Configuración del pipeline y su vínculo con el tablero

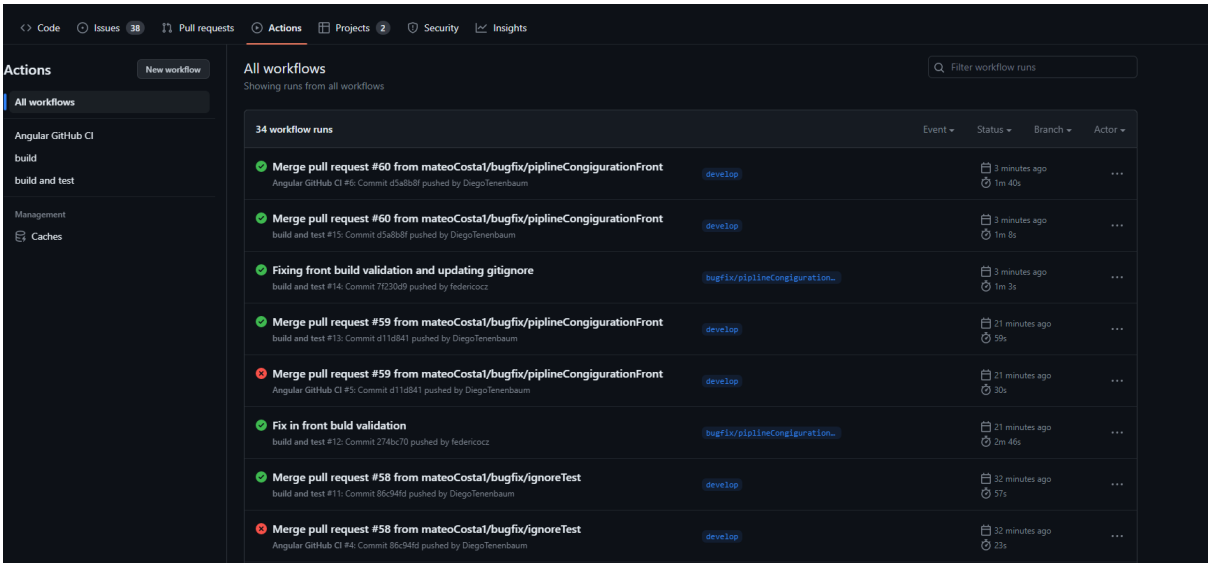
Para configurar el pipeline se siguió la documentación de Microsoft sobre GitHub Actions y .NET. La idea se basa en automatizar el código del repositorio respecto al Backend y Frontend, con el fin de aportar mejoras considerables a la Integración Continua(CI) y la Entrega Continua (CD). Mediante la creación de un archivo yml, se escribieron las validaciones correspondientes para ejecutar una github action en el momento de pushear o mergear código a main y/o develop. Cabe destacar que el yaml del back, también ejecuta y valida que los test unitarios estén en correcto funcionamiento y pasen todas las pruebas. Con el fin de acelerar el funcionamiento, solamente se corren los build en Ubuntu.
Se adjunta una captura de pantalla del código del yml correspondiente a la validación del build del backend:

```yaml
name: build and test

on:
  push:
  pull_request:
    branches: [ develop, main ]
    paths:
     - 'Obligatorio/codigo/ArenaGestor/**.cs'
     - 'Obligatorio/codigo/ArenaGestor/**/*.csproj'
     - 'Obligatorio/codigo/ArenaGestor/*.sln'

env:
  DOTNET_VERSION: '5.0' # The .NET SDK version to use

jobs:
  build:
    name: build-${{matrix.os}}
    runs-on: ${{ matrix.os }}
    strategy:
      matrix:
        os: [ubuntu-latest]

    steps:
    - uses: actions/checkout@v3
    - name: Setup .NET Core
      uses: actions/setup-dotnet@v3
      with:
        dotnet-version: ${{ env.DOTNET_VERSION }}

    - name: Restore Dependencies
      run: dotnet restore Obligatorio/codigo/ArenaGestor/ArenaGestor.sln

    - name: Build
      run: dotnet build Obligatorio/codigo/ArenaGestor/ArenaGestor.sln --configuration Release

    - name: Test
      run: dotnet test Obligatorio/codigo/ArenaGestor/ArenaGestor.sln --no-restore --verbosity normal
```

Se adjunta una captura de pantalla del código yml correspondiente a la validación del build del frontend:

```yaml
name: Angular GitHub CI #
on:
  push:
    branches: [ develop, main ]
jobs:
  ci:
    runs-on: ubuntu-latest

    steps:
      - name: Checkout
        uses: actions/checkout@v2

      - name: Use Node.js
        uses: actions/setup-node@v1
        with:
          node-version: "16.13.0"

      - name: Install Angular CLI
        run: npm install -g @angular/cli
        working-directory: ./Obligatorio/codigo/ArenaGestorFront/

      - name: Cache node modules
        id: cache-nodemodules
        uses: actions/cache@v2
        env:
          cache-name: cache-node-modules
        with:
          # caching node_modules
          path: node_modules #  path for node_modules folder
          key: ${{ runner.os }}-build-${{ env.cache-name }}-${{ hashFiles('**/package-lock.json') }}
          #  name of the chache key includes package-lock.json
          restore-keys: |
            ${{ runner.os }}-build-${{ env.cache-name }}-
            ${{ runner.os }}-build-
            ${{ runner.os }}-

      - name: Install Dependencies
        if:
          steps.cache-nodemodules.outputs.cache-hit != 'true'
          #  if cache hits the skip this step.
        run: npm ci
        working-directory: ./Obligatorio/codigo/ArenaGestorFront/

      - name: Build
        run: ng build --prod
        working-directory: ./Obligatorio/codigo/ArenaGestorFront/
```

Los archivo yml, quedaron en la ruta del repositorio “.github/workflows/l” Como se nota en la imagen, se especifica bajo qué acciones se desea que se ejecute el build, mediante la línea [on:] seguido de los paths donde se encuentran los archivos a correr. Se adjuntan las imágenes de las ejecuciones:



Se actualizó la configuración del pipeline mediante archivos de validación del build de la aplicación front-end y back-end en los archivos denominados: build-validation-front y build-validation-back respectivamente que son archivos yaml donde se especifica las acciones a seguir para generar el build de las dos aplicaciones.
Específicamente lo que se hace como steps en los archivos de configuración yml es la instalación de dependencias necesarias para ejecutar el build; se ejecuta el build; corren los test unitarios de la solución de manera automática.

De esta manera, cada vez que se pushea o se realiza un pull request en las ramas main o develop, se ejecuta una github action para chequear si el build se genera correctamente, incluyendo el pasaje correcto de todos los test unitarios del proyecto.

Cabe destacar el aporte que esto da en el marco que se está trabajando, ya que se gana tiempo en cuanto a las configuraciones, ejecuciones y corrida de test, pudiendo contribuir a la entrega continua y a la continua integración como se desea.