Lab 8: Final Project

In this final project, you will use all the knowledge you have learned about Git, GitLab CI/CD, Gradle, and Docker to develop a Java application with Gradle, build a CI/CD pipeline that will build, test, and deploy your application to Docker Hub.

You may need to read some documentations about how to do certain tasks but most of the steps are using what you have already learned.

Please follow the instructions and provide necessary code/scripts/screenshot to demonstrate what you did and/or the result(s) of the steps:

1. Clone your repository to local. Create a “Lab8-dev” branch off “main”. Delete everything.

Graphical user interface, application

Description automatically generated

Text

Description automatically generated

1. Create a “Lab8-production” branch off “Lab8-dev”.

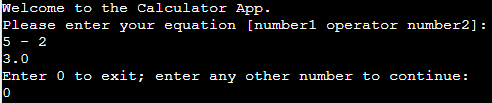


1. On “Lab8-dev” branch, use “gradle init” to set up a Java application project.

Text

Description automatically generated

1. Write a simple Java application. It is in the package “lab8”.
   1. The project name is “calculator”.
   2. This project should have 2 classes: App.java and Calculator.java.
   3. Calculator class should have 2 methods: Add and Subtract.
   4. App.java has a main function that will interact with the user, which looks like this:



List your all .java files created during this step.

Graphical user interface, text, application

Description automatically generated

|  |
| --- |
| package lab8;  import java.util.InputMismatchException;  import java.util.Scanner;  import java.util.StringTokenizer;  import java.lang.\*;  public class App {      public static void main(String[] args) {          Calculator math = new Calculator();          float op1;          String operator;          float op2;          String equation;          StringTokenizer equTokens;          String continueInput;          StringTokenizer contInputTokens;          int isEquationlooping = 1;          boolean isEndInputlooping = true;          Scanner scanner = new Scanner(System.in);          System.out.println("Welecome to the Calculator App.");          while(isEquationlooping != 0) {              System.out.println("\nOperations supported: addition, subtraction.");              System.out.println("Please enter your equation [number1 operator number2]:");              try {                  equation = scanner.nextLine();                  equTokens = new StringTokenizer(equation);                  if(equTokens.countTokens() < 3)                      throw new TooFewArgsException();                  if(equTokens.countTokens() > 3)                      throw new TooManyArgsException();                    op1 = Float.parseFloat(equTokens.nextToken());                  operator = equTokens.nextToken();                  op2 = Float.parseFloat(equTokens.nextToken());                  if(operator.equals("+"))                      System.out.println(math.Add(op1, op2));                  else if(operator.equals("-"))                      System.out.println(math.Subtract(op1, op2));                  else                      throw new InvalidOpperationTypeException();              } catch (NumberFormatException e) {                  System.out.println("Incorect Input Format! Please follow the format of [number1 operator number2] \n");              } catch (TooFewArgsException e) {                  System.out.println(e.toString());              } catch (TooManyArgsException e) {                  System.out.println(e.toString());              } catch (InvalidOpperationTypeException e) {                  System.out.println(e.toString());              }            isEndInputlooping = true;              while(isEndInputlooping) {                  try {                      System.out.println("Enter 0 to exit; enter any other number to continue:");                      continueInput = scanner.nextLine();                      contInputTokens = new StringTokenizer(continueInput);                      if(contInputTokens.countTokens() > 1)                          throw new TooManyArgsException();                      if(contInputTokens.countTokens() < 1)                          throw new TooFewArgsException();                      isEquationlooping = Integer.parseInt(contInputTokens.nextToken());                      isEndInputlooping = false;                  } catch (NumberFormatException e) {                      System.out.println("\nIncorect Input Format! please enter a number!");                      isEndInputlooping = true;                  } catch (TooFewArgsException e) {                      System.out.println(e.toString());                      isEndInputlooping = true;                  } catch (TooManyArgsException e) {                      System.out.println(e.toString());                      isEndInputlooping = true;                  }              }          }          scanner.close();      }  } |
| package lab8;  public class Calculator {      public float Add(float num1, float  num2) {          return num1 + num2;      }      public float Subtract(float num1, float num2) {          return num1 - num2;      }  } |
| package lab8;  public class InvalidOpperationTypeException extends Exception {      String message;      InvalidOpperationTypeException() {          message = "Invalid Operation! only addition and subtraction are allowed";      }      @Override      public String toString(){          return message;      }  }  package lab8;  public class TooFewArgsException extends Exception {      String message;      TooFewArgsException() {          message = "Too few arguments for expression!";      }      @Override      public String toString(){          return message;      }  }  package lab8;  public class TooManyArgsException extends Exception {      String message;      TooManyArgsException() {          message = "Too many arguments for operation!";      }      @Override      public String toString(){          return message;      }  }  Each of the three classes are in three separate files with the same name as the class |
| OUTPUT |

1. Write unit tests for Calculator class. Execute the test in Gradle. Make sure all tests pass.

List your test java file.

Graphical user interface, text, website

Description automatically generated

A screenshot of a computer

Description automatically generated

Text

Description automatically generated

Text

Description automatically generated

Text

Description automatically generated

Graphical user interface

Description automatically generated with medium confidence

1. Add an appropriate .gitignore file to your project.

Graphical user interface, text, application, email

Description automatically generated

This is a basic Java gitignore from gitlab mixed with the .gitignore created from gradle

1. Create a GitLab **Gradle** CI/CD pipeline that contains 2 stages: build and test:
   1. build stage has a build job, where you use Gradle to build the project.
      * Gradle build should create a *CalculatorApp.jar* file.
   2. test stage has a test job, where you use Gradle to test the project.
   3. Save the test coverage html report as an artifact for the test job.

Hint: Don’t use the template for Gradle on GitLab.com. It is too complicated. Write your own script!

List your build.gradle script, .gitlab-ci.yml file and a screenshot of the passed job logs.

|  |  |
| --- | --- |
| Gitlab-ci.yml | |
| image: openjdk:8  stages:          *# List of stages for jobs, and their order of execution*    - build    - test  before\_script:    - chmod +x gradlew  build-job:       *# This job runs in the build stage, which runs first.*    stage: build    script:      - ./gradlew build    artifacts:      paths:        - build/libs/\*.jar  unit-test-job:   *# This job runs in the test stage.*    stage: test    *# It only starts when the job in the build stage completes successfully.*    variables:      DOWNLOAD\_NAME: "calculator-app-test-coverage"    script:      - ./gradlew test    artifacts:      paths:        - build/reports/tests/test/index.html      name: "'$DOWNLOAD\_NAME'" | |
| Build.gradle | Artifacts created |
|  |  |

Graphical user interface, text, application

Description automatically generated Text

Description automatically generated

1. Once your pipeline has passed, expand your code and unit tests to add two more calculations and corresponding unit tests to your app: multiply and divide.

List the additional code and tests.

|  |  |
| --- | --- |
|  |  |

1. Push to remote and make sure the modified project passes the pipeline.

Post a screenshot of the jobs passing.

Graphical user interface, text, application

Description automatically generated Graphical user interface, text, application, email

Description automatically generated

1. Now that you have implemented and tested your app, it is time to deploy. Build a docker image of your Calculator app. Upon start of a docker container built with this image, the container should execute *CalculatorApp.jar* file created by Gradle build.

List your Dockerfile.  
Graphical user interface, text

Description automatically generated with medium confidence

Text

Description automatically generated

1. When you run the container, you should be able to use it as in Step 4.

Show the execution console when running your docker.

Text

Description automatically generated

Extra to show that error handling works with docker

Text

Description automatically generated

1. **[Do some research on this step]** Add a new stage “deploy” to your CI/CD pipeline and make the deploy jobs merge-request only. The deploy jobs should:
   1. login to docker
   2. build a docker image
   3. publish the image to Docker Hub with the name “*you-username*/calculator”.

List your updated complete .gitlab-ci.yml file.

|  |
| --- |
| Gitlab cicd pipeline code |
| stages:          *# List of stages for jobs, and their order of execution*    - build    - test    - deploy  workflow:    rules:      - if: $CI\_PIPELINE\_SOURCE == 'merge\_request\_event'  before\_script:    - chmod +x gradlew    - export GRADLE\_USER\_HOME='pwd'/.gradle    build-job:       *# This job runs in the build stage, which runs first.*    image: gradle:8.0    stage: build    script:      - ./gradlew build    artifacts:      paths:        - build/libs/\*.jar  unit-test-job:   *# This job runs in the test stage.*    image: gradle:8.0    stage: test    *# It only starts when the job in the build stage completes successfully.*    variables:      DOWNLOAD\_NAME: "calculator-app-test-coverage"    script:      - ./gradlew test    artifacts:      paths:        - build/reports/tests/test/index.html      name: "'$DOWNLOAD\_NAME'"  deploy-job:    stage: deploy    image: docker    services:      - docker:dind    script:      - echo $DOCKERHUB\_USERNAME      - docker login -u $DOCKERHUB\_USERNAME -p $DOCKERHUB\_TOKEN      - docker build -t $DOCKERHUB\_USERNAME/calculator .      - docker push $DOCKERHUB\_USERNAME/calculator |



1. Change the setting to require pipeline pass for merge request and enable merged results pipelines.

Show a screenshot of the setting configuration on GitLab.com.

Graphical user interface, text, application, email

Description automatically generated

1. Create a merge request from “Lab8-dev” to “Lab8-production”.

Show a screenshot of the merge request.

Graphical user interface, text, application, email

Description automatically generated

1. Review the pipeline successfully runs all jobs.

Graphical user interface, text, application, email

Description automatically generated

1. Try to run a container from your published “*your-username*/calculator” image on the docker playground: <https://labs.play-with-docker.com/> .

Show a screenshot of the docker image on Docker Hub under your username, and a screenshot of the docker container running on the docker playground.



Graphical user interface, text

Description automatically generated

Congratulations! You have completed the entire course. Hopefully now you have some basic understanding about git, CI/CD, Gradle and Docker. However, this is only the starting point of using them in different scenarios you will encounter in your future study and job. The most important thing you should have learned from this course is how to utilize documentation and Internet resources to figure out a solution to a specific problem. Enjoy these tools in your future projects!