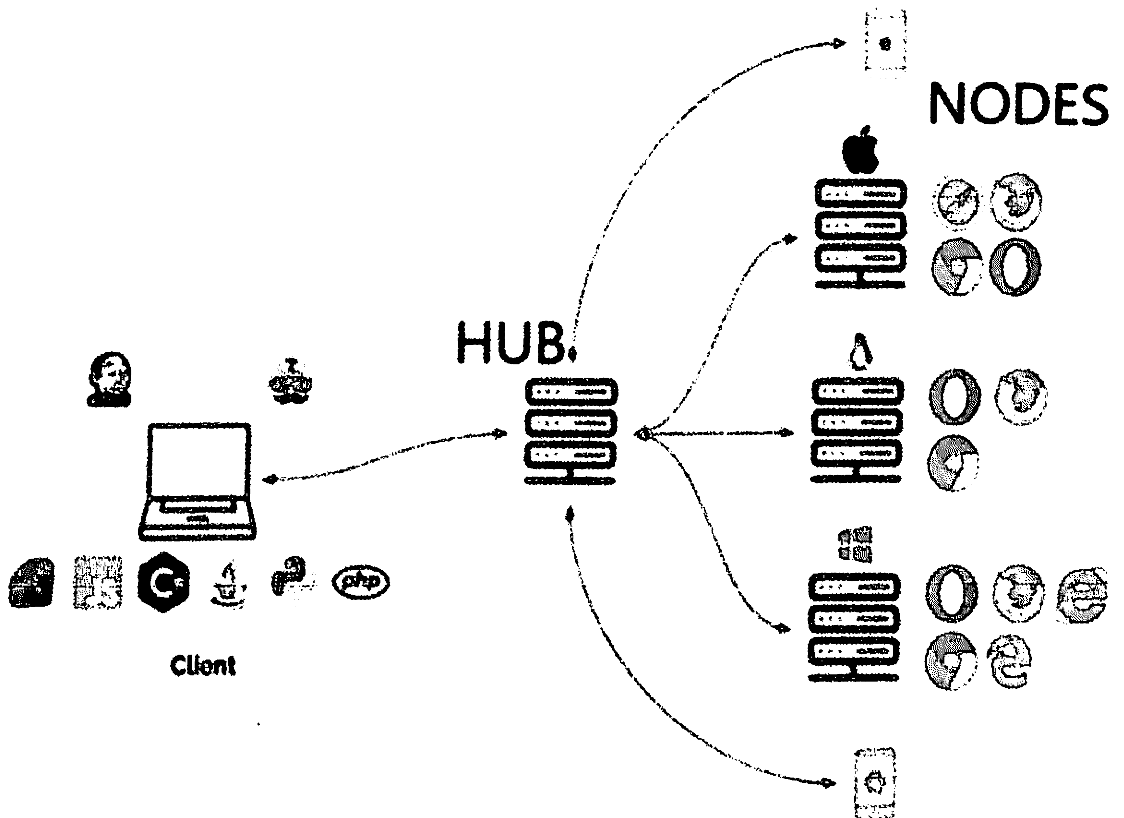


Figure A.1: Integration

B. Appendix B

B.1 Time execution automated test

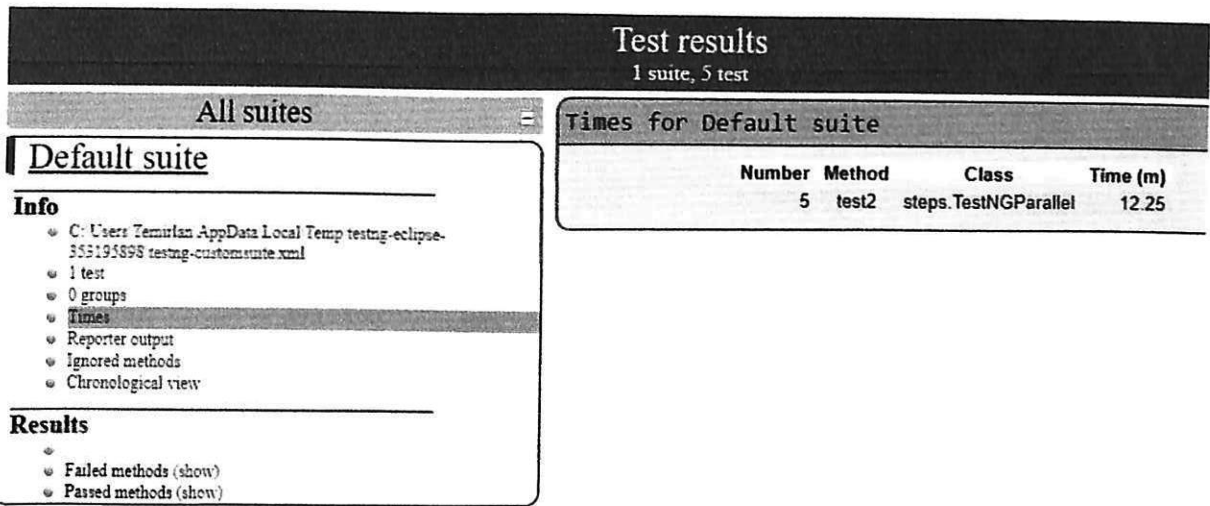


Figure B.1: Time execution automated test

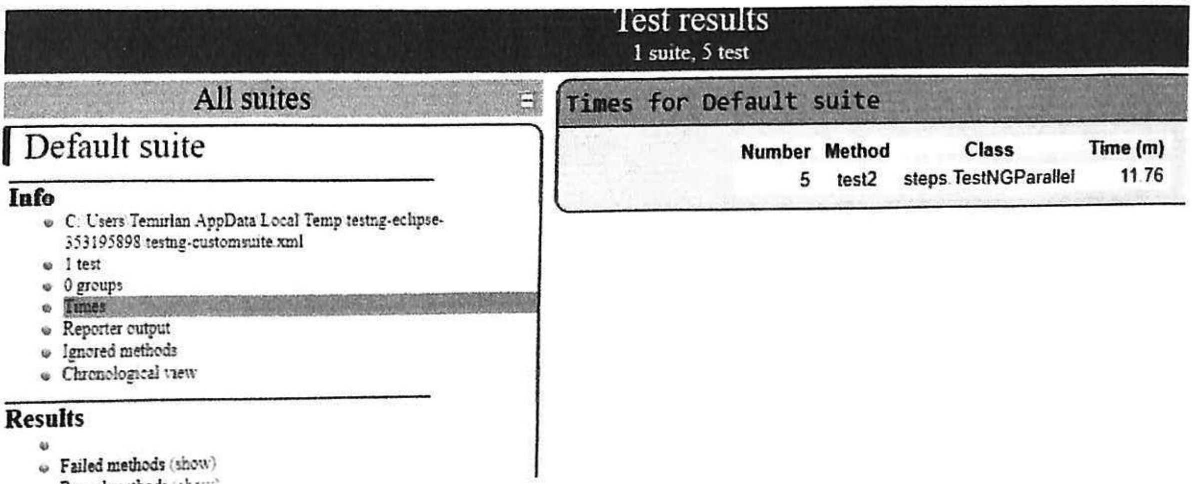


Figure B.2: Time execution automated test

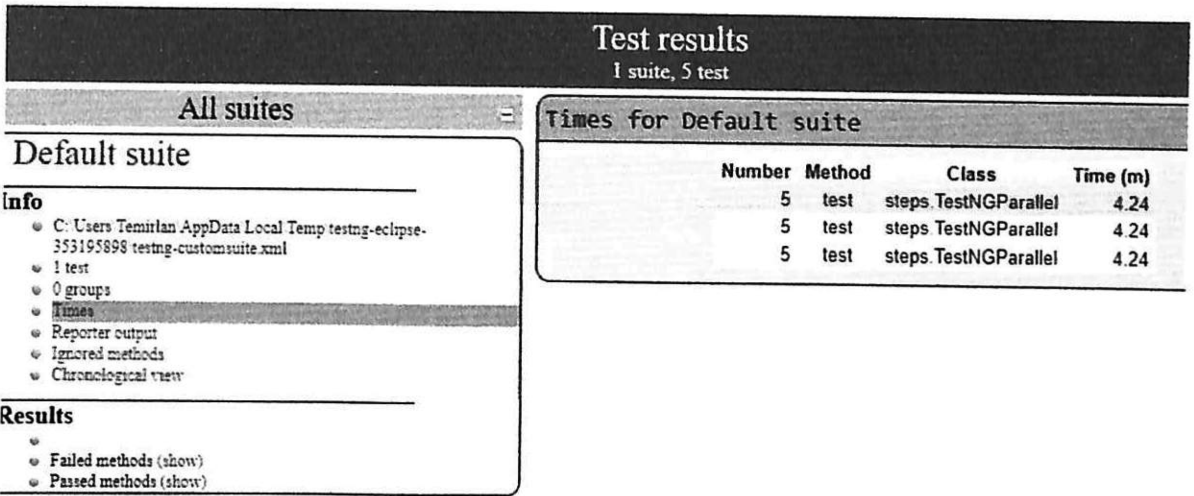


Figure B.3: Time execution automated test

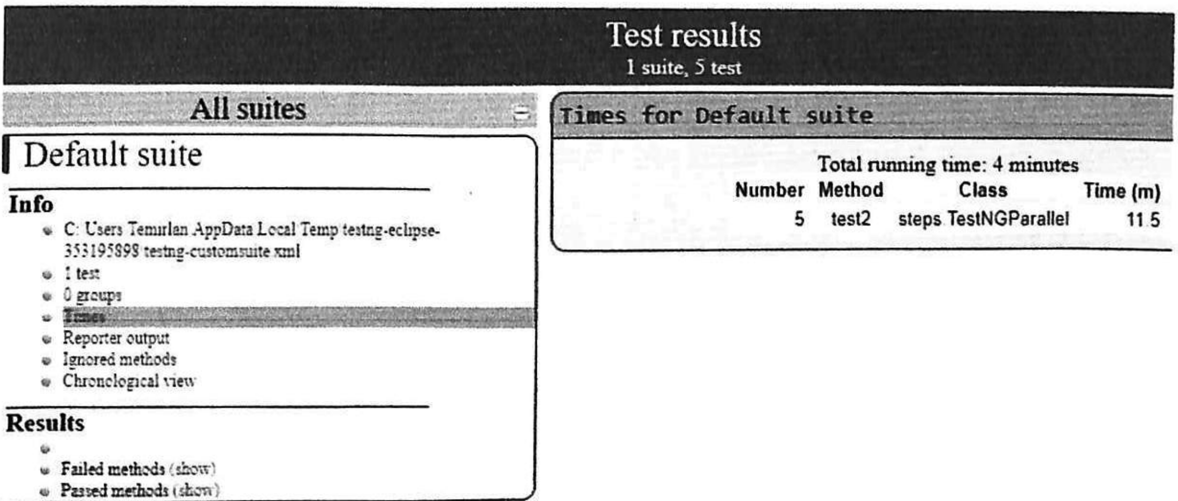


Figure B.4: Time execution automated test

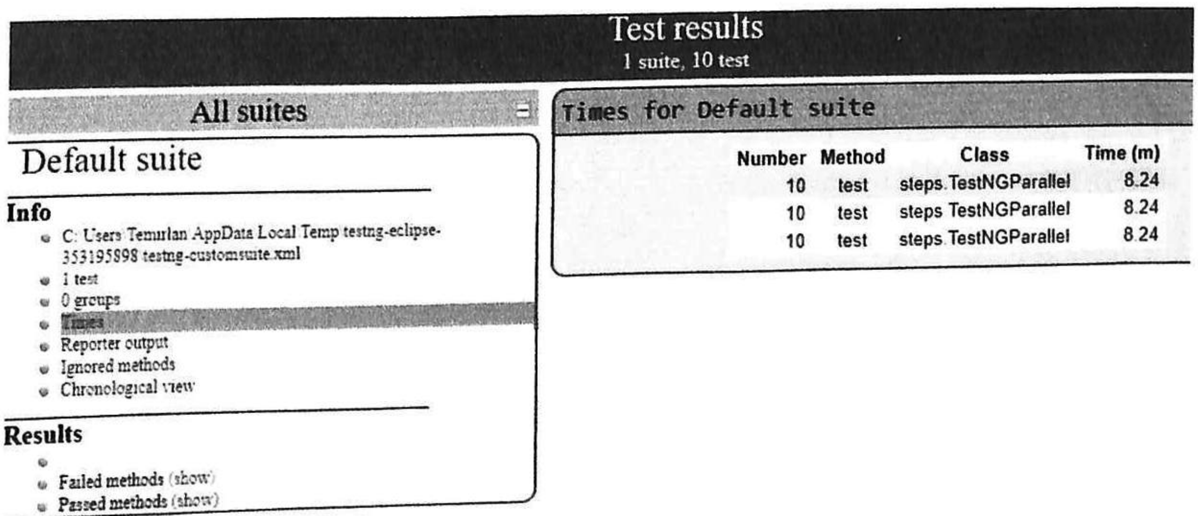


Figure B.5: Time execution automated test

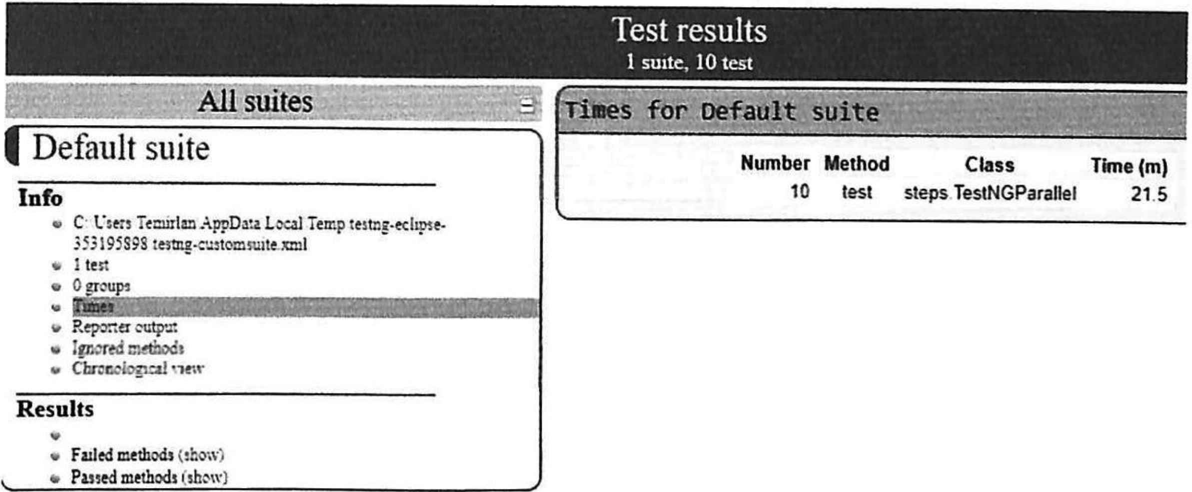


Figure B.6: Time execution automated test

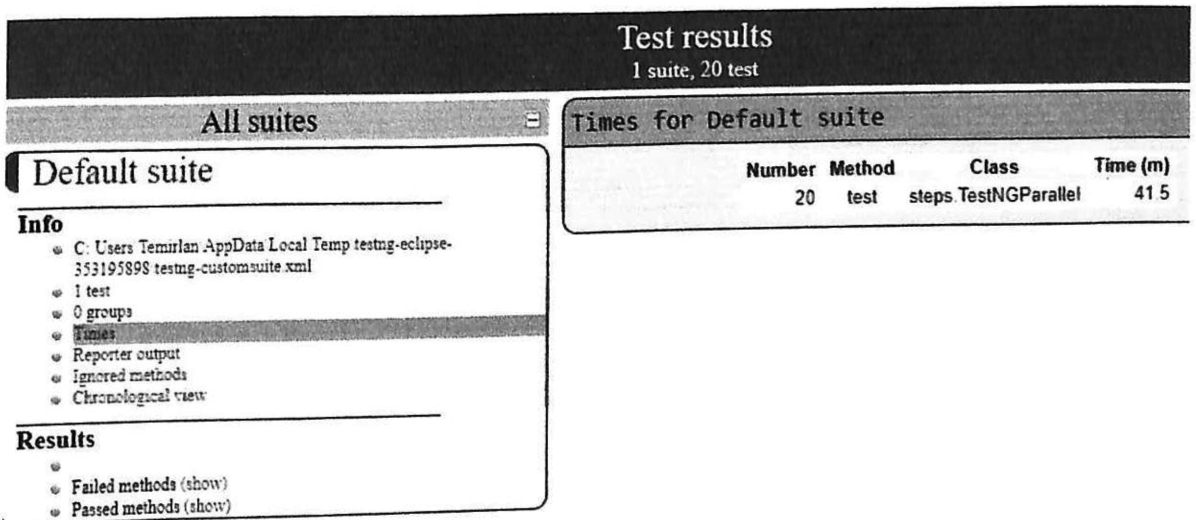


Figure B.7: Time execution automated test

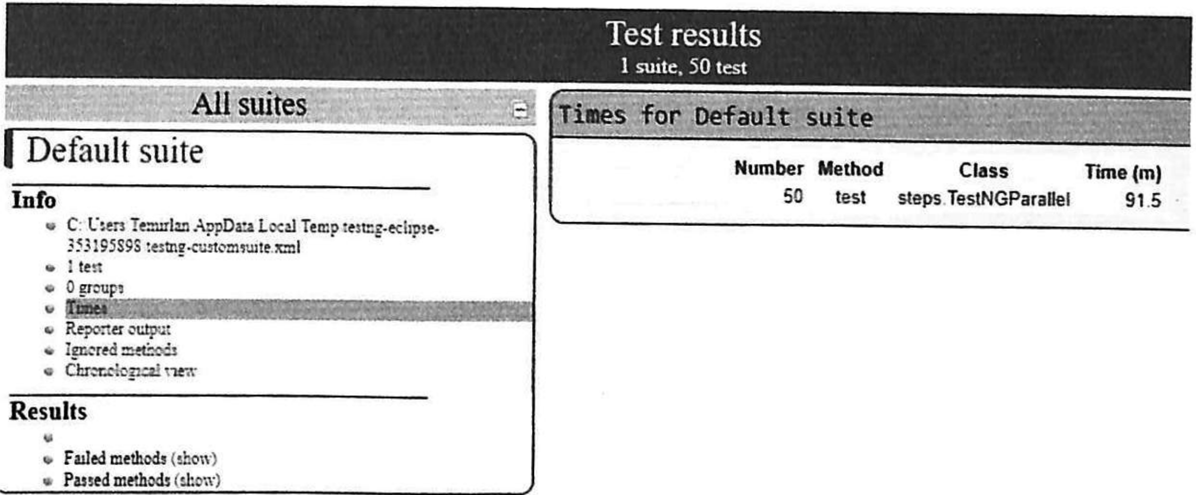


Figure B.8: Time execution automated test

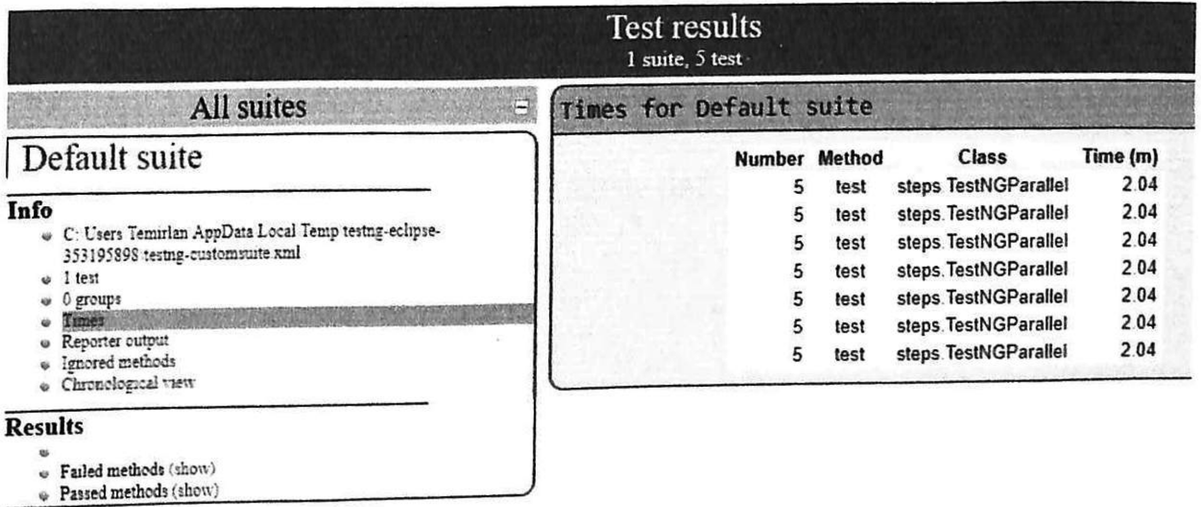
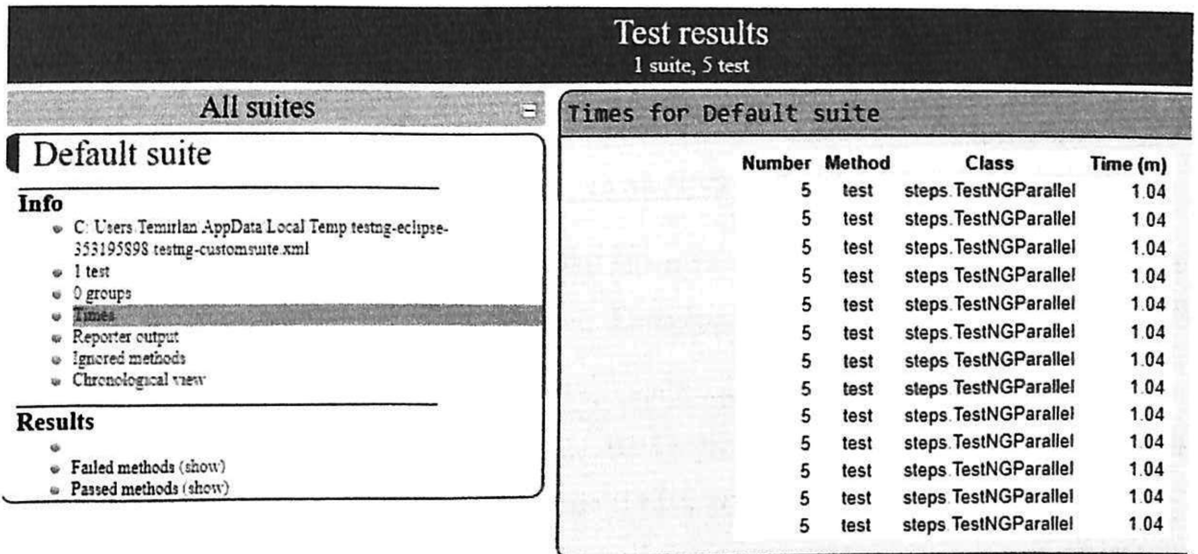


Figure B.9: Time execution automated test



Number	Method	Class	Time (m)
5	test	steps.TestNGParallel	1.04
5	test	steps.TestNGParallel	1.04
5	test	steps.TestNGParallel	1.04
5	test	steps.TestNGParallel	1.04
5	test	steps.TestNGParallel	1.04
5	test	steps.TestNGParallel	1.04
5	test	steps.TestNGParallel	1.04
5	test	steps.TestNGParallel	1.04
5	test	steps.TestNGParallel	1.04
5	test	steps.TestNGParallel	1.04
5	test	steps.TestNGParallel	1.04
5	test	steps.TestNGParallel	1.04
5	test	steps.TestNGParallel	1.04

Figure B.10: Time execution automated test

References

[1] Victoria Bezsmolna. *Parallel Testing.* URL: https://smartbear.com/blog/parallel-testing-and-why-you-should-adopt-it. (accessed: 03.04.2022).

[2] Dietmar Winkler; Reinhard Hametner; Thomas Östreicher; Stefan Biffl. "A framework for automated testing of automation systems". In: *2010 IEEE 15th Conference on Emerging Technologies Factory Automation (ETFA 2010)* 1.1 (2010), p. 15. DOI: 10.1109/ETFA.2010.5641264.

[3] Lundberg L. Damm L.-O. "Quality Impact of Introducing Component-Level Test Automation and Test-Driven Development". In: Proc. EuroSPI, 2007. Chap. 1.

[4] Selenium dev. *Selenium Grid 4.* URL: https://www.selenium.dev/documentation/grid/. (accessed: 03.04.2022).

[5] Bas Dijkstra. *Testers to learn test automation.* URL: https://techbeacon.com/app-dev-testing/why-its-still-so-difficult-testers-learn-test-automation. (accessed: 03.04.2022).

[6] Cucumber documents. *Cucumber documents.* URL: https://cucumber.io/. (accessed: 03.04.2022).

[7] Elements documents. *Elements documents.* URL: https://www.selenium.dev/documentation/webdriver/elements/. (accessed: 03.04.2022).

[8] Jenkins documents. *Jenkins documents.* URL: https://www.jenkins.io/. (accessed: 03.04.2022).

[9] Locators documents. *Locators documents.* URL: https://www.selenium.dev/documentation/webdriver/elements/locators/. (accessed: 03.04.2022).

[10] Maven documents. *Maven documents.* URL: https://maven.apache.org/. (accessed: 03.04.2022).

[11] RemoteWebDriver documents. *RemoteWebDriver documents.* URL: https://www.selenium.dev/documentation/webdriver/remote_webdriver/. (accessed: 03.04.2022).

[12] Wait documents. *Wait documents.* URL: https://www.selenium.dev/documentation/webdriver/waits/. (accessed: 03.04.2022).

[13] Tomas Fernandez. *Revving up Continuous Integration with Parallel Testing.* URL: https://semaphoreci.com/blog/revving-up-continuous-integration-with-parallel-testing. (accessed: 03.04.2022).

[14] javatpoint. *javatpoint documents.* URL: https://www.javatpoint.com/java-tutorial. (accessed: 03.04.2022).

[15] DZhenyu Liu; Qiang Chen; Xu Jiang. "A Maintainability Spreadsheet-Driven Regression Test Automation Framework". In: *2013 IEEE 16th International Conference on Computational Science and Engineering* 1.1 (2013), p. 38. DOI: 10.1109/CSE.2013.175.

[16] Prachi Paigude; Vaijayanti Gajul; Jitendra Mishra; Suhas Katkar. "Software Integration Test Report Analysis Automation Using Python". In: *2021 Asian Conference on Innovation in Technology (ASIANCON)* 1.1 (2021), p. 24. DOI: 10.1109/ASIANCON51346.2021.9544984.

[17] matplotlib. *matplotlib documents.* URL: https://matplotlib.org/3.5.0/tutorials/introductory/pyplot.html. (accessed: 03.04.2022).

[18] python. *python documents.* URL: https://www.python.org/. (accessed: 03.04.2022).

[19] SeleniumHQ docker selenium. *Docker images for the Selenium Grid Server.* URL: https://github.com/SeleniumHQ/docker-selenium. (accessed: 03.04.2022).

[20] Temirlan Meiramuly Shaikenov. "DEVELOPMENT OF TEST AUTOMATION WITH DISTRIBUTED". In: *XIX International Multidisciplinary Conference "Innovations and Tendencies of State-of-Art Science"* 1.1 (2022), pp. 40–50. DOI: DOI:10.32743/NetherlandsConf.2022.5.19.339505.

[21] Shynggyskhan Nurlanuly Turganbekov Temirlan Meiramuly Shaikenov Ulbike Sherakyzy Zhanysbek. "The best framework for QA automation testing: Advantages and disadvantages of Robot framework." In: *Suleyman Demirel University Bulletin: Natural and Technical Sciences* 57.4 (2022), pp. 20–24. DOI: https://doi.org/10.47344/sdubnts.v57i4.629.

[22] Software Testing. *Integrate Selenium Grid With Docker.* URL: https://www.softwaretestinghelp.com/docker-selenium-tutorial/. (accessed: 03.04.2022).

[23] TestNG. *TestNG documents.* URL: https://testng.org/doc/. (accessed: 03.04.2022).