**Practical Exercise 32 - Reusing Workflows from Other Repositories**

**Exercise Description**

**In this practical exercise, our goal is to get familiar with creating and using reusable workflows between repositories.**

This exercise requires a few more setup steps, all of them outlined here. Here are the instructions for the exercise:

**1. Setting up a Cypress E2E testing project in a new repository:**

This first part is to setup a new Cypress project. You don't need any knowledge of Cypress, all you need is explained in these steps and in the solution videos.

1. Create a new repository in GitHub named github-actions-course-example-e2e.
   1. **Mark the repository as private** and provide a reasonable description.
   2. Check the box to add a README file to the repository.
   3. Scroll to the bottom and click on Create repository.
   4. Clone this to your local setup using your preferred method (https or ssh). **Do not clone this inside of the existing github-actions-course directory**. Instead, clone it as a sibling folder.
2. Open the new directory in Visual Studio Code and open a terminal in the new directory.
3. Initialize an npm project with npm init -y.
4. Install Cypress with npm install cypress@13.6.1 --save-dev --save-exact.
5. Once the installation is over, run npx cypress open to setup Cypress.
6. A new window should show up. Click on Continue and on E2E Testing.
7. Click on Continue once again, and then on Start E2E Testing in Chrome. A new window should show up once again.
8. On this new window, click on Scaffold example specs. This will generate several test files in the github-actions-course-example-e2e directory. Take a few minutes to inspect the files created under the cypress/e2e folder.
9. Go back to the window running the E2E tests, and click on any of the file names listed on the main screen. Take a few minutes to inspect how Cypress runs the tests, as well as the test results.
10. You can now close all Chrome windows, the setup is ready and we can use Cypress on the CLI.
11. Add a .gitignore folder and ignore the node\_modules folder.
12. Add a new entry to the scripts section of the package.json file:
    1. "test:e2e": "cypress run"
13. Run the Cypress tests with npm run test:e2e.
14. To have the tests running faster, delete the folder 2-advanced-examples folder and all its contents. This will leave a single file running, which is much faster than running all the files.
15. Commit and push all the changes. The Cypress setup step is done.

**2. Creating the E2E workflow within the new repository:**

1. Create a file named e2e.yaml under the .github/workflows folder of the github-actions-course-example-e2e repository.
2. Name the workflow E2E Tests.
3. Add the following triggers to the workflow:
   1. workflow\_dispatch
4. Add a single job named e2e-tests to the workflow.
   1. It should run on ubuntu-latest.
   2. It should contain four steps:
      1. The first step, named Checkout code, should checkout the code using the appropriate third-party action.
      2. The second step, named Setup node, should use the appropriate third-party action to setup Node with a version of '20.x'.
      3. The third step, named Install dependencies, should install the npm dependencies with npm ci.
      4. The fourth step, named Run E2E tests, should run the npm run test:e2e command.
5. Commit the changes and push the code. Trigger the workflow manually from the UI and take a few moments to inspect the result of the workflow run. Keep in mind that we should look at the new repository for the workflow!

**3. Running a reusable workflow from within another repository:**

1. In the original repository (github-actions-course):
   1. Make sure that you can run reusable workflows from other repository by going into the repository settings, then Actions, then General, and marking the "Allow all actions and reusable workflow" option at the top of the page.
   2. Create a file named 18-3-reusable-workflows.yaml under the .github/workflows folder.
      1. Name the workflow 18 - 3 - Reusable Workflows.
      2. Add the following triggers with event filters and activity types to your workflow:
         1. workflow\_dispatch
      3. Add a first job named deploy to the workflow.
         1. It should use the reusable workflow defined above. For that, simply use the uses key at the job level and pass the path of the workflow: ./.github/workflows/18-1-reusable-workflows.yaml
            1. Pass any value as a target-directory argument to the reusable workflow.
      4. Add a second job named e2e-tests to the workflow.
         1. It should have a dependency to the deploy job.
         2. It should use the E2E reusable workflow defined above. For that, simply use the uses key at the job level and pass the full identifier of the workflow: <owner>/<repository>/.github/workflows/e2e.yaml@<branch, tag, or sha commit>.
   3. Commit the code and push the changes.

This will not work yet. Let's now go through the steps necessary to make everything work.

1. In the new repository (github-actions-course-example-e2e):
   1. Add a workflow\_call trigger event to the e2e.yaml workflow definition file.
   2. Go into the repository settings -> Actions -> General, and scroll to the bottom. Under the **Access** heading, select the option "Accessible from repositories owned by the user..." (or in the organization xyz depending on whether you are using organizations). Save the changes.
2. If you the github-actions-course repository you created is public, it will not be able to run a reusable workflow from a private repository. Therefore, mark the github-actions-course repository as private:
   1. Go to the repository settings page and scroll to the bottom.
   2. In the **Danger zone** section, click on the Change Visibility button, and then Change to Public.
3. Once these steps are performed, try to retrigger the workflow. It will still fail, but for a different reason. We will work on that next.
4. In the new repository (github-actions-course-example-e2e):
   1. Change the e2e.yaml workflow definition file:
      1. Pass the following parameters to the checkout action:
         1. repository: <your username or oganization>/<your original github-actions-course repository>
         2. ref: main
   2. Commit the changes and push the code. If you retrigger the workflow, you will still see an access issue when checking out the code of the e2e repository. Let's fix this now.
   3. Under the workflow\_call trigger, add a secrets key and then a single secret named access-token, which should be marked as required.
   4. Pass an additional parameter to the checkout action:
      1. token: <retrieve the access-token input. If not available, default to secrets.GITHUB\_TOKEN>
5. In the original repository (github-actions-course):
   1. Create a new fine-grained Personal Access Token (PAT). If you are using organizations, you will have to authorize users to access the repository contents via PATs. I explain how to do that in the **Solution & Discussion - Part 2** video for this exercise, timestamp 13:44.
      1. Click on your picture, scroll down and click on Settings.
      2. On the new page, scroll down to Developer settings.
      3. Under Personal access tokens, click on fine-grained and then on Generate a new token.
      4. Select only the github-actions-course-example-e2e repository and give the token read access to the repository contents.
      5. Save the changes and store the token securely.
      6. Create a new repository secret (it can be created similarly to how we created repository variables) named GH\_TOKEN and save the value of the token as a secret.
   2. Now we can access the secret from within our 18-3-reusable-workflows.yaml file via the secrets context:
      1. Add a secrets key next to the uses key that references the e2e workflow and pass a single secret named access-token with a value of ${{ secrets.GH\_TOKEN }}.
6. Commit the changes and push the code. If all the steps were executed correctly, the reusable workflow should be correctly executed now. Congratulations!