

ENSE 375 Group D

Activity 4

April 5th, 2021

<https://github.com/NolanFlegel/ENSE375-GroupD>

Learning Outcomes

Nolan Flegel:

Jenkins Pipeline:

To facilitate collaboration, we created an AWS EC2 instance and installed Jenkins where all of our group could access the tools freely. This involved learning about how Amazon AWS worked and setting up security groups, port forward and user accounts. Inside the AWS management console it was necessary to configure the security group to allow tcp traffic on ports 80, 8080, 4444 and 4445 for all of the Jenkins tools. After creating the EC2 instance we found that the t2.micro instance was too small and the server would become non responsive. It was necessary to upgrade to a t3.small instance. From the command console, the ipTables were modified to allow port forwarding of http traffic to the Jenkins configuration console page. Once all of the EC2 configuration was completed and the necessary tools installed we were able to access the Jenkins configuration portal and begin working on Activity 4.

The RiskMeter application tasks involved debugging modules completed in Activity 3 as we discovered new issues when App.java was implemented. We used the Jenkins pipeline with maven running in a docker container to run our Junit tests and complete our TDD process on implementing the complete Risk Meter application. The next step in the pipeline involved publishing our app to DockerHub using Jenkins.

Jenkins					Pipelines	Administration	Logout	
AutomatedJenkins ☆ ⚙					Activity	Branches	Pull Requests	
STATUS	RUN	COMMIT	BRANCH	MESSAGE	DURATION	COMPLETED		
🕒	19	4b20096	main	made by mistake	2 commits 1h 2m 7s	-	🕒	
✅	4	4e74a33	probely-pipeline	Branch indexing	2m 45s	an hour ago	🔄	
🕒	1	670fee9	JmeterTests	Branch indexing	1h 2m 7s	-	🕒	
✅	2	a48bff4	SeleniumTests	Update TestDuckDuckGo.java	40s	an hour ago	🔄	
✅	3	4e74a33	probely-pipeline	Update Jenkinsfile	2m 22s	4 hours ago	🔄	
❌	2	ed0ed4a	probely-pipeline	Started by user ense375	7s	4 hours ago	🔄	
❌	18	20aadf8	main	Update Jenkinsfile	18s	5 hours ago	🔄	
❌	17	ff40631	main	Testing if jenkins is updating	2 commits 20s	5 hours ago	🔄	
❌	16	13cd16e	main	Added Dockerfile, Updated JenkinsFile	5s	6 hours ago	🔄	
❌	15	2d41743	main	Replayed #14	6s	6 hours ago	🔄	
❌	14	2d41743	main	Update Jenkinsfile for DockerHub	10s	6 hours ago	🔄	
⏮	13	b85c250	main	Replayed #12	14m 25s	6 hours ago	🔄	
❌	12	b85c250	main	Selenium Tests init	4 commits 17s	7 hours ago	🔄	
❌	1	ed0ed4a	probely-pipeline	Branch indexing	29s	7 hours ago	🔄	
❌	1	706de6a	SeleniumTests	Branch indexing	17s	7 hours ago	🔄	
✅	11	d8db737	main	Replayed #10	25s	a day ago	🔄	
❌	10	d8db737	main	all tests pass	3 commits 48s	a day ago	🔄	
❌	9	8fa214d	main	Replayed #8	12h 53m 30s	a day ago	🔄	

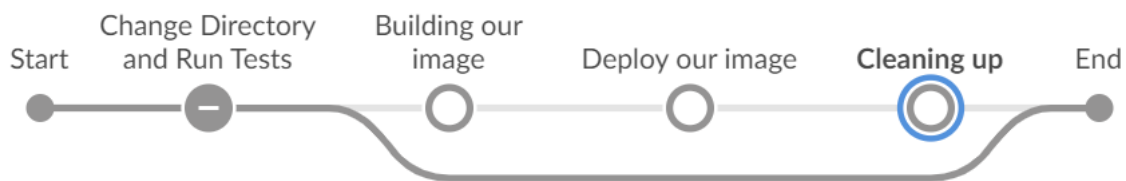
Branch: main 🔗	🕒 25s	Changes by 55856533+Jacobchapman99, flegel2n
Commit: d8db737	🕒 a day ago	Replayed #10



Change Directory - 19s

[Restart Change Directory](#) [🔗](#) [📄](#)

✓	> Check out from version control	<1s
✓	> mvn clean test — Shell Script	19s



After working on deploying our app to DockerHub we explored several unit testing plugins. We created different branch in Github so that we could create and maintain separate jenkinsfiles and pipelines for each of these tools. This also allowed us to create separate maven pom.xml files for each tool and test different implementations without altering the RiskMeter project.

Selenium:

The Selenium Grid plugin was difficult to install and is out of security compliance for several standards. It appears that the development of this tool has been abandoned. We were able to create Selenium nodes for Internet explorer, Chrome and Firefox WebDrivers and connect the nodes to the Selenium Hub. However when attempting to connect to the remote driver protocols using java and Junit tests, the tests would result in errors with remote connections refused. We attempted a number of solutions found on various support forums and technical documents however we were unable to get the remote drivers to work.


The selenium tests we wrote were tested on a local pc with installations of Chrome and Firefox. We were able to perform successful tests using Junit and maven. This was a proof of concept for the tool. We were unable to convert this method to use the remote driver tool of Selenium Nodes running in Jenkins due to the aforementioned connection issue. We believe that the implementation should be possible using a different cloud service and firewall tools.

Selenium Configuration

Running configurations

Name ↓	Status	Environment variables	JVM options	Selenium options	Service actions
SeleniumNodes	Started		webdriver.ie.driver=	-port 4445 -maxSession 5 -browser seleniumProtocol=WebDriver,browserName=internet explorer,maxInstances=1 -browser seleniumProtocol=WebDriver,browserName=firefox,maxInstances=5 -browser seleniumProtocol=WebDriver,browserName=chrome,maxInstances=5 -host ip-172-17-0-1.ec2.internal	<div>Restart</div> <div>Start</div> <div>Stop</div>

Matching configurations

Type	Name ↓	Matching type	Description summary	Actions
	SeleniumNodes	Match all nodes	1 instances of Internet Explorer (version : Not specified) 5 instances of Firefox (version : Not specified) 5 instances of Chrome (version : Not specified)	<div>Restart</div>

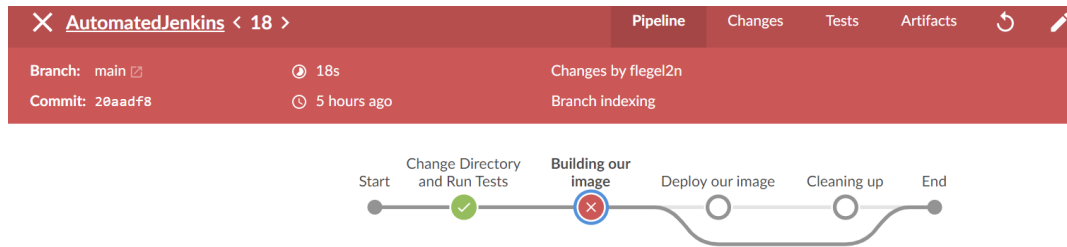
RiskMeter:

I aided in debugging a number of modules and code refactoring. We refactored PostalCode.java to use regex statements and simplified the exception handling for a few statements. We used the TDD process to systematically track down the different errors and we appreciated how the process helped simplify our code and our focus. By working through one error at a time, we were able to improve our code and refactor as we went.

Shane Toma:

Deploying to Docker Hub

To deploy the maven application to docker hub a few things were required. First we had to create a docker hub repository and create a dockerfile for the project. Then, stages had to be added to the projects jenkins file to expand the job to automatically deploy a containerized version of the java project to the docker hub repo. Originally a stage to clone the github repo was included but since we were already automatically pulling changes from our repo it was redundant and removed. Other stages added include building the docker image and deploying the docker image. Errors that arose included not having credentials properly connected to the pipeline and mistakenly providing github credentials where docker hub credentials were required. Both issues were solved rather quickly. Unfortunately due to a limited knowledge of Jenkins files there were issues that we did not manage to overcome, while trying to follow a tutorial symbolic links became involved in the process and an error stating “unable to evaluate symbolic links” brought progress to a halt. Ultimately we did not succeed in reaching a functional state for the jenkins file and so did not succeed in deployment.



Jacob Chapman:

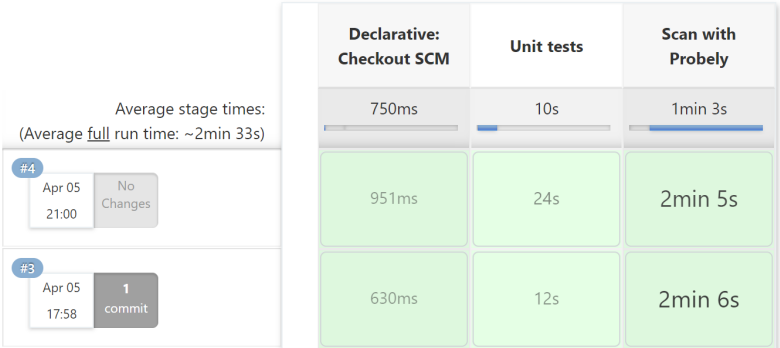
addPatient.java implementation and fixes

Continuing from activity 3, we had to integrate the pieces together, i.e. having a working pipeline inside Jenkins that had our application. With this, there were errors upon building the application, especially when testing add patient and delete patient methods. I was responsible for making sure the addPatient method complied with the tests written. There were some inconsistencies inside the method, to which I managed to fix and got the tests to pass after some time.

Security Plugin

A part of the tasks for this activity was to look at 3 different plugins and configure them into Jenkins to test with our application. As a group, we chose to set up and configure Probely as our security plugin. The Probely plug-in that is a part of Jenkins is a security scanner that scans for any vulnerabilities in the application. There was a tutorial inside the Jenkins documentation (<https://plugins.jenkins.io/probely-security/>) that guided the process of configuring it to a Jenkins pipeline. The structure behind using this plug in is by providing the Jenkinsfile specific blocks of code that tells which docker container to run, where the POM file is located under a certain directory, and finally to run the security scanner. In a Jenkinsfile, these are known as tasks that the pipeline will run. Upon running the plug-in with our Jenkinsfile, there were some errors encountered which were mostly related to build errors. These were eventually resolved by providing the proper location of the POM file. Additionally, the security scanning task failed in the end and due to limited knowledge and time, we were unable to overcome this issue. The figure below displays the running stages of Probely at its minimum. This functionality can be expanded as well.

Stage View



At its minimum, probably can scan for any vulnerabilities on an application, and in our case we scanned our repo for any viruses, vulnerabilities, and other security factors. The figure shown below shows the logs from our most successful build, however it failed in the end. This issue never got resolved.

Stage Logs (Scan with Probely)

Probely Security Scanner (self time 2min 5s)

```
Probely Security Scanner: Requesting scan for target: 3Ky3mxM6gERJ
Probely Security Scanner: Requested scan: id: fQUopLgbnH2H | status: queued | vulnerabilities: high: 0, medium: 0, low: 0
Probely Security Scanner: Scan progress details: id: fQUopLgbnH2H | status: queued | vulnerabilities: high: 0, medium: 0, low: 0
Probely Security Scanner: Scan progress details: id: fQUopLgbnH2H | status: started | vulnerabilities: high: 0, medium: 0, low: 0
Probely Security Scanner: Scan progress details: id: fQUopLgbnH2H | status: started | vulnerabilities: high: 0, medium: 0, low: 0
Probely Security Scanner: Scan has finished. Details: id: fQUopLgbnH2H | status: failed | vulnerabilities: high: 0, medium: 0, low: 0
```

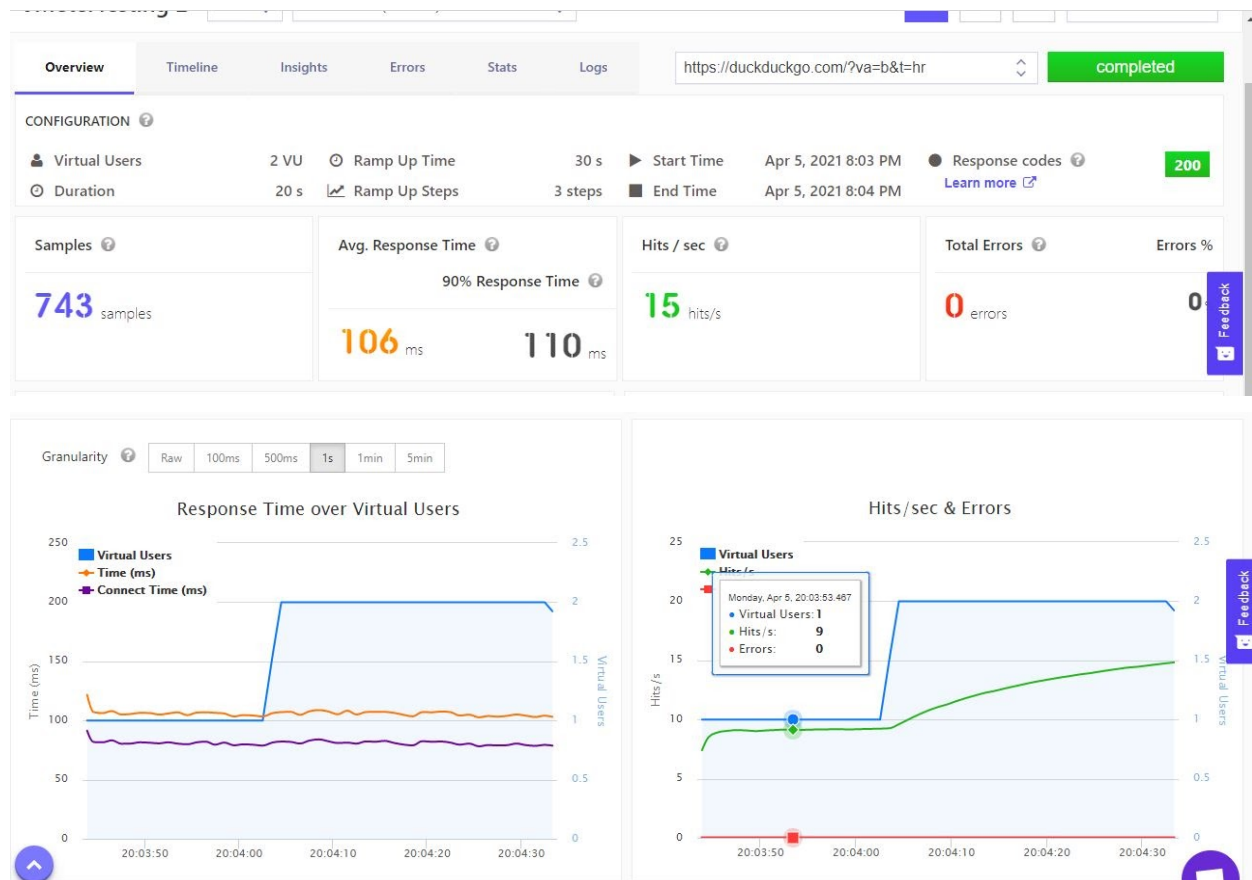
Krupalkumar Patel:

removePatientTest.java

For my first part implementation of the test cases for removing patients using the patient id which had to be removed. For that add a patient in the arraylist and then attempt to remove it. Also, when our code was run there were several errors found. I had aided in debugging the code with the other team members.

JMeter Configuration and test

JMeter is available in Jenkins as a plugin software for this part of the activity there were two ways the implementation was executed Website used to test the performance is (<https://duckduckgo.com/?va=b&t=hr>). Firstly, downloading JMeter and testing the website by placing duration, users, repetitions. Using our github repository as a source for a .JML file which is the script of the test which can be found in ENSE375-GroupD/tree/JmeterTests. For the setup a separate pipeline was created and git was configured but that resulted in errors. The second method was using a cloud service to run JMeter tests for the performance of DuckDuckGo website. The resulting tests in cloud service (Load Focus) were successful. As seen in the figures below.



The number of users were 2, duration of the test was 20 seconds and each step was executed 3 times. During the implementation of LoadFocus on our Jenkins server an API key of the account was added in the credentials. New pipeline was set up and used to run this JMeter test. A new project was configured with the errors for stable and unstable build. The JMeter showed errors after a build action was executed but the tests in the LoadFocus cloud had passed without errors.

Console Output

```
Started by user ense375
Running as SYSTEM
Building in workspace /var/lib/jenkins/workspace/JmeterTesting
loadfocus.com: Test Started: -1
loadfocus.com: Test Config: Build UNSTABLE if errors percentage greater than or equal to 3%
loadfocus.com: Test Config: Build FAILURE if errors percentage greater than or equal to 5%
loadfocus.com: Test Config: Build UNSTABLE if response time greater than or equal to 500ms
loadfocus.com: Test Config: Build FAILURE if response time greater than or equal to 10000ms
ERROR: Build step failed with exception
net.sf.json.JSONException: Invalid JSON String
    at net.sf.json.JSONSerializer.toJSON(JSONSerializer.java:143)
    at net.sf.json.JSONSerializer.toJSON(JSONSerializer.java:103)
    at net.sf.json.JSONSerializer.toJSON(JSONSerializer.java:84)
    at com.loadfocus.jenkins.api.LoadAPI.runTest(LoadAPI.java:196)
    at com.loadfocus.jenkins.LoadPublisher.perform(LoadPublisher.java:78)
    at hudson.tasks.BuildStepMonitor$3.perform(BuildStepMonitor.java:45)
    at hudson.model.AbstractBuild$AbstractBuildExecution.perform(AbstractBuild.java:803)
    at hudson.model.AbstractBuild$AbstractBuildExecution.performAllBuildSteps(AbstractBuild.java:752)
    at hudson.model.Build$BuildExecution.post2(Build.java:177)
    at hudson.model.AbstractBuild$AbstractBuildExecution.post(AbstractBuild.java:697)
    at hudson.model.Run.execute(Run.java:1931)
    at hudson.model.FreeStyleBuild.run(FreeStyleBuild.java:43)
    at hudson.model.ResourceController.execute(ResourceController.java:97)
    at hudson.model.Executor.run(Executor.java:429)
Build step 'JMeter Load Testing in Cloud by LoadFocus' marked build as failure
Finished: FAILURE
```

The console output showed that the JMeter code did not execute successfully even though the tests in the cloud server had passed without errors.

Renz Rivero:

deletePatient method

With Krupal's test cases for the deletePatient method, I implemented a working implementation of the deletePatient on my local machine as well as made changes to my previous contribution of implementing PostalCode.java. After merging my branch to main, my implementation overall caused some errors that the group were able to fix.

```

renz@Renzs-MacBook-Pro RiskMeter % java -jar junit-platform-console-standalone-1.7.0.jar -cp bin --scan-class-path
Patient Added.
Patient 123456789 Found.
Patient Added.
Patient lskjflkj Is Not on The List.
Patient Not Found
Patient Added.
Patient Is Not on The List.
Patient Not Found
Patient Added.
Patient 123456780 Is Not on The List.
Patient Not Found
Patient Added.
Patient 12345678 Is Not on The List.
Patient Not Found
Patient Added.
Patient 1234567899 Is Not on The List.
Patient Not Found

Thanks for using JUnit! Support its development at https://junit.org/sponsoring

- JUnit Jupiter ✓
  - RemovePatientTest ✓
    - shouldAnswerWithTrue() ✓
    - deleteIfId_True() ✓
    - dontDeleteIfId_Invalid() ✓
    - dontDeleteIfId_Empty() ✓
    - dontDeleteIfId_False() ✓
    - dontDeleteIfId_Short() ✓
    - dontDeleteIfId_Long() ✓
  - JUnit Vintage ✓

Test run finished after 94 ms
[ 3 containers found ]
[ 0 containers skipped ]
[ 3 containers started ]
[ 0 containers aborted ]
[ 3 containers successful ]
[ 0 containers failed ]
[ 7 tests found ]
[ 0 tests skipped ]
[ 7 tests started ]
[ 0 tests aborted ]
[ 7 tests successful ]
[ 0 tests failed ]

renz@Renzs-MacBook-Pro RiskMeter %

```