howest
hogeschool

PE 2
Automated Workflow

Continuous Integration Advanced

Graduaat in het Programmeren

DEEL 1 – ACTIONS 2 PT

Aanvaard de **Classroom Assignment** voor **deel 1** zodat je een startsituatie krijgt in een eigen repository. Je hoeft deze repository **nog niet te clonen**.

1 REPOSITORY SETTINGS

De repository bevat al een **main** branch.

- Maak een nieuwe branch genaamd **dev**.
- Stel de **dev** branch in als de **standaard branch** waarop geprogrammeerd wordt.
- Stel zowel de **main** als **dev** branch zodanig in dat er steeds een Pull Request nodig is wanneer iemand daar nieuwe code op wil integreren.

2 BUILD WORKFLOW

- Maak een nieuwe workflow aan via GitHub Actions.
  Kies, indiend gewenst, het workflow sjabloon genaamd **.NET by GitHub Actions**

- Noem het workflow bestand **integration.yml**

Zorg voor een workflow met de volgende eigenschappen realiseert:

- Naam van de workflow: **Integration**
- Wordt enkel getriggerd via Pull Requests naar de **dev** en **main** branches.
- Mag manueel worden getriggerd via het `workflow_dispatch` event. Zoek op wat dit is.
- Bevat 1 job genaamd **restore-build-test** die uitgevoerd wordt op **ubuntu-latest**.
- De job bevat zes stappen:

  1. *uses*: **Checkout** van de repository in de virtuele omgeving
  2. *name*: **Setup .NET**
     uses: **Setup** van de dotnet SDK
  3. *name*: **Output student name**
     run: Voer `echo "jouw naam hier"` uit als command line actie (vervang met je eigen naam!!)
  4. *name*: **Restore dependencies**
     run: Voer `dotnet restore` uit als command line actie.
  5. *name*: **Build solution**
     run: Voer `dotnet build` uit als command line actie.
  6. *name*: **Run all tests**
     run: Voer `dotnet test` uit als command line actie.

Opgepast, mocht je de solution nu reeds builden, dan zal deze een **error** geven. Dit passen we in een latere fase aan.

Er is reeds een job aanwezig die een aantal vaak gebruikte stappen bevat voor de dotnet setup. Je hoeft deze enkel nog wat aan te passen om met bovenstaande zaken overeen te laten komen.

- Zoek op in de GitHub Actions documentatie wat de working-directory variabele doet.

- Wijzig stappen 4, 5 en 6 zodanig de volgende working-directory variabele hebben ingesteld:

```
- name:
  working-directory: ./src
  run:
```

- In de cursusvoorbeelden was deze variabele niet nodig. Hier is dat wel zo. Weet je waarom?

- Zoek op wat de `--no-restore` en `--no-build` opties betekenen bij de twee laatste stappen.
  Welk voordeel bieden deze opties voor de uitvoering?

- Maak een **nieuwe feature branch aan** via GitHub en merge deze naar de **dev** branch

3 BLOKKEER PULL REQUESTS BIJ FALING

Een pull request mag niet gemerged worden als de **restore-build-test job** faalt.

- Keer terug naar je repository **Settings**.

Wijzig de branch protection rule voor zowel **dev** en **main** als volgt:

- Zorg ervoor dat alle **status checks** van je actions moeten **slagen** om te mogen mergen.
- Eis dat de feature branch **up-to-date** is met de integratie branch.
- Vereis de **restore-build-test** status check.

4 TRIGGER DE WORKFLOW

- **Clone** je repository naar je machine en/of synchroniseer deze met de remote dev branch.
- Maak een nieuwe feature branch aan genaamd `feature/randomservice-tests`.
- Maak in het test project genaamd Howest.Prog.CoinChop.Infrastructure.Tests een nieuwe klasse aan genaamd `RandomTokenServiceTests`.

Deze klasse zal de units bevatten voor de `RandomTokenService`. Deze laatste bevat een stuk functionaliteit om unieke strings (token) te genereren van een welbepaalde lengte.
De bijbehorende unit test is te vinden op deze plaats: https://github.com/howest-gp-cia/oe-achiefbestanden/blob/master/gists/pe02-01/RandomTokenServiceTests.cs

- Kopieer de code uit dit bestand naar jouw `RandomTokenServiceTests` file.
- Maak een **commit** met een passende boodschap.
- **Push** je feature branch naar je remote repository.

- Open een nieuwe **pull request** op GitHub om je branch te mergen met **dev**.

De workflow zou nu getriggerd moeten worden. Als dat niet zo is, moet je de workflow file corrigeren.

- Je merkt dat de Build action faalt vanwege een Build error in het Infrastructure project.
  Los dit probleem op in je lokale `feature/randomservice-tests` branch.

Tips: Controleer de action log van je job voor meer informatie, wijzig de `SmtpMailService` klasse.

Zorg dat de code **lokaal** compileert alvorens te committen.
Vergeet de bijgewerkte branch daarna niet te **pushen** zodat de wijzigingen zichtbaar zijn in de PR.

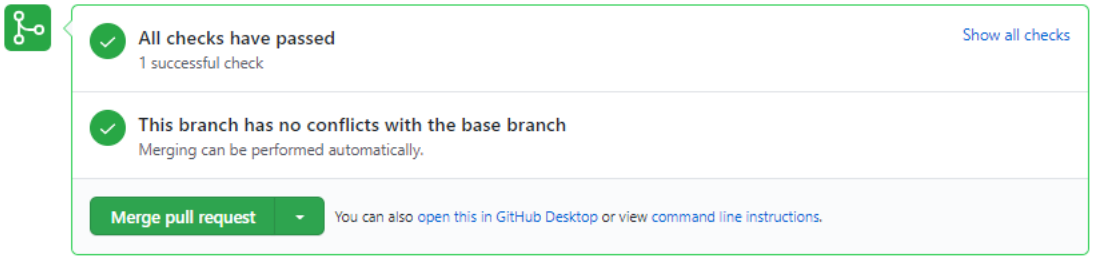
- Eens je het compilatieprobleem hebt opgelost, merk je dat jouw unit test faalt. Dit komt omdat de `GenerateToken` methode van de `RandomTokenService` klasse een logische bug bevat.
  Los dit probleem op in je lokale `feature/randomservice-tests` branch.

  Tips: Controleer de action log van je job voor meer informatie, ga op goed geluk zoeken in regel 13…

  Zorg dat de **unit test slaagt** op jouw lokale machine alvorens te committen.
  Vergeet de bijgewerkte branch niet te **pushen** zodat de wijzigingen zichtbaar zijn in de PR.

5 MERGE ON SUCCESS

Eens jouw feature de strenge status checks van de build server doorstaat, zou je nu jouw feature branch moeten kunnen mergen. Proficiat!



DEEL 2 - LINTING 3 PT

1.1 WAT IS LINTING

"Linting" is het proces waarbij de geschreven code wordt gecontroleerd op het correct gebruik van conventies. Deze conventies zijn bijvoorbeeld het correct gebruik van hoofdletters, declaratie van variabelen, het gebruik van accessors en meer.

Een Linter controleert niet op compilatiefouten zoals dat gebeurt bij een Build, en ook niet op logische fouten zoals dat gebeurt bij Unit Testing. Het gaat hier om een loutere esthetische check van de geschreven code zodat de code van elke developer in dezelfde stijl wordt geschreven. Dit verhoogt de leesbaarheid en het gemeenschappelijke code-ownership gevoel in een team.

1 SUPER-LINTER TOEVOEGEN

Een handige voorgedefinieerde action van de GitHub marketplace is **super-linter**. Deze action kan de linting verzorgen van zo maar eventjes 38 verschillende programmeer en/of configuratietalen, waaronder C#, javascript, CSS, SQL, maar ook YAML en JSON.

De homepage van deze tool vind je hier: https://github.com/marketplace/actions/super-linter

- Aanvaard de **Classroom Assignment** voor **deel 2** zodat je de startsituatie krijgt in je eigen repository. De repository bevat al een branch genaamd **master**.
- Maak een nieuwe branch genaamd **dev**.
- Stel de **dev** branch in als de **standaard branch** waarop geprogrammeerd wordt.

Neem de documentatie van super-linter door om hiervoor een workflow te configureren. Gebruik gerust het sjabloon dat aangeboden wordt in de documentatie, maar zorg voor het volgende:

- De workflow file moet **linter-dev.yml** heten.
- Geef de **job** de naam "linting".
- Zorg dat de workflow enkel getriggerd wordt bij een **pull request** naar **dev**.
- Zoek op in de documentatie hoe je linting inschakelt voor de **volledige codebase**.
- Zoek op in de documentatie hoe je **linting uitschakelt** voor de volgende bestandstypen, en stel dat zo in:
    1. CSS
    2. Javascript

- Zorg ervoor dat super-linter **dev** als de Default Branch behandelt.
- Maak de workflow file aan via een commit op de **dev** branch, rechtstreeks via GitHub.

2 SUPER-LINTER UITTESTEN

- Voeg een **README.md** toe aan je repository en **commit** deze in een branch genaamd `feature/fix-lint`.
- Maak hiervoor een nieuwe pul request aan.

Als alles goed is ingesteld zou de pull request de linter action moeten triggeren.

- Bekijk de linting problemen en probeer deze op te lossen vanuit Visual Studio

Blijf werken in de `feature/fix-lint branch` zodat je linting output kan opvolgen in de Pull Request.

3 AUTOMATISCHE BUILD VAN WPF

- Voeg een nieuwe workflow toe dat je het project automatisch kan compileren.
  Gebruik gerust het standaard dotnet sjabloon.
- Noem het bestand **build.yml**.
- Trigger de workflow.

Je merkt dat de code niet gecompileerd kan worden. Kijk in het logbestand waarom dat zo is. Tip: je hebt hier te maken met een WPF (Windows Presentation Foundation) project.

Los het probleem op in het workflow bestand. Indienen

Dien in voor de vermelde deadline op LEHO.

Je oplossing voor deze opdracht moet terug te vinden zijn in de GitHub repositories die aangemaakt worden met behulp van de GitHub classroom links die bij elk labo vermeld staan.