**Practical Exercise 22 - Creating a Composite Custom Action**

**Exercise Description**

**In this practical exercise, our goal is to explore the syntax of and how to create composite custom actions in GitHub Actions.**

The goal of our composite custom action is to handle the installation and caching of npm dependencies, and to abstract that behind an easy-to-use reusable action. Here are the instructions for the exercise:

1. Create a file named action.yaml under the folder .github/actions/composite-cache-deps. This folder does not exist yet, so you might have to create it.
2. In the action.yaml file, add the following properties:
   1. A name of Cache Node and NPM Dependencies;
   2. A description of "This action allows to cache both Node and NPM dependencies based on the package-lock.json file.";
   3. The action should receive two inputs:
      1. The first one, named node-version, should be required, have a description of NodeJS version to use, and default to 20.x.
      2. The second one, named working-dir, should not be required, have a description of The working directory of the application, and default to the current directory if no input is passed.
   4. Add a top-level runs key. This is the core of defining our composite custom action. For a composite custom action, the runs key has the following shape:
      * 1. runs:
        2. using: composite
        3. steps: [...]

where:

* + 1. using: composite simply informs GitHub Actions that this is a composite custom action.
    2. steps: [...] contains the array of steps you wish to execute. This is very similar to the steps we have already defined for jobs in our previous exercises.
  1. Add the following steps under the steps key:
     1. The first step, named Setup NodeJS version <retrieve the node-version input value here>, should setup NodeJS by using the provided version from the inputs.
     2. The second step, named Cache dependencies, should have an id of cache, and use the actions/cache action to cache the node\_modules path **under the provided working directory**, and using a suitable key. Check the **Tips** section below for how to calculate the key effectively.
     3. The third step, named Install dependencies, should run if and only if there was not a cache hit in step two. It should run the npm ci script under the provided working-dir input, and it should use the bash shell.

1. Commit the changes and push the code. Take a few moments to understand the syntax of the composite custom action and the similarities and differences to the steps of job. For now nothing will happen, since this custom action is not used in any workflow, so let's move on to the next exercise where we create our workflow!

**Tips**

**Calculating the key for our cache**

When calculating the key, we should take into consideration our package-lock.json file which is under the provided working directory. We should not look at all package-lock.json files, since there may be more than one project within our repository (which is actually the case for us), and this may lead to incorrect changes in the cache key due to changes in unrelated files. In order to hash files and use the provided working directory from within an expression, you can use the following syntax: your-key-prefix-${{ hashFiles(format('{0}/{1}', inputs.working-dir, 'package-lock.json')) }}.