6/20/2024

# CI/CD Implementation

Individual: FestivalConnect

Name: Lucas Jacobs
Class: S-A-RB06
PCN: 490692
Student number: 4607368
Technical teachers: Felipe Ebert, Bartosz Paszkowski
Semester coach: Gerard Elbers

# Table of Contents

# Flow CI/CD

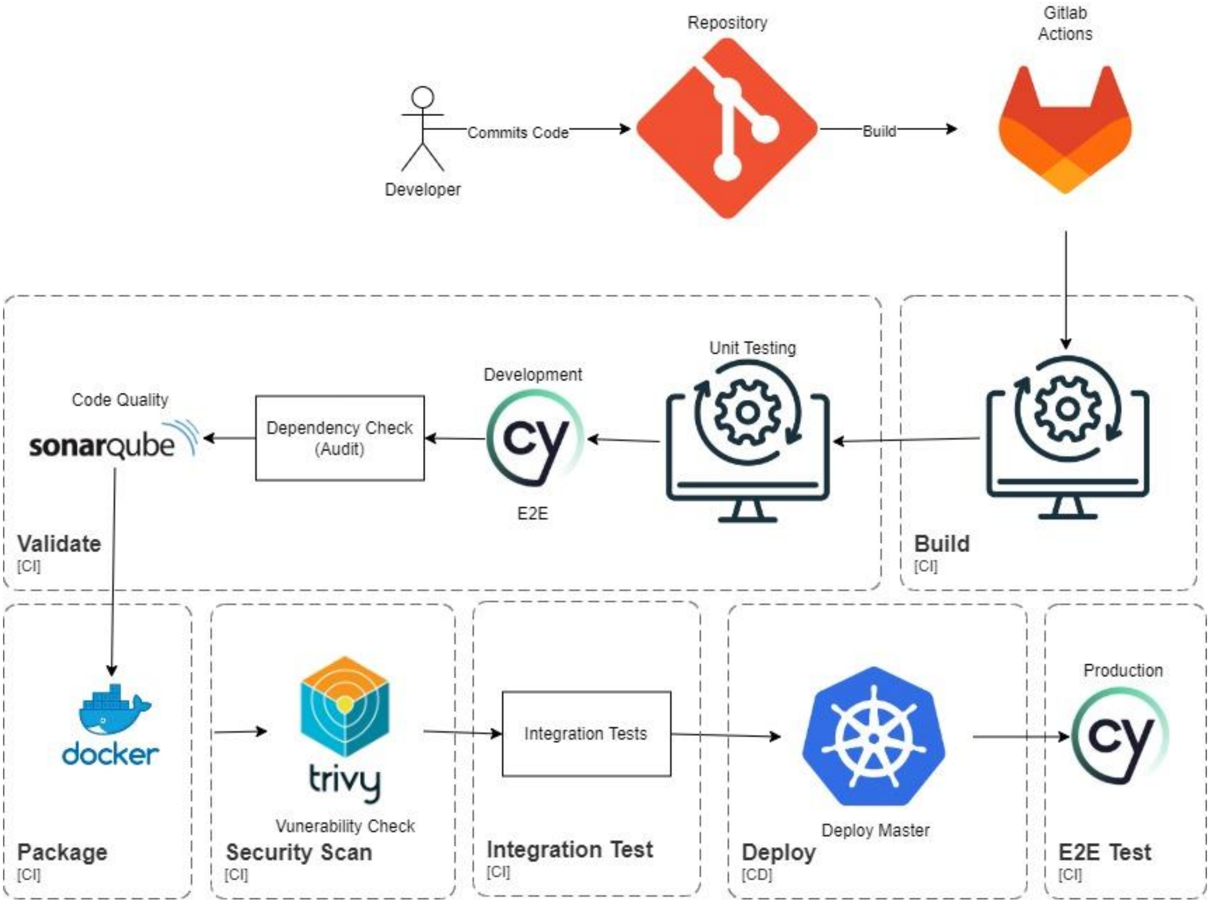


*Figure 1: CI/CD Flow*

The following stages are implemented: Build, Test, Analyze (Code quality plus Test coverage), Security, Package, and Deploy.

I am using the GitLab environment of school, with a self-running runner.

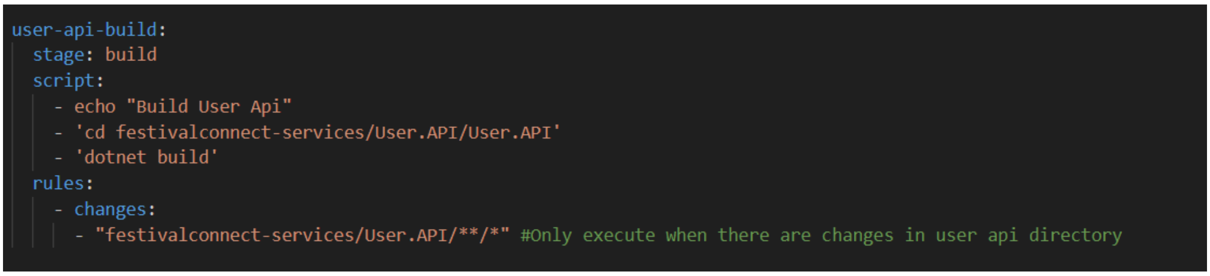
I will go over each result and how it is implemented in my project, I will use the user service for a service example and the front end.

# Implementation

The stages of services and front end are only getting triggered when there are changes to this service. This will make sure that not all the services will be triggered when committing changes to only one service.

## Build

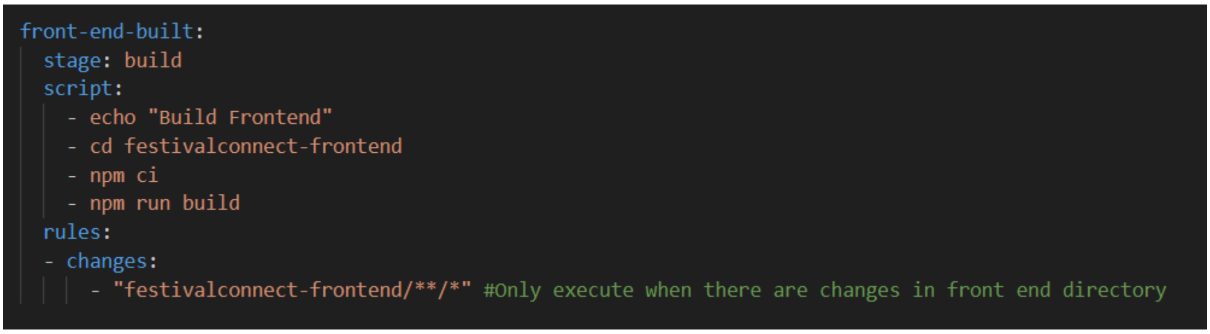
User Service



```
user-api-build:
  stage: build
  script:
    - echo "Build User Api"
    - 'cd festivalconnect-services/User.API/User.API'
    - 'dotnet build'
  rules:
    - changes:
      - "festivalconnect-services/User.API/**/*" #Only execute when there are changes in user api directory
```

*Figure 2: Build User Service*

Front End

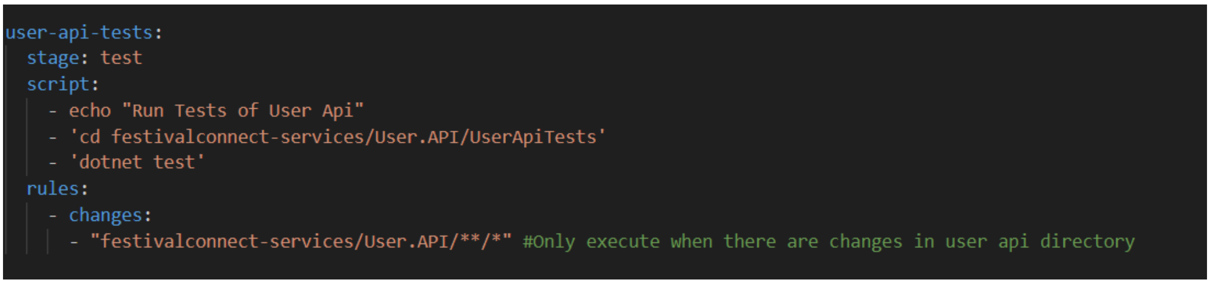


```
front-end-built:
  stage: build
  script:
    - echo "Build Frontend"
    - cd festivalconnect-frontend
    - npm ci
    - npm run build
  rules:
  - changes:
    - "festivalconnect-frontend/**/*" #Only execute when there are changes in front end directory
```

*Figure 3: Build Front End*

# Tests

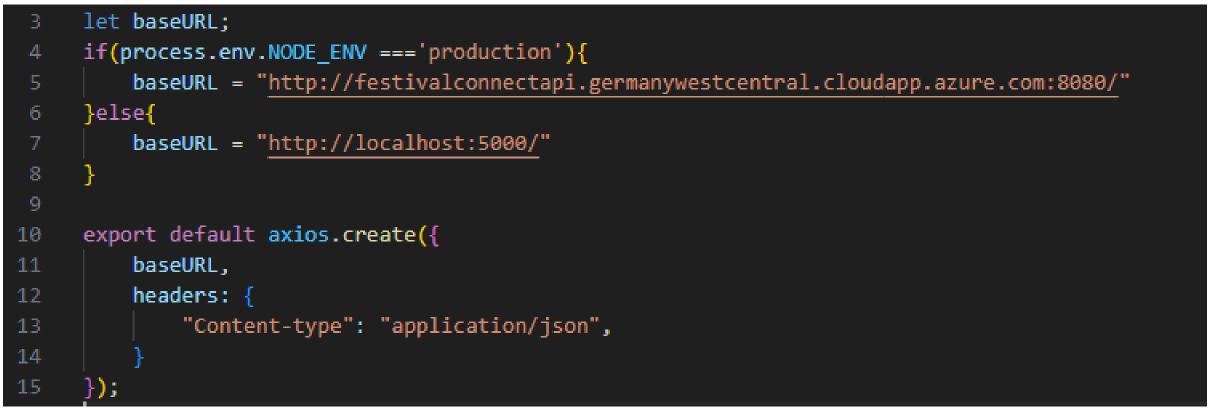
After the build, I will perform the unit tests, which are the fastest to execute of the tests.



```yaml
user-api-tests:
  stage: test
  script:
    - echo "Run Tests of User Api"
    - 'cd festivalconnect-services/User.API/UserApiTests'
    - 'dotnet test'
  rules:
    - changes:
      - "festivalconnect-services/User.API/**/*" #Only execute when there are changes in user api directory
```

*Figure 4: User Service Test Pipeline*

Then, when this is finished, the E2E tests will be performed, for the browsers Firefox and Chrome. I both have E2E tests for development and production. To allocate in which environment we are, the serverbase will get the base URL based on the environment variable.

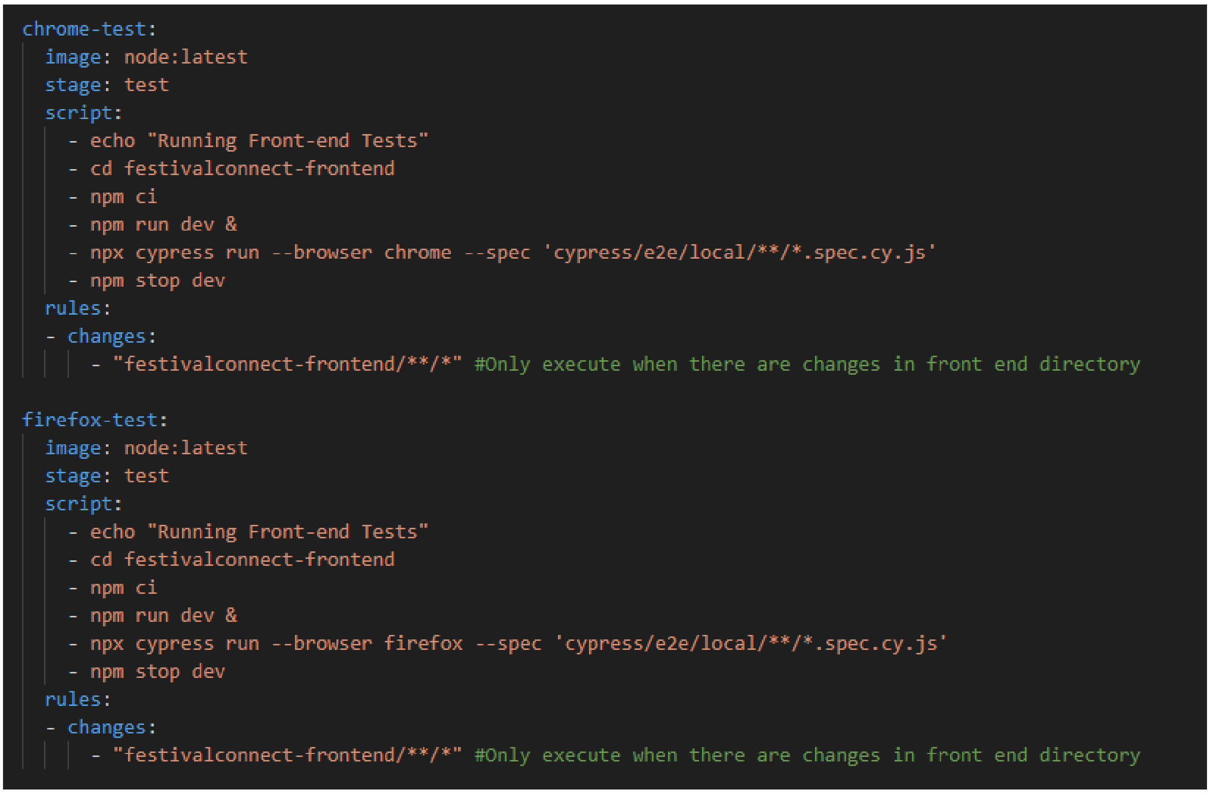


```javascript
let baseURL;
if(process.env.NODE_ENV ==='production'){
    baseURL = "http://festivalconnectapi.germanywestcentral.cloudapp.azure.com:8080/"
}else{
    baseURL = "http://localhost:5000/"
}

export default axios.create({
    baseURL,
    headers: {
        "Content-type": "application/json",
    }
});
```

*Figure 5: Base URL Code Based on Stage*

The following tests are in development, E2E.

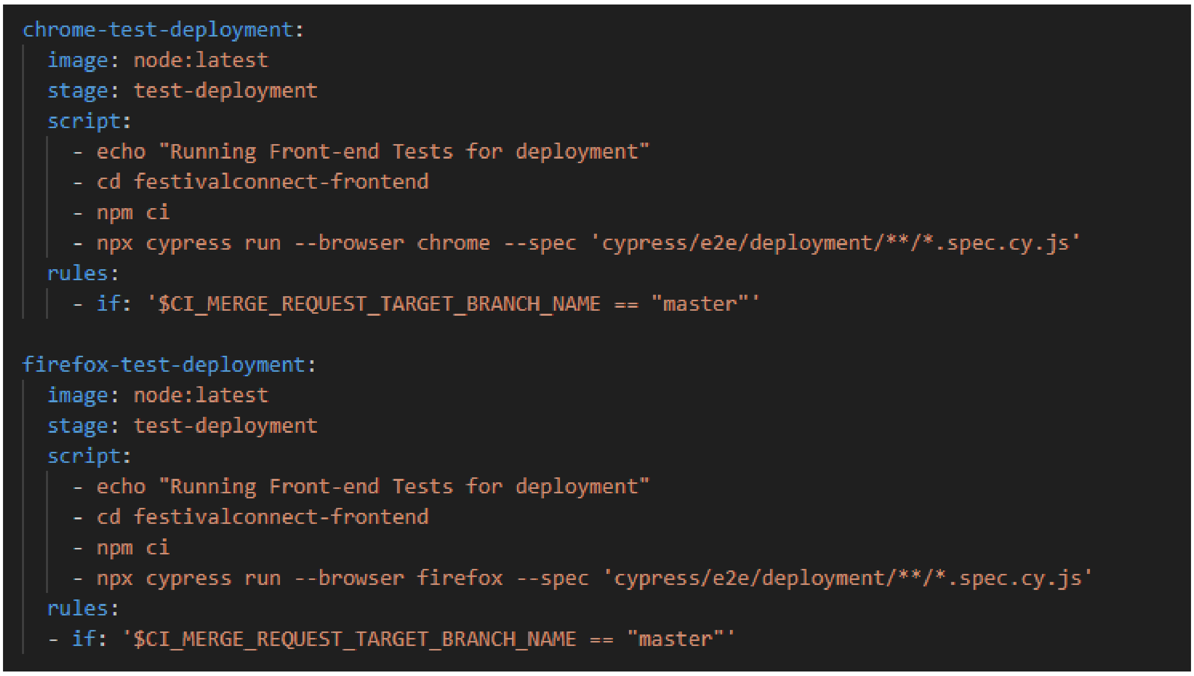


```
chrome-test:
  image: node:latest
  stage: test
  script:
    - echo "Running Front-end Tests"
    - cd festivalconnect-frontend
    - npm ci
    - npm run dev &
    - npx cypress run --browser chrome --spec 'cypress/e2e/local/**/*.spec.cy.js'
    - npm stop dev
  rules:
  - changes:
      - "festivalconnect-frontend/**/*" #Only execute when there are changes in front end directory

firefox-test:
  image: node:latest
  stage: test
  script:
    - echo "Running Front-end Tests"
    - cd festivalconnect-frontend
    - npm ci
    - npm run dev &
    - npx cypress run --browser firefox --spec 'cypress/e2e/local/**/*.spec.cy.js'
    - npm stop dev
  rules:
  - changes:
      - "festivalconnect-frontend/**/*" #Only execute when there are changes in front end directory
```

*Figure 6: Development E2E tests Pipeline*

Furthermore, I made E2E tests for also the production environment, this is specified as follows.



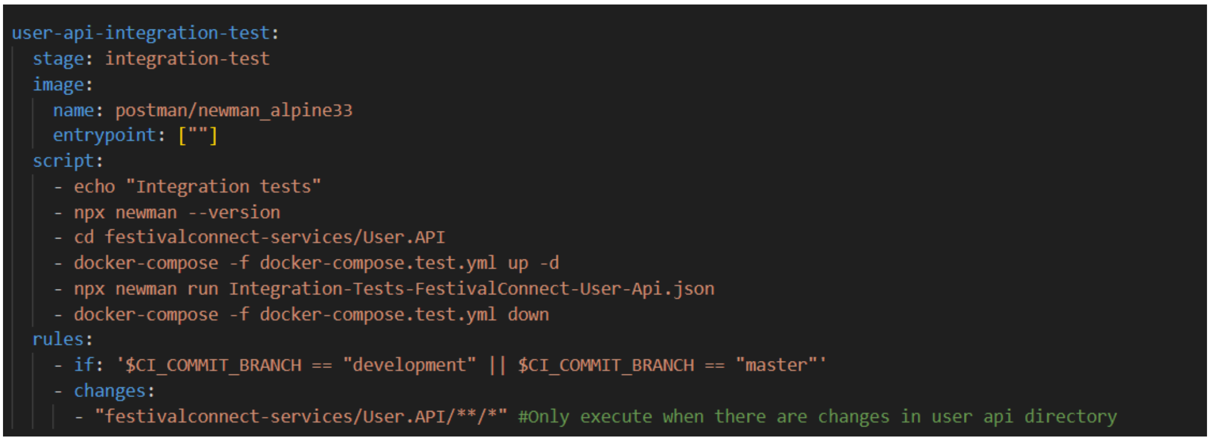
```
chrome-test-deployment:
  image: node:latest
  stage: test-deployment
  script:
    - echo "Running Front-end Tests for deployment"
    - cd festivalconnect-frontend
    - npm ci
    - npx cypress run --browser chrome --spec 'cypress/e2e/deployment/**/*.spec.cy.js'
  rules:
    - if: '$CI_MERGE_REQUEST_TARGET_BRANCH_NAME == "master"'

firefox-test-deployment:
  image: node:latest
  stage: test-deployment
  script:
    - echo "Running Front-end Tests for deployment"
    - cd festivalconnect-frontend
    - npm ci
    - npx cypress run --browser firefox --spec 'cypress/e2e/deployment/**/*.spec.cy.js'
  rules:
  - if: '$CI_MERGE_REQUEST_TARGET_BRANCH_NAME == "master"'
```

*Figure 7: Production E2E Tests Pipeline*

Note that the production E2E tests only be executed when merged to master, since deployment also only will be deployed when merging to master. Also they will be executed after the deployment happens.

The last tests are performed after the publishing of the images to Docker Hub, to test if the services are working together as expected.



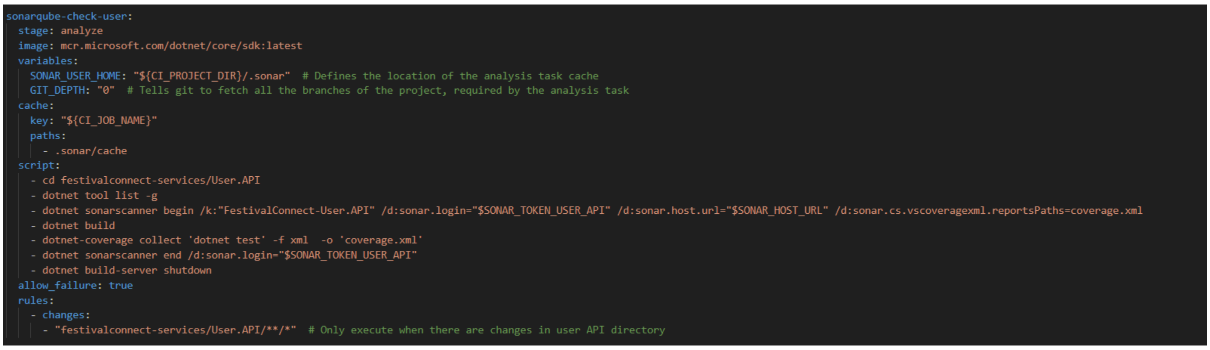
```yaml
user-api-integration-test:
  stage: integration-test
  image:
    name: postman/newman_alpine33
    entrypoint: [""]
  script:
    - echo "Integration tests"
    - npx newman --version
    - cd festivalconnect-services/User.API
    - docker-compose -f docker-compose.test.yml up -d
    - npx newman run Integration-Tests-FestivalConnect-User-Api.json
    - docker-compose -f docker-compose.test.yml down
  rules:
    - if: '$CI_COMMIT_BRANCH == "development" || $CI_COMMIT_BRANCH == "master"'
    - changes:
      - "festivalconnect-services/User.API/**/*" #Only execute when there are changes in user api directory
```

*Figure 8: Integration Tests Pipeline*

The integration tests are made in Postman and exported as JSON files. It uses Newman to perform the tests, which allows it to execute a collection of tests, like in postman.

# Analyze

For analyzing I am using SonarQube.



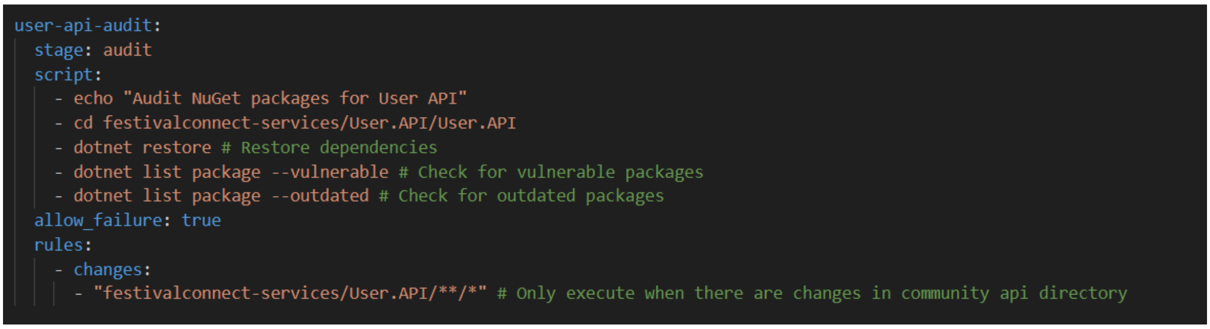
```yaml
sonarqube-check-user:
  stage: analyze
  image: mcr.microsoft.com/dotnet/core/sdk:latest
  variables:
    SONAR_USER_HOME: "${CI_PROJECT_DIR}/.sonar"  # Defines the location of the analysis task cache
    GIT_DEPTH: "0"  # Tells git to fetch all the branches of the project, required by the analysis task
  cache:
    key: "${CI_JOB_NAME}"
    paths:
      - .sonar/cache
  script:
    - cd festivalconnect-services/User.API
    - dotnet tool list -g
    - dotnet sonarscanner begin /k:"FestivalConnect-User.API" /d:sonar.login="$SONAR_TOKEN_USER_API" /d:sonar.host.url="$SONAR_HOST_URL" /d:sonar.cs.vscoveragexml.reportsPaths=coverage.xml
    - dotnet build
    - dotnet-coverage collect 'dotnet test' -f xml  -o 'coverage.xml'
    - dotnet sonarscanner end /d:sonar.login="$SONAR_TOKEN_USER_API"
    - dotnet build-server shutdown
  allow_failure: true
  rules:
    - changes:
      - "festivalconnect-services/User.API/**/*"  # Only execute when there are changes in user API directory
```

*Figure 9: SonarQube Pipeline*

For collecting information about the test coverage, I use coverlet, which can make a coverage file created in XML format that SonarQube can analyze.

# Security

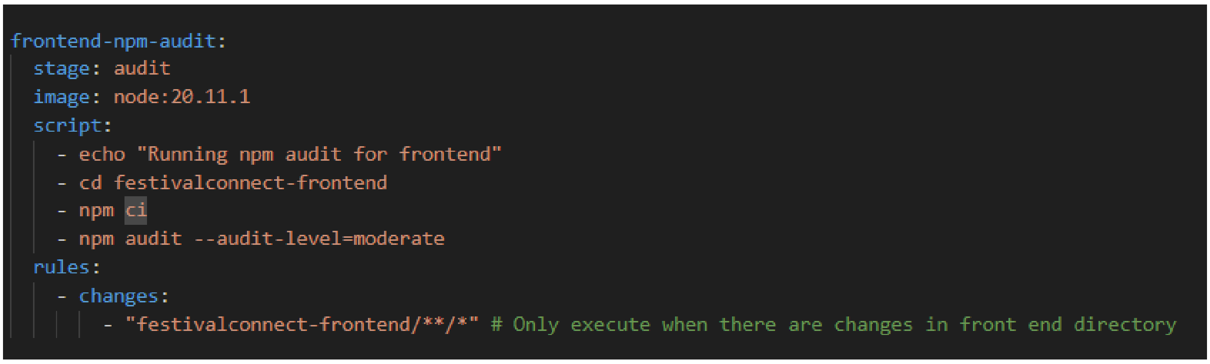
After testing, I will do an audit check, to see if the packages that I am using are vulnerable and outdated.



```yaml
user-api-audit:
  stage: audit
  script:
    - echo "Audit NuGet packages for User API"
    - cd festivalconnect-services/User.API/User.API
    - dotnet restore # Restore dependencies
    - dotnet list package --vulnerable # Check for vulnerable packages
    - dotnet list package --outdated # Check for outdated packages
  allow_failure: true
  rules:
    - changes:
      - "festivalconnect-services/User.API/**/*" # Only execute when there are changes in community api directory
```

*Figure 10: Dependency Check Service Pipeline*

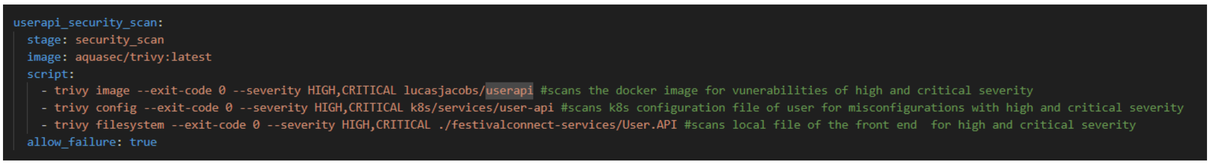
For the front-end



```yaml
frontend-npm-audit:
  stage: audit
  image: node:20.11.1
  script:
    - echo "Running npm audit for frontend"
    - cd festivalconnect-frontend
    - npm ci
    - npm audit --audit-level=moderate
  rules:
    - changes:
        - "festivalconnect-frontend/**/*" # Only execute when there are changes in front end directory
```

*Figure 11: Dependency Check Front End Pipeline*

When the publishing to docker is done, I will use Trivy to check for vulnerabilities in my code in the services, Kubernetes files, and the images uploaded to docker.

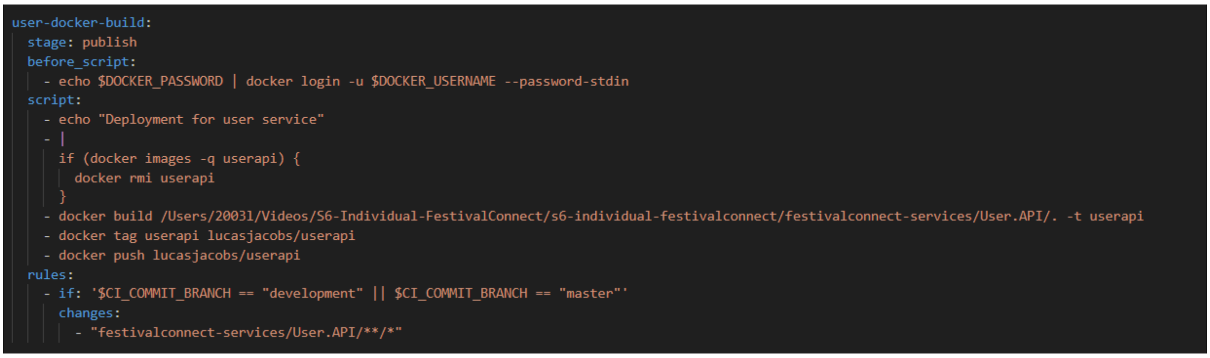


```yaml
userapi_security_scan:
  stage: security_scan
  image: aquasec/trivy:latest
  script:
    - trivy image --exit-code 0 --severity HIGH,CRITICAL lucasjacobs/userapi #scans the docker image for vunerabilities of high and critical severity
    - trivy config --exit-code 0 --severity HIGH,CRITICAL k8s/services/user-api #scans k8s configuration file of user for misconfigurations with high and critical severity
    - trivy filesystem --exit-code 0 --severity HIGH,CRITICAL ./festivalconnect-services/User.API #scans local file of the front end  for high and critical severity
  allow_failure: true
```

*Figure 12: Vulnerability Check Pipeline*

## Package

I will package my services using docker.



```
user-docker-build:
  stage: publish
  before_script:
    - echo $DOCKER_PASSWORD | docker login -u $DOCKER_USERNAME --password-stdin
  script:
    - echo "Deployment for user service"
    - |
      if (docker images -q userapi) {
        docker rmi userapi
      }
    - docker build /Users/20031/Videos/S6-Individual-FestivalConnect/s6-individual-festivalconnect/festivalconnect-services/User.API/. -t userapi
    - docker tag userapi lucasjacobs/userapi
    - docker push lucasjacobs/userapi
  rules:
    - if: '$CI_COMMIT_BRANCH == "development" || $CI_COMMIT_BRANCH == "master"'
      changes:
        - "festivalconnect-services/User.API/**/*"
```

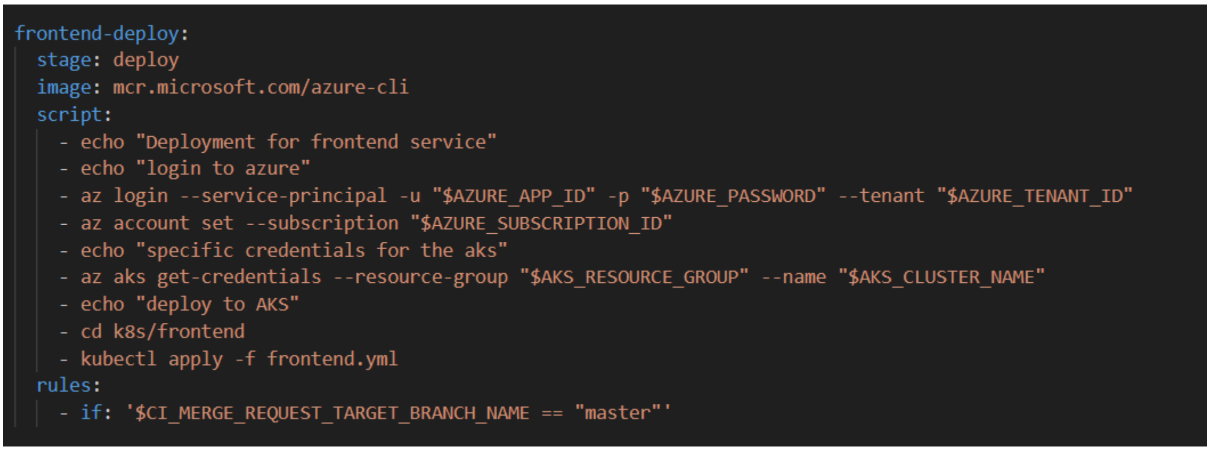
*Figure 13: Package Docker Pipeline*

First check if the image exists, when it does this one will be removed. Then it will create a new one, give it a tag, and finally publish it to Docker Hub.

# Deploy

I will use Kubernetes to deploy my services to, this is only done when merging to the master branch, to make sure a stable version is deployed.

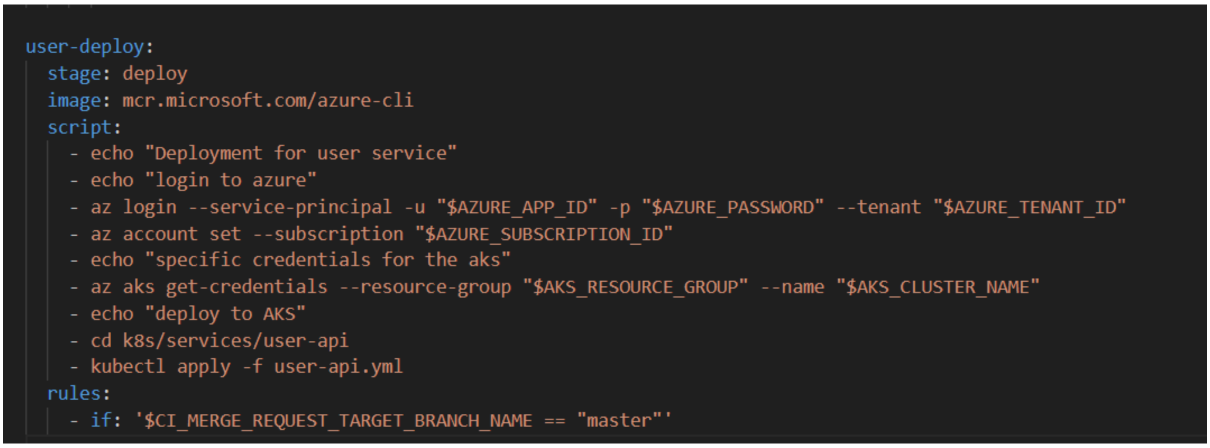
Front end:



```yaml
frontend-deploy:
  stage: deploy
  image: mcr.microsoft.com/azure-cli
  script:
    - echo "Deployment for frontend service"
    - echo "login to azure"
    - az login --service-principal -u "$AZURE_APP_ID" -p "$AZURE_PASSWORD" --tenant "$AZURE_TENANT_ID"
    - az account set --subscription "$AZURE_SUBSCRIPTION_ID"
    - echo "specific credentials for the aks"
    - az aks get-credentials --resource-group "$AKS_RESOURCE_GROUP" --name "$AKS_CLUSTER_NAME"
    - echo "deploy to AKS"
    - cd k8s/frontend
    - kubectl apply -f frontend.yml
  rules:
    - if: '$CI_MERGE_REQUEST_TARGET_BRANCH_NAME == "master"'
```

*Figure 14: Deployment Front End Pipeline*

User Service:



```yaml
user-deploy:
  stage: deploy
  image: mcr.microsoft.com/azure-cli
  script:
    - echo "Deployment for user service"
    - echo "login to azure"
    - az login --service-principal -u "$AZURE_APP_ID" -p "$AZURE_PASSWORD" --tenant "$AZURE_TENANT_ID"
    - az account set --subscription "$AZURE_SUBSCRIPTION_ID"
    - echo "specific credentials for the aks"
    - az aks get-credentials --resource-group "$AKS_RESOURCE_GROUP" --name "$AKS_CLUSTER_NAME"
    - echo "deploy to AKS"
    - cd k8s/services/user-api
    - kubectl apply -f user-api.yml
  rules:
    - if: '$CI_MERGE_REQUEST_TARGET_BRANCH_NAME == "master"'
```

*Figure 15: Deployment User Service Pipeline*

# Results

The following is the result of the pipeline.

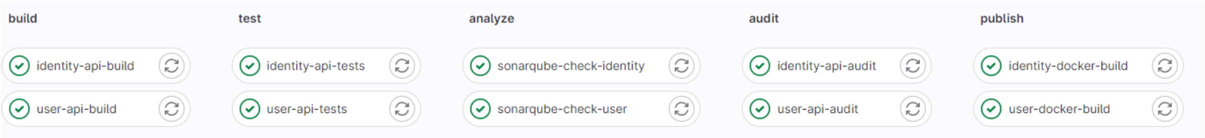


*Figure 16: Results build, test, analyze, audit, publish*

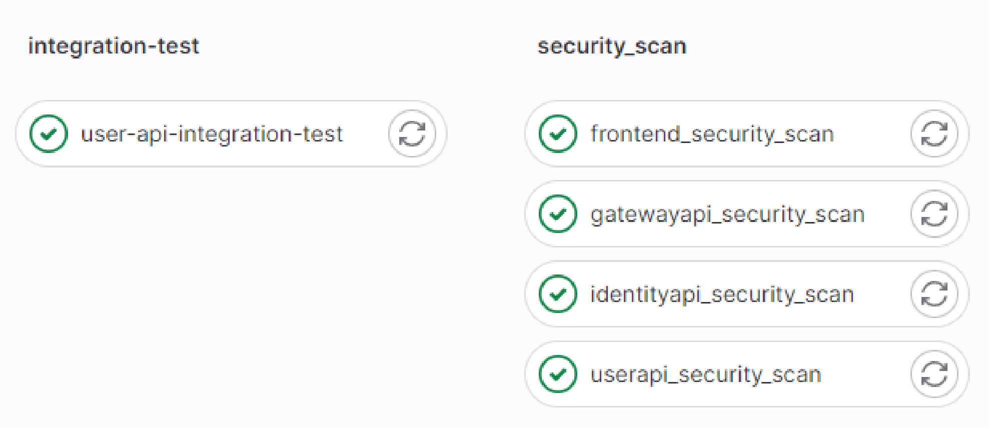


*Figure 17: Results Integration tests, security scan*

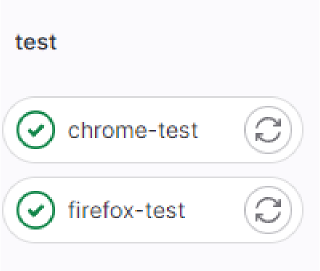
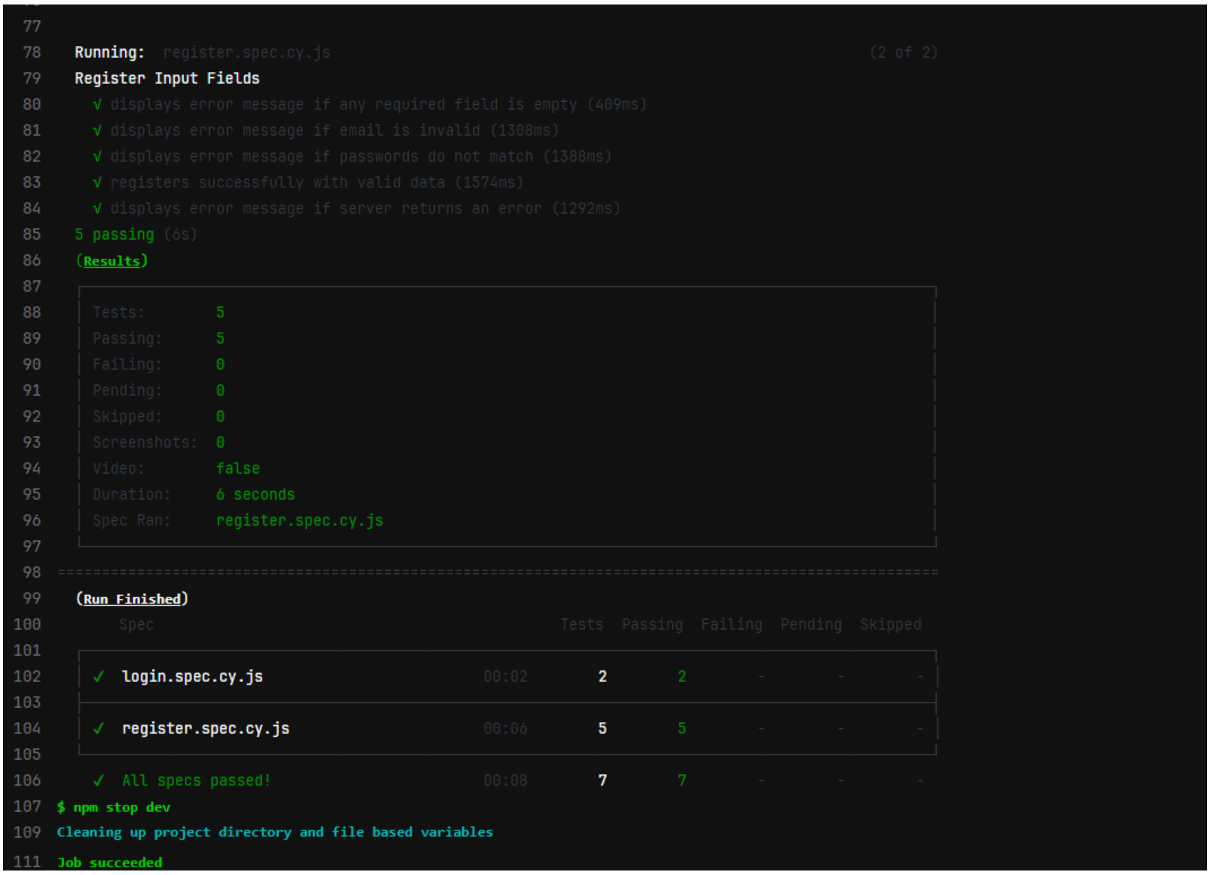


*Figure 18: Results E2E Development*

For E2E development tests the following results are displayed.

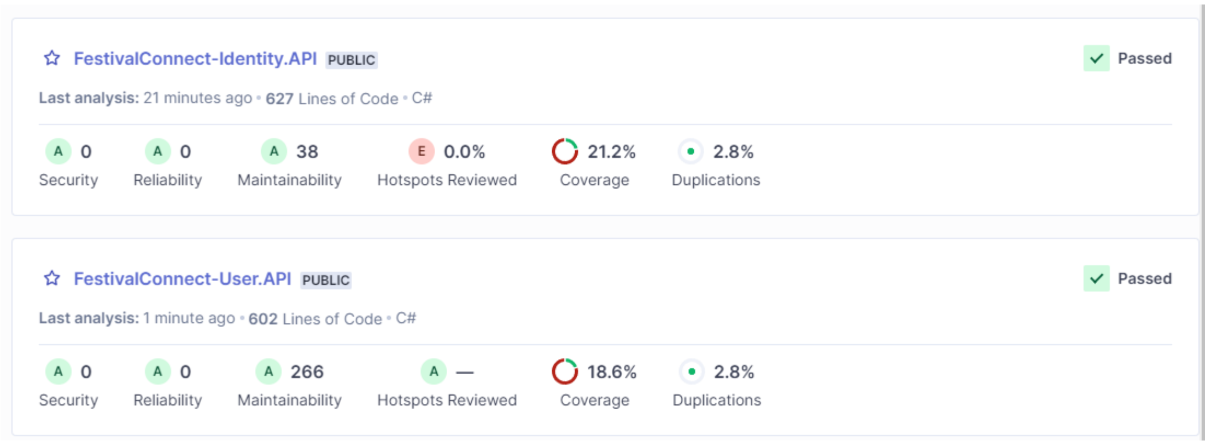


```

  Running:  register.spec.cy.js                                                   (2 of 2)
  Register Input Fields
    √ displays error message if any required field is empty (409ms)
    √ displays error message if email is invalid (1308ms)
    √ displays error message if passwords do not match (1388ms)
    √ registers successfully with valid data (1574ms)
    √ displays error message if server returns an error (1292ms)
  5 passing (6s)
  (Results)
  ┌────────────────────────────────────────────────────────────────────────────────────────────────┐
  │ Tests:        5                                                                                │
  │ Passing:      5                                                                                │
  │ Failing:      0                                                                                │
  │ Pending:      0                                                                                │
  │ Skipped:      0                                                                                │
  │ Screenshots:  0                                                                                │
  │ Video:        false                                                                            │
  │ Duration:     6 seconds                                                                        │
  │ Spec Ran:     register.spec.cy.js                                                              │
  └────────────────────────────────────────────────────────────────────────────────────────────────┘
====================================================================================================
  (Run Finished)
       Spec                                              Tests  Passing  Failing  Pending  Skipped
  ┌────────────────────────────────────────────────────────────────────────────────────────────────┐
  │ ✓  login.spec.cy.js                         00:02        2        2        -        -        - │
  ├────────────────────────────────────────────────────────────────────────────────────────────────┤
  │ ✓  register.spec.cy.js                      00:06        5        5        -        -        - │
  └────────────────────────────────────────────────────────────────────────────────────────────────┘
    ✓  All specs passed!                        00:08        7        7        -        -        -
$ npm stop dev
Cleaning up project directory and file based variables
Job succeeded
```

*Figure 19: Results of development E2E tests*

For SonarQube the following will be shown when SonarQube is performed in the pipeline.



☆ FestivalConnect-Identity.API PUBLIC ✓ Passed

Last analysis: 21 minutes ago • 627 Lines of Code • C#

A 0 Security · A 0 Reliability · A 38 Maintainability · E 0.0% Hotspots Reviewed · 21.2% Coverage · 2.8% Duplications

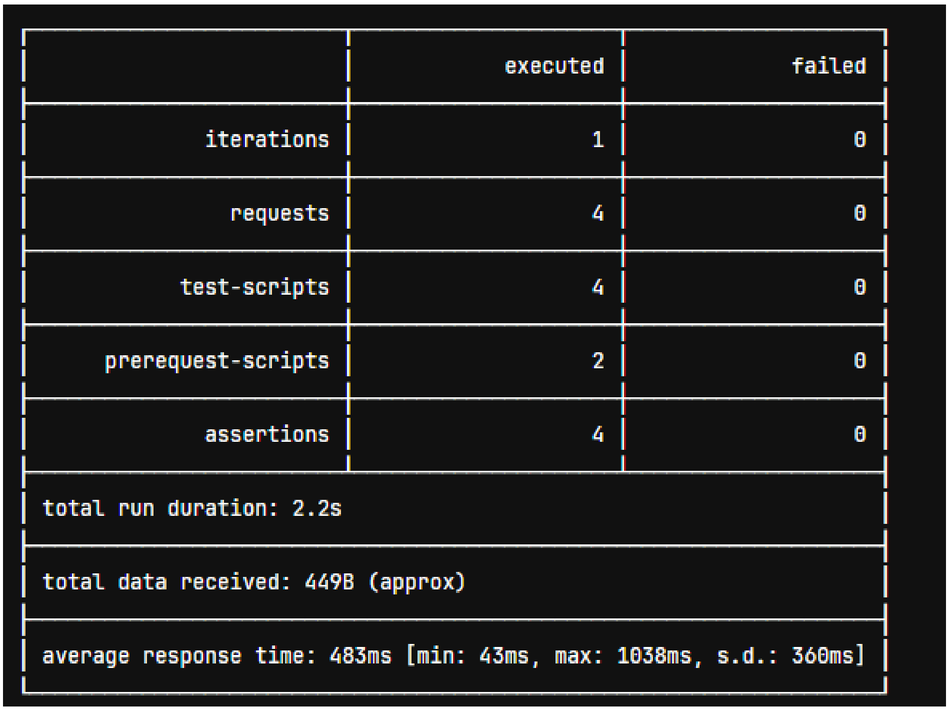
☆ FestivalConnect-User.API PUBLIC ✓ Passed

Last analysis: 1 minute ago • 602 Lines of Code • C#

A 0 Security · A 0 Reliability · A 266 Maintainability · A — Hotspots Reviewed · 18.6% Coverage · 2.8% Duplications

*Figure 20: Analyze Results (Code quality plus Code coverage)*

For the integration tests the following is displayed:



```
┌─────────────────────────┬──────────────────┬─────────────────┐
│                         │         executed │          failed │
├─────────────────────────┼──────────────────┼─────────────────┤
│              iterations │                1 │               0 │
├─────────────────────────┼──────────────────┼─────────────────┤
│                requests │                4 │               0 │
├─────────────────────────┼──────────────────┼─────────────────┤
│            test-scripts │                4 │               0 │
├─────────────────────────┼──────────────────┼─────────────────┤
│      prerequest-scripts │                2 │               0 │
├─────────────────────────┼──────────────────┼─────────────────┤
│              assertions │                4 │               0 │
├─────────────────────────┴──────────────────┴─────────────────┤
│ total run duration: 2.2s                                     │
├──────────────────────────────────────────────────────────────┤
│ total data received: 449B (approx)                           │
├──────────────────────────────────────────────────────────────┤
│ average response time: 483ms [min: 43ms, max: 1038ms, s.d.: 360ms] │
└──────────────────────────────────────────────────────────────┘
```

*Figure 21: Integration Test Results*

For Deployment:

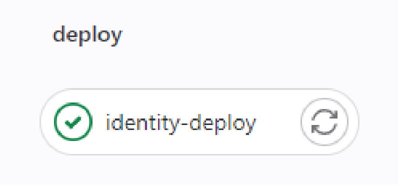
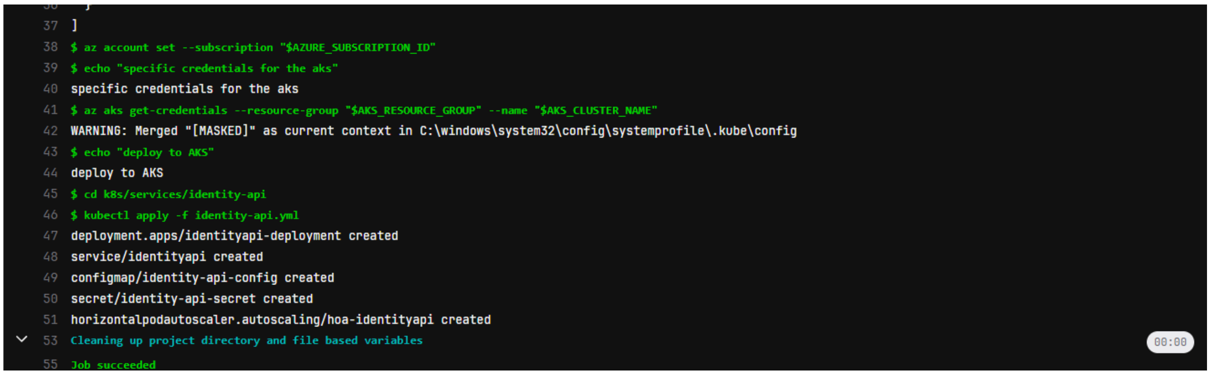


*Figure 22: Deployment result*



```
]
$ az account set --subscription "$AZURE_SUBSCRIPTION_ID"
$ echo "specific credentials for the aks"
specific credentials for the aks
$ az aks get-credentials --resource-group "$AKS_RESOURCE_GROUP" --name "$AKS_CLUSTER_NAME"
WARNING: Merged "[MASKED]" as current context in C:\windows\system32\config\systemprofile\.kube\config
$ echo "deploy to AKS"
deploy to AKS
$ cd k8s/services/identity-api
$ kubectl apply -f identity-api.yml
deployment.apps/identityapi-deployment created
service/identityapi created
configmap/identity-api-config created
secret/identity-api-secret created
horizontalpodautoscaler.autoscaling/hoa-identityapi created
Cleaning up project directory and file based variables
Job succeeded
```

*Figure 23: Deployment Result Specifics*

Finally, for the E2E tests in production, the following results are displayed.

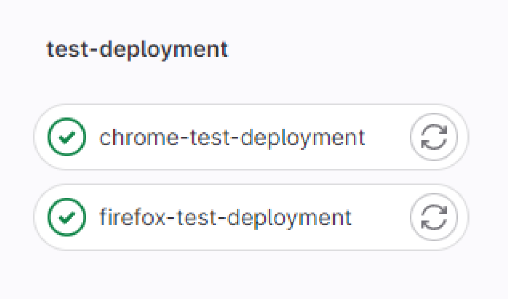
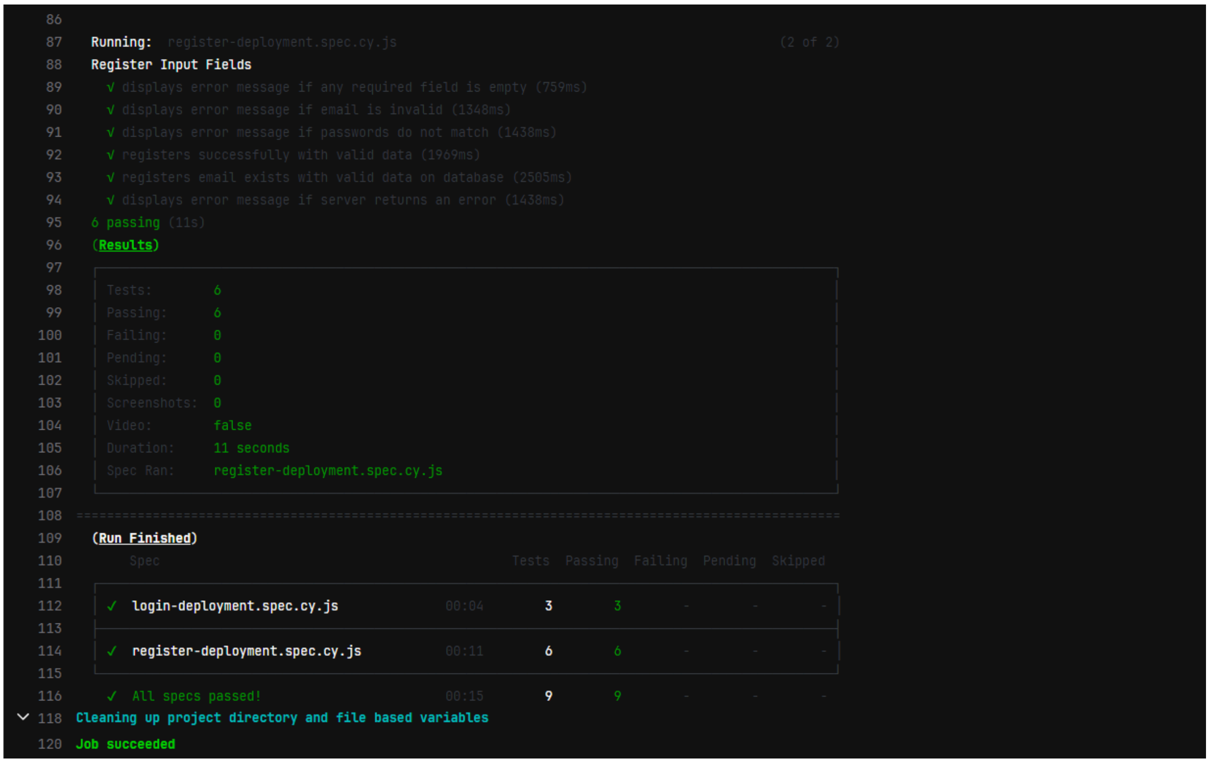


*Figure 24: E2E tests results deployment*

Chrome has the following results.



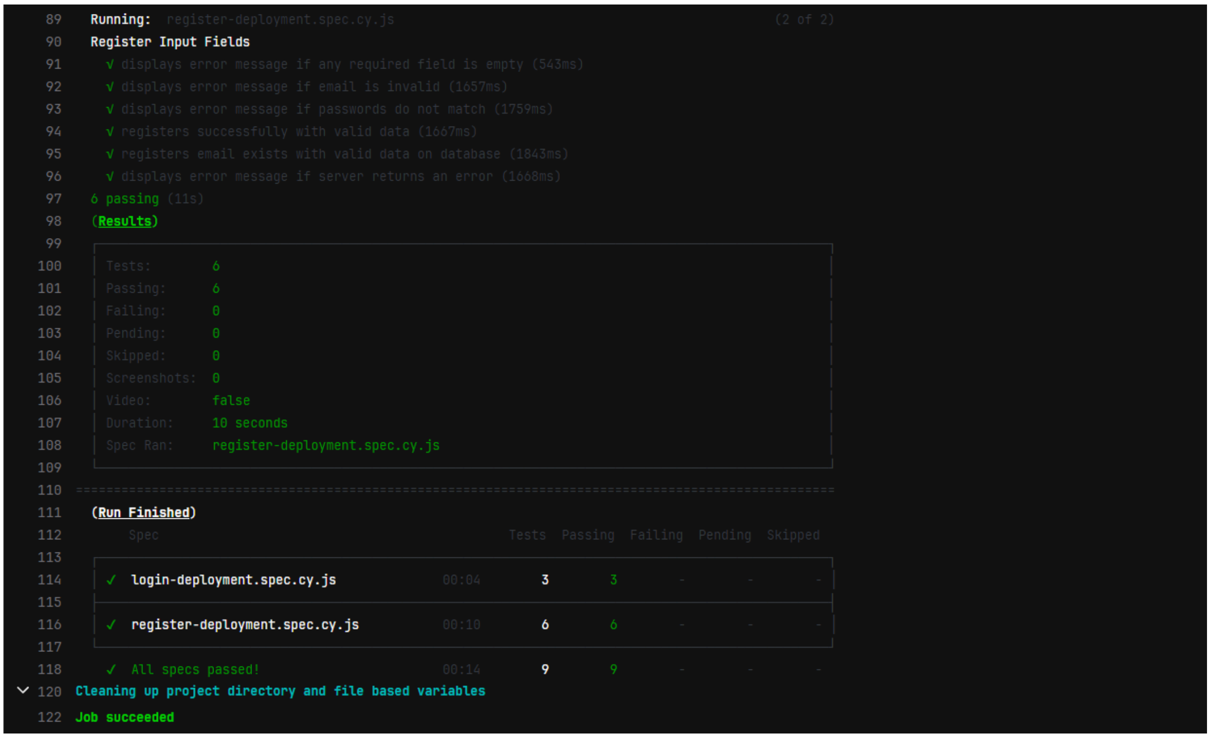
```

Running:  register-deployment.spec.cy.js                                                 (2 of 2)
Register Input Fields
  √ displays error message if any required field is empty (759ms)
  √ displays error message if email is invalid (1348ms)
  √ displays error message if passwords do not match (1438ms)
  √ registers successfully with valid data (1969ms)
  √ registers email exists with valid data on database (2505ms)
  √ displays error message if server returns an error (1438ms)
6 passing (11s)
(Results)

 ┌────────────────────────────────────────────────────────────────────────────────────────────┐
 │ Tests:        6                                                                            │
 │ Passing:      6                                                                            │
 │ Failing:      0                                                                            │
 │ Pending:      0                                                                            │
 │ Skipped:      0                                                                            │
 │ Screenshots:  0                                                                            │
 │ Video:        false                                                                        │
 │ Duration:     11 seconds                                                                   │
 │ Spec Ran:     register-deployment.spec.cy.js                                               │
 └────────────────────────────────────────────────────────────────────────────────────────────┘
====================================================================================================
(Run Finished)
     Spec                                              Tests  Passing  Failing  Pending  Skipped
 ┌────────────────────────────────────────────────────────────────────────────────────────────┐
 │ ✓  login-deployment.spec.cy.js              00:04        3        3        -        -        - │
 ├────────────────────────────────────────────────────────────────────────────────────────────┤
 │ ✓  register-deployment.spec.cy.js           00:11        6        6        -        -        - │
 └────────────────────────────────────────────────────────────────────────────────────────────┘
   ✓  All specs passed!                        00:15        9        9        -        -        -
Cleaning up project directory and file based variables
Job succeeded
```

*Figure 25: Deployment E2E Chrome Results*

And Firefox test results.



```
  Running:  register-deployment.spec.cy.js                                        (2 of 2)
  Register Input Fields
    √ displays error message if any required field is empty (543ms)
    √ displays error message if email is invalid (1657ms)
    √ displays error message if passwords do not match (1759ms)
    √ registers successfully with valid data (1667ms)
    √ registers email exists with valid data on database (1843ms)
    √ displays error message if server returns an error (1668ms)
  6 passing (11s)
  (Results)

  ┌────────────────────────────────────────────────────────────────────────────────────────────────┐
  │ Tests:        6                                                                                │
  │ Passing:      6                                                                                │
  │ Failing:      0                                                                                │
  │ Pending:      0                                                                                │
  │ Skipped:      0                                                                                │
  │ Screenshots:  0                                                                                │
  │ Video:        false                                                                            │
  │ Duration:     10 seconds                                                                       │
  │ Spec Ran:     register-deployment.spec.cy.js                                                   │
  └────────────────────────────────────────────────────────────────────────────────────────────────┘
====================================================================================================
  (Run Finished)
       Spec                                              Tests  Passing  Failing  Pending  Skipped

  ┌────────────────────────────────────────────────────────────────────────────────────────────────┐
  │ ✓  login-deployment.spec.cy.js              00:04        3        3        -        -        - │
  ├────────────────────────────────────────────────────────────────────────────────────────────────┤
  │ ✓  register-deployment.spec.cy.js           00:10        6        6        -        -        - │
  └────────────────────────────────────────────────────────────────────────────────────────────────┘
    ✓  All specs passed!                        00:14        9        9        -        -        -
Cleaning up project directory and file based variables
Job succeeded
```

*Figure 26: Deployment E2E Firefox Results*

# Conclusion

By implementing all these stages inside the pipeline we will bring efficiency, reliability, scalability, maintenance, and security throughout the software development lifecycle of FestivalConnect.

- **Efficiency**: With the automation of the different stages, we will improve the development process, and reduce the errors of the developer.
- **Reliability**: With a standard process and the automation of it, there is consistency in development, testing, and deploying. Also, the different tests, validate that the functionalities are working and behaving as expected.
- **Scalability**: By automating and standardizing the stages, we will allow for rapid iteration of tasks to perform.
- **Maintenance**: with automated code quality checks, we can improve and address problems and reduce the efforts of maintaining in the future.
- **Security**: With the different checks such as tests, audit checks, and file checks it will reduce the risks and potential attacks.