DALHOUSIE UNIVERSITY

FACULTY OF COMPUTER SCIENCE

BUILD DOCUMENTATION

PlantOne

CSCI 5308

ADVANCED TOPICS IN SOFTWARE DEVELOPMENT

WINTER 2022

INSTRUCTOR

DR. TUSHAR SHARMA

TEACHING ASSISTANT

MANJINDER SINGH

GROUP 12

SARTHAK PANDIT
B00900388, sr215260@dal.ca

JAY PATEL
B00906433,jy977367@dal.ca

FENIL PARMAR
B00895684, fn902491@dal.ca

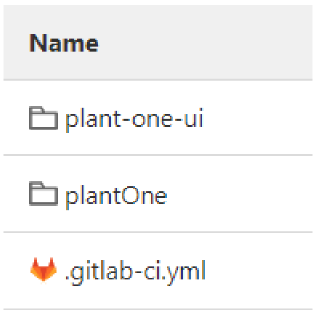
AMAN PATEL
B00888136,am501483@dal.ca

SAHIL PAREKH
B00900956, sh883193 @dal.ca

Steps for application deployment [1]-[4].

Source Stage

- In the first stage, we developed two distinct applications in a same repository.
- One is plantOne, a backend application, and the other is plat-one-ui, a frontend application.
- We used Vue.js in the frontend and spring boot in the backend.
- One more file is called **.gitlab-ci.yml** which is for project configuration. This file is placed in the root of the repository and defines the project's Pipelines, Jobs, and Environments.

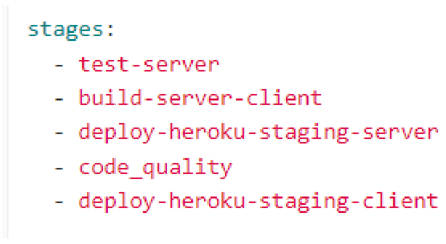


Name
plant-one-ui
plantOne
.gitlab-ci.yml

Figure 1: Source Stage

The following stages are mentioned for project testing, building, and deployment:

Our CI/CD Pipeline is divided into five sections. In every stage we are using dalfcs_docker_autoscale runner.



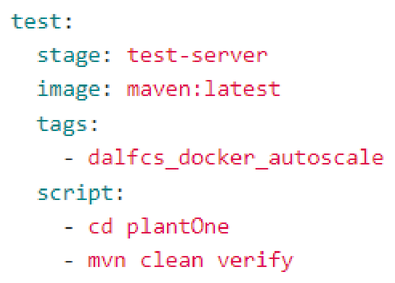
```
stages:
  - test-server
  - build-server-client
  - deploy-heroku-staging-server
  - code_quality
  - deploy-heroku-staging-client
```

Figure 2: Source Stage

1) Backend application testing (test-server)

- Here in this stage, we first install maven image so we can run maven commands.

- Next, we are running “mvn clean verify” to run test cases in spring boot application.
- Whenever we are doing changes in new branch, main branch or releasing new tag this stage will work.



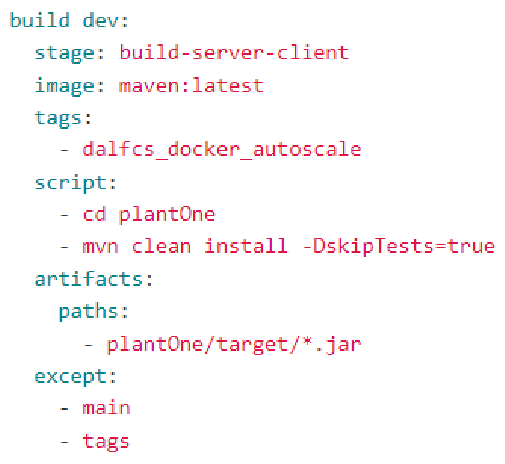
```
test:
  stage: test-server
  image: maven:latest
  tags:
    - dalfcs_docker_autoscale
  script:
    - cd plantOne
    - mvn clean verify
```

Figure 3: Backend application testing

2) Build application of backend and frontend (build-server-client)

For backend application we defined three jobs: build dev, build test, build prod

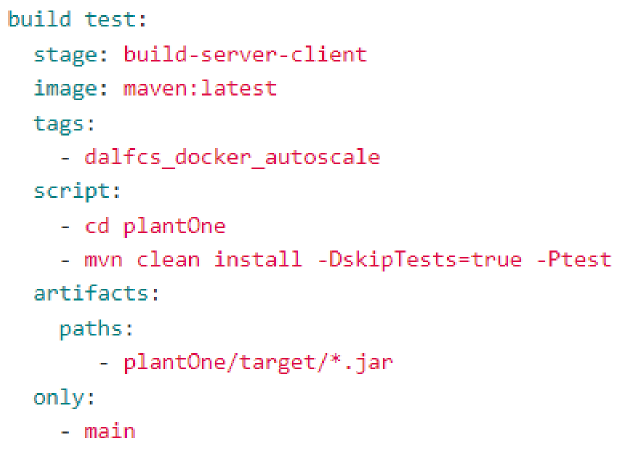
- Here, we first install the maven image in order to perform maven commands.
- We are executing “mvn clean install -DskipTests=true” and bypassing test cases because we have previously tested in the previous stage.
- Build dev will work except doing changes in main branch and releasing tags. It will work when we are creating new branch. Here by default we are not passing any profile variable, so our application will refer application.properties file. Here we added database details of developer environment.



```
build dev:
  stage: build-server-client
  image: maven:latest
  tags:
    - dalfcs_docker_autoscale
  script:
    - cd plantOne
    - mvn clean install -DskipTests=true
  artifacts:
    paths:
      - plantOne/target/*.jar
  except:
    - main
    - tags
```

Figure 4: Build application of backend dev environment

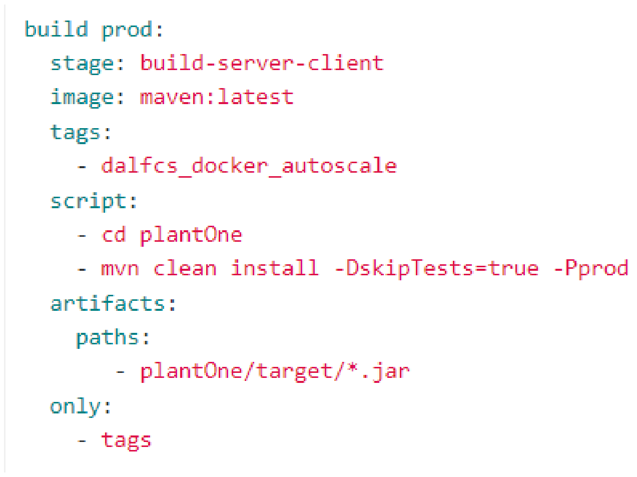
- When we make modifications to the main branch, the build test will work. Here we are passing “-Ptest” so our application will refer application-test.properties where we added database details of testing environment.



```
build test:
  stage: build-server-client
  image: maven:latest
  tags:
    - dalfcs_docker_autoscale
  script:
    - cd plantOne
    - mvn clean install -DskipTests=true -Ptest
  artifacts:
    paths:
        - plantOne/target/*.jar
  only:
    - main
```

Figure 5: Build application of backend test environment

- Build prod will function when we release a new tag. Here we are passing “-Pprod” so our application will refer application-prod.properties where we added database details of production environment.

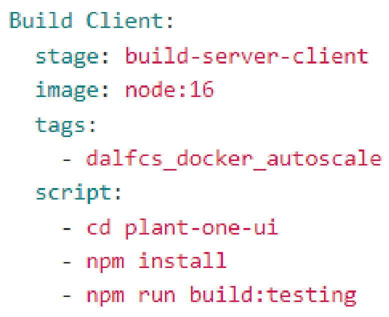


```
build prod:
  stage: build-server-client
  image: maven:latest
  tags:
    - dalfcs_docker_autoscale
  script:
    - cd plantOne
    - mvn clean install -DskipTests=true -Pprod
  artifacts:
    paths:
        - plantOne/target/*.jar
  only:
    - tags
```

Figure 6: Build application of backend production environment

For frontend application we defined one job: build client

- The build client will function in all circumstances. Here we are installing node image to run npm commands.
- Then we are running “npm install” in plant-one-ui folder and running “npm run build:testing” to start build.



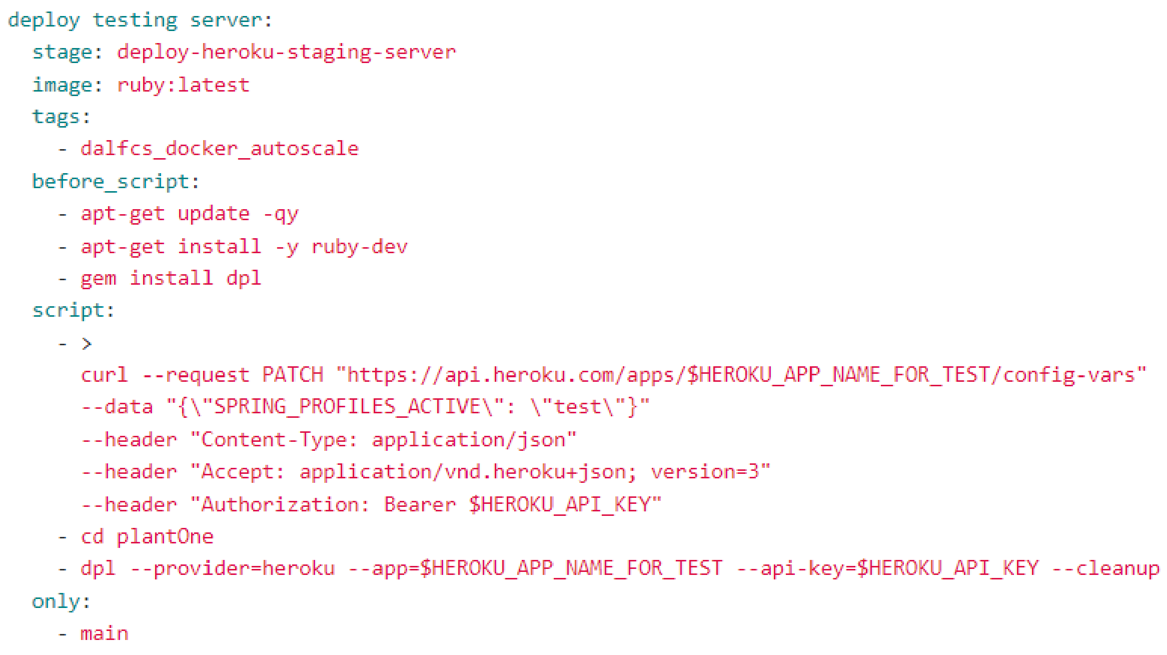
```
Build Client:
  stage: build-server-client
  image: node:16
  tags:
    - dalfcs_docker_autoscale
  script:
    - cd plant-one-ui
    - npm install
    - npm run build:testing
```

Figure 7: Build application of client

3) Deploying backend application on Heroku server. (deploy-heroku-staging-server)

Here we have two jobs: deploy testing server and deploy production server

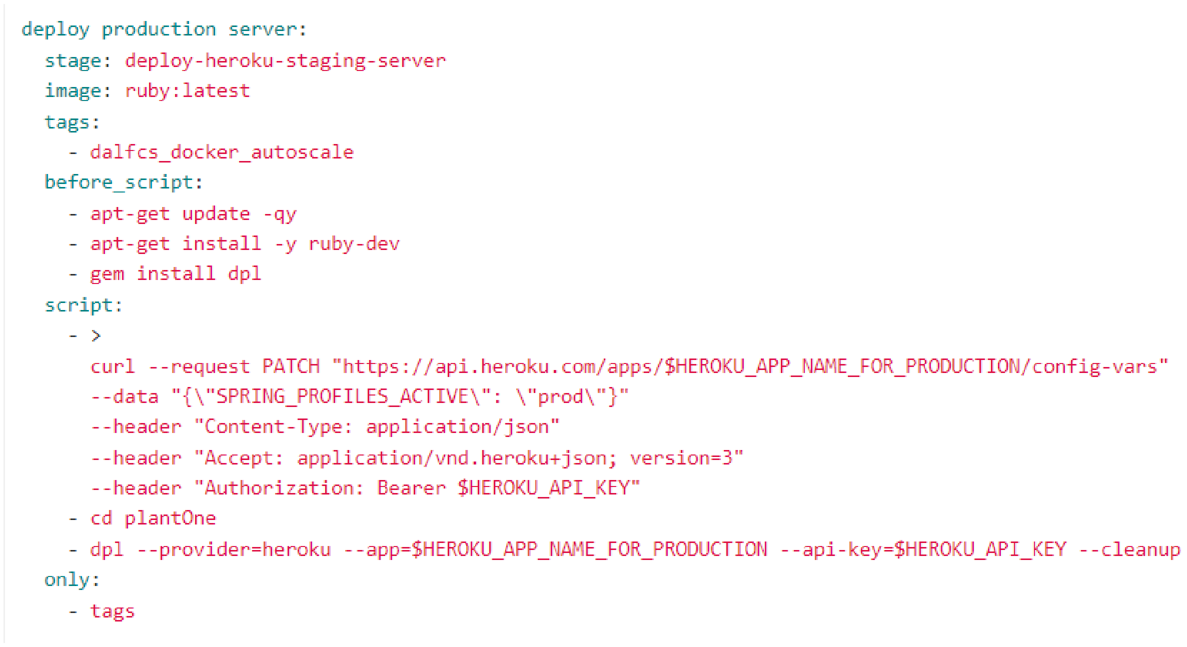
- When we make modifications to the main branch, the deploy testing server will be notified, and the deploy production server will be notified when we release tags.
- In both portions, we begin by installing the most recent Ruby image.
- The ruby image is then updated, and ruby-dev and dpl are installed.
- We install ruby-dev to execute dpl, a deploy tool designed for continuous deployment.
- In the following script step, we access Heroku's API and pass SPRING PROFILES ACTIVE variable value to Heroku.



```
deploy testing server:
  stage: deploy-heroku-staging-server
  image: ruby:latest
  tags:
    - dalfcs_docker_autoscale
  before_script:
    - apt-get update -qy
    - apt-get install -y ruby-dev
    - gem install dpl
  script:
    - >
      curl --request PATCH "https://api.heroku.com/apps/$HEROKU_APP_NAME_FOR_TEST/config-vars"
      --data "{\"SPRING_PROFILES_ACTIVE\": \"test\"}"
      --header "Content-Type: application/json"
      --header "Accept: application/vnd.heroku+json; version=3"
      --header "Authorization: Bearer $HEROKU_API_KEY"
    - cd plantOne
    - dpl --provider=heroku --app=$HEROKU_APP_NAME_FOR_TEST --api-key=$HEROKU_API_KEY --cleanup
  only:
    - main
```

Figure 8: Deploy testing backend

- Finally, we will deploy our application to Heroku. Here, we provide the name of our Heroku application and api key using GitLab's CI/CD global variables. These variables are HEROKU_APP_NAME_FOR_TEST, HEROKU_API_KEY and HEROKU_APP_UI_FOR_TEST.



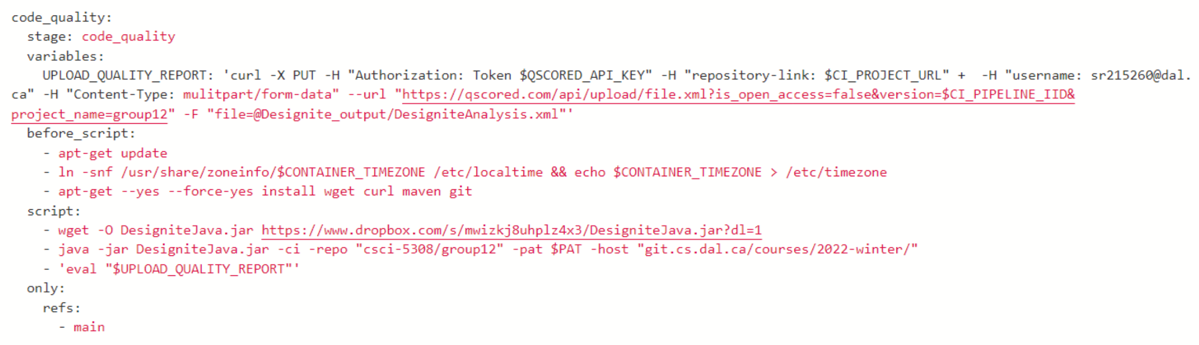
```yaml
deploy production server:
  stage: deploy-heroku-staging-server
  image: ruby:latest
  tags:
    - dalfcs_docker_autoscale
  before_script:
    - apt-get update -qy
    - apt-get install -y ruby-dev
    - gem install dpl
  script:
    - >
      curl --request PATCH "https://api.heroku.com/apps/$HEROKU_APP_NAME_FOR_PRODUCTION/config-vars"
      --data "{\"SPRING_PROFILES_ACTIVE\": \"prod\"}"
      --header "Content-Type: application/json"
      --header "Accept: application/vnd.heroku+json; version=3"
      --header "Authorization: Bearer $HEROKU_API_KEY"
    - cd plantOne
    - dpl --provider=heroku --app=$HEROKU_APP_NAME_FOR_PRODUCTION --api-key=$HEROKU_API_KEY --cleanup
  only:
    - tags
```

Figure 9: Deploy production backend

4) Using Designite to check code quality (code_quality)

Here we define code_quality job which will run only when we are doing any changes in main.

- Inside this job we mention one variable UPLOAD_QUALITY_REPORT which is calling API using curl where we are sending DesignteAnalysis to the Qscored website. This variable will call in further steps.
- In next step we are calling script where we are updating and installing wget, curl, maven and git.
- Wget is a free software package for retrieving files using HTTPS, which we are using to download DesigniteJava.jar from dropbox.
- Then running DesigniteJava.jar using java command and passing repo, personal access token and host.
- After running successfully, we are calling UPLOAD_QUALITY_REPORT and send report to QScored.



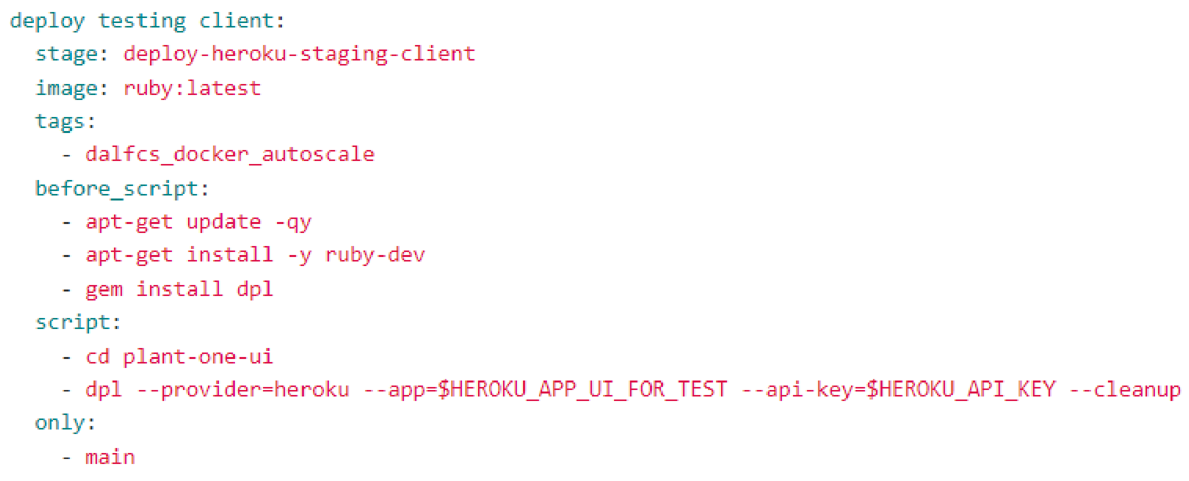
```
code_quality:
  stage: code_quality
  variables:
    UPLOAD_QUALITY_REPORT: 'curl -X PUT -H "Authorization: Token $QSCORED_API_KEY" -H "repository-link: $CI_PROJECT_URL" +  -H "username: sr215260@dal.
ca" -H "Content-Type: mulitpart/form-data" --url "https://qscored.com/api/upload/file.xml?is_open_access=false&version=$CI_PIPELINE_IID&
project_name=group12" -F "file=@Designite_output/DesigniteAnalysis.xml"'
  before_script:
    - apt-get update
    - ln -snf /usr/share/zoneinfo/$CONTAINER_TIMEZONE /etc/localtime && echo $CONTAINER_TIMEZONE > /etc/timezone
    - apt-get --yes --force-yes install wget curl maven git
  script:
    - wget -O DesigniteJava.jar https://www.dropbox.com/s/mwizkj8uhplz4x3/DesigniteJava.jar?dl=1
    - java -jar DesigniteJava.jar -ci -repo "csci-5308/group12" -pat $PAT -host "git.cs.dal.ca/courses/2022-winter/"
    - 'eval "$UPLOAD_QUALITY_REPORT"'
  only:
    refs:
      - main
```

Figure 10: Codequality

5) Deploying frontend application on Heroku server (deploy-heroku-staging-client)

Here we have two jobs: deploy testing client and deploy production client

- When we make modifications to the main branch, the deploy testing client will be notified, and the deploy production client will be notified when we release tags.
- First of all, we installed ruby latest image. Then update it using "apt-get update -qy". After we added ruby-dev and dpl for deployment.



```
deploy testing client:
  stage: deploy-heroku-staging-client
  image: ruby:latest
  tags:
    - dalfcs_docker_autoscale
  before_script:
    - apt-get update -qy
    - apt-get install -y ruby-dev
    - gem install dpl
  script:
    - cd plant-one-ui
    - dpl --provider=heroku --app=$HEROKU_APP_UI_FOR_TEST --api-key=$HEROKU_API_KEY --cleanup
  only:
    - main
```

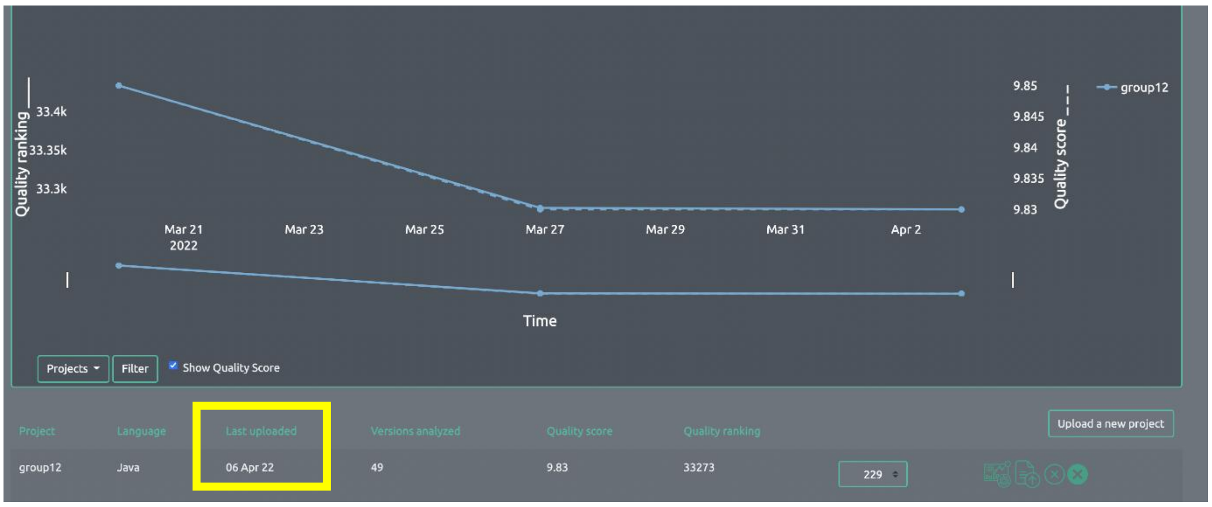
Figure 11: Deploy testing frontend

- Finally, we will deploy our application to Heroku. Here, we provide the name of our Heroku application and API key using GitLab's CI/CD global variables. These variables are HEROKU_APP_UI_FOR_TEST, HEROKU_API_KEY and HEROKU_APP_UI_FOR_PRODUCTION.

```
deploy production client:
  stage: deploy-heroku-staging-client
  image: ruby:latest
  tags:
    - dalfcs_docker_autoscale
  before_script:
    - apt-get update -qy
    - apt-get install -y ruby-dev
    - gem install dpl
  script:
    - cd plant-one-ui
    - dpl --provider=heroku --app=$HEROKU_APP_UI_FOR_PRODUCTION --api-key=$HEROKU_API_KEY --cleanup
  only:
    - tags
```

Figure 12: Deploy production frontend

6) QScored Quality ranking graph (Last update: 6th April)



Project	Language	Last uploaded	Versions analyzed	Quality score	Quality ranking	
group12	Java	06 Apr 22	49	9.83	33273	229

Figure 13: QScored Quality Ranking

References

[1] T. Sharma, "QScored", *Qscored.com*, 2022. [Online]. Available: https://qscored.com/. [Accessed: April 06, 2022].

[2] "Gitlab", *Gitlab* 2022. [Online]. Available: https://gitlab.com/users/sign_in. [Accessed: April 06, 2022].

[3] "Cloud Application Platform | Heroku", *Heroku.com*, 2022. [Online]. Available: https://www.heroku.com/. [Accessed: April 06, 2022].

[4] "Designite - Reduce Technical Debt of your Software", *Designite-tools.com*, 2022. [Online]. Available: https://www.designite-tools.com/. [Accessed: April 06, 2022].