# Workshop: CI System with Selenium Appium Tests – Part I

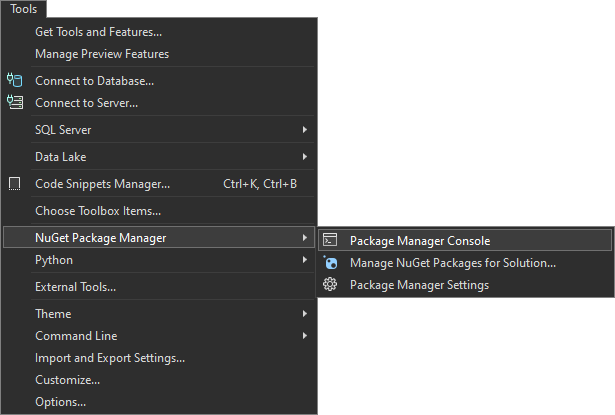
## Selenium IDE

### Step 1: Run the App Locally

We have the "SeleniumIde" solution in the **resources which has one test projects already**. Your task is to **create a CI workflow** with **GitHub Actions** to **run the tests automatically.**

It's a good practice to **build the solution locally** in Visual Studio, in order to be sure everything works properly and as expected.

Open **Visual Studio** and from there navigate to the **Tools** menu. Select **NuGet Package Manager** and select **Package** **Manager** **Console**:



Let's first build the application by using the following command:

|  |
| --- |
| **dotnet build** |

After you have **ensured** that the **build** was **successful**, you can **run** the **tests**, too, by using the command below or just by clicking on the **[Run All Tests in View]** button in the **Text Explorer**.

|  |
| --- |
| **dotnet test** |

**After** we have ensured that the **tests** **run** **successfully**, we can proceed with the next step.

### Step 2: Create a GitHub Repo

Now you should **upload the solution to** GitHub.

It's a good practice to start using the console and not the interface of GitHub, in case you haven't started doing so yet.

If you don't have Git already installed on your machine, follow the provided installation instructions from the resources.

Try using the following commands in order to initialize a repository in your project directory, add the code to the repo, commit and push:

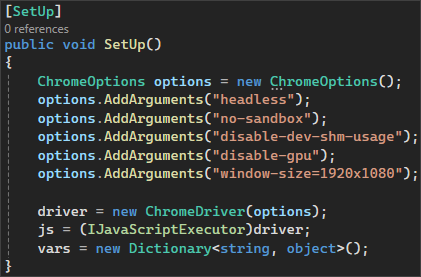
|  |
| --- |
| **git init**  **git add .**  **git commit -m "Initial commit"**  **git remote add origin** [**https://github.com/{name-of-your-repository}**](https://github.com/%7bname-of-your-repository%7d)  **git branch -M main**  **git push -u origin main** |

After running the commands, check you GitHub repo – the application code should be visible.

### Step 3: Add Changes to Test Files

Before creating the workflow file, we have to make some adjustments in the **.cs** files. This is needed due to the fact that the default GitHub runner does not have Chrome installed. We will take care of this in the workflow, but we also need the prepare the tests to run Chrome in a headless mode within the CI environment.

In order to do that, go to the **SetUp()** method of the project and modify it so it looks like below:



Don't forget to **commit** and **push** the changes from the file.

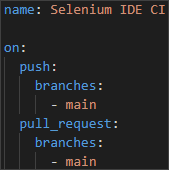
### Step 4: Create and Run Workflow

Now you should **create a** GitHubActionsCI **workflow** to **start and test the app**.

In the root directory of the repository, create a new folder **.github** and in it create another one, called **workflows**. Then, inside this new folder, create a **YAML** **file**, which will hold the workflow definition.

Now, let's define our workflow file.

We have to give it a meaningful name and specify the event which will trigger the workflow. In our case, this will be the push and pull request events on the **main** branch:



Then, we have to specify the **job** and the **environment**:

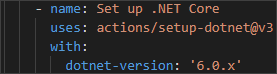


After that, we start defining the **steps.** You have to create several **steps** for the **job**:

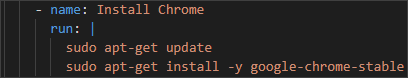
* **Checkout code**
  + Give the step a meaningful name
  + Checkout the repository code



* **Set up .NET Core**
  + Give the step a meaningful name
  + Use the appropriate action to set up the .NET Core SDK
    - Specify the .NET Core version



* **Install Chrome**
  + Give the step a meaningful name
  + Executes commands to update the package list and install Google Chrome



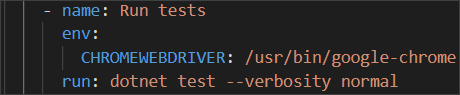
* **Install dependencies**
  + Give the step a meaningful name
  + Run the appropriate command to restore the dependencies specified in the solution file



* **Build the solution**
  + Give the step a meaningful name
  + Run the appropriate command to build the solution without restoring the dependencies again

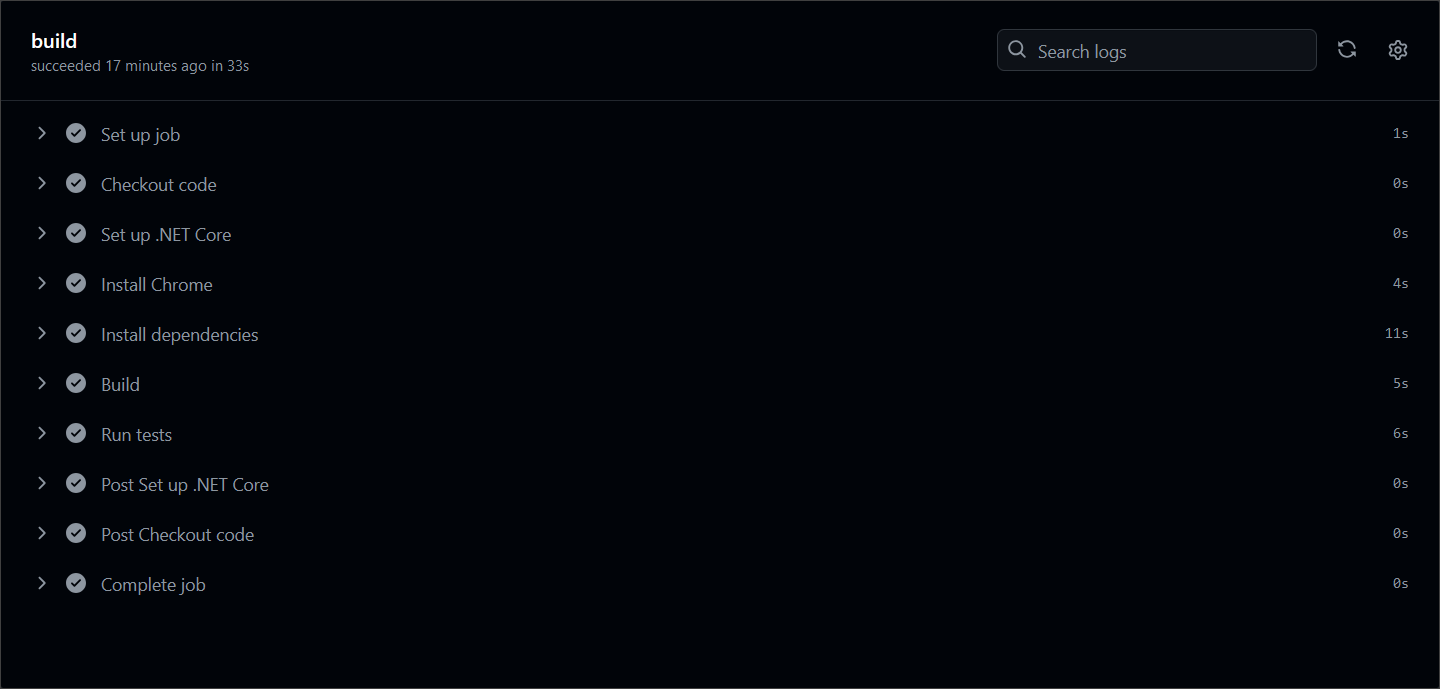


* **Run the test project** 
  + Give the step a meaningful name
  + Sets the environment variable **CHROMEWEBDRIVER** to the path of the Chrome executable
  + Run the tests in the project with normal verbosity



Now, commit the changes to the main branch of the repository.

Finally, **the workflow job should run after the commit.** Make sure that it is **successful**:



## Selenium Web Driver

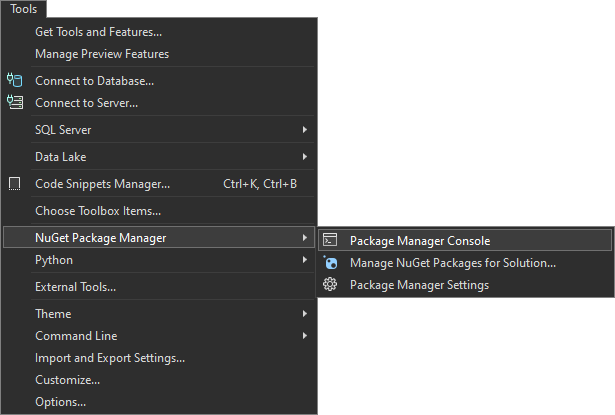
Our second task will be to create a CI for using Selenium to automate several test projects, combined in one solution.

### Step 1: Run the App Locally

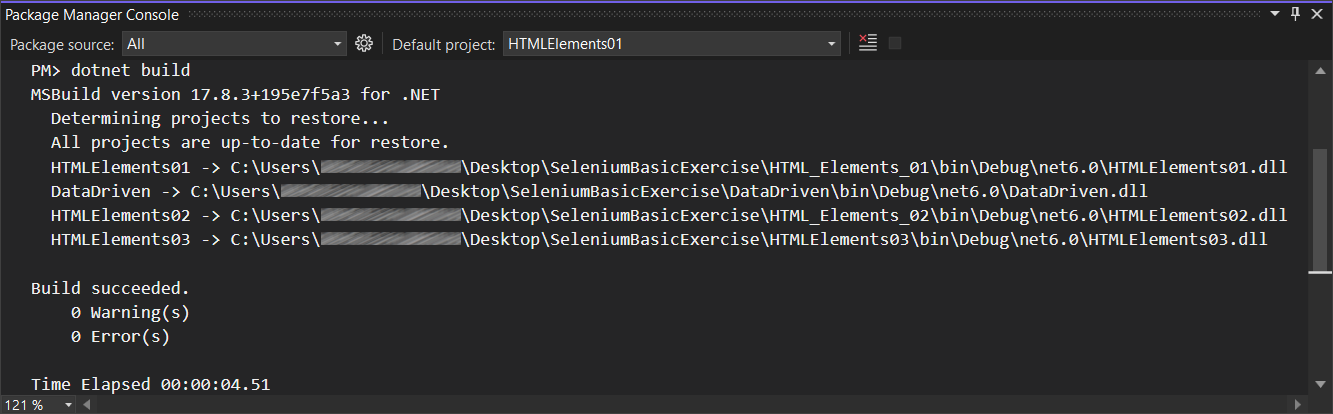
We have the "SeleniumBasicExercise" solution in the **resources which has two test projects already**. Your task is to **create a CI workflow** with **GitHub Actions** to **run the tests automatically.**

It's a good practice to **build the solution locally** in Visual Studio, in order to be sure everything works properly and as expected.

Open **Visual Studio** and from there navigate to the **Tools** menu. Select **NuGet Package Manager** and select **Package** **Manager** **Console**:



Let's first build the application by using the **dotnet build** command:



After you have **ensured** that the **build** was **successful**, you can **run** the **tests**, too, by using the **dotnet test** command or just by clicking on the **[Run All Tests in View]** button in the **Text Explorer**.

**After** we have ensured that the **tests** **run** **successfully**, we can proceed with the next step.

### Step 2: Create a GitHub Repo

Now you should **upload the solution to** GitHub.

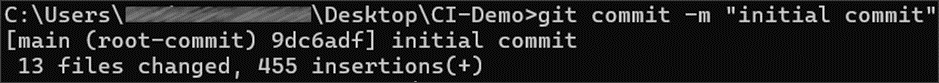
It's a good practice to start using the console and not the interface of GitHub, in case you haven't started doing so yet.

If you don't have Git already installed on your machine, follow the provided installation instructions from the resources.

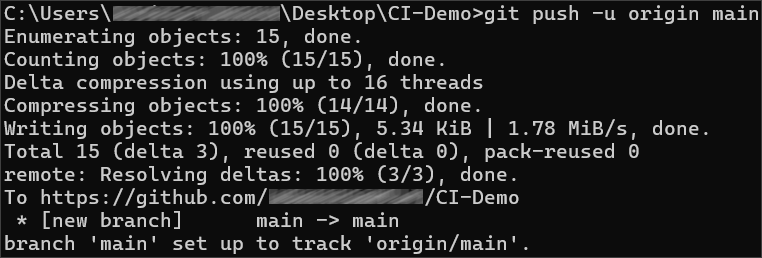
Try using the following commands in order to initialize a repository in your project directory, add the code to the repo, commit and push:







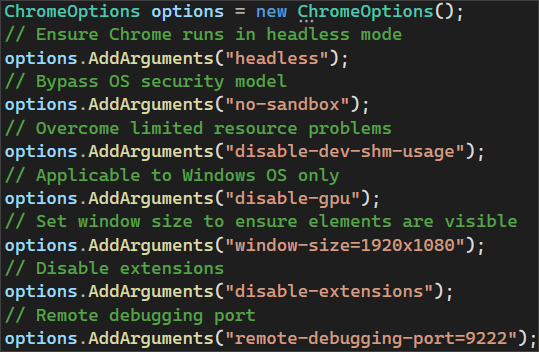




After running the commands, check you GitHub repo – the application code should be visible.

### Step 3: Add Changes to Test Files

Before creating the workflow file, we have to make some adjustments in the **.cs** files. This is needed due to the fact that the default GitHub runner does not have Chrome installed. We will take care of this in the workflow, but we also need the prepare the tests to run Chrome in a headless mode within the CI environment.

In order to do that, go to the **SetUp()** method of each project and add the following code:  


Then, we need to pass the **ChromeOptions** to the **ChromeDriver** constructor:



Don't forget to **commit** and **push** the changes to each one of the files.

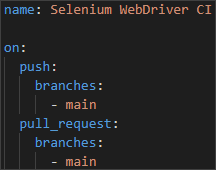
### Step 4: Create and Run Workflow

Now you should **create a** GitHubActionsCI **workflow** to **start and test the app**.

In the root directory of the repository, create a new folder **.github** and in it create another one, called **workflows**. Then, inside this new folder, create a **YAML** **file**, which will hold the workflow definition.

Now, let's define our workflow file.

We have to give it a meaningful name and specify the event which will trigger the workflow. In our case, this will be the push and pull request events on the **main** branch:



Then, we have to specify the **job** and the **environment**:

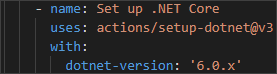


After that, we start defining the **steps.** You have to create several **steps** for the **job**:

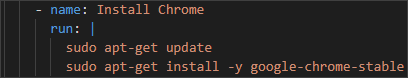
* **Checkout code**
  + Give the step a meaningful name
  + Checkout the repository code



* **Set up .NET Core**
  + Give the step a meaningful name
  + Use the appropriate action to set up the .NET Core SDK
    - Specify the .NET Core version



* **Install Chrome**
  + Give the step a meaningful name
  + Executes commands to update the package list and install Google Chrome



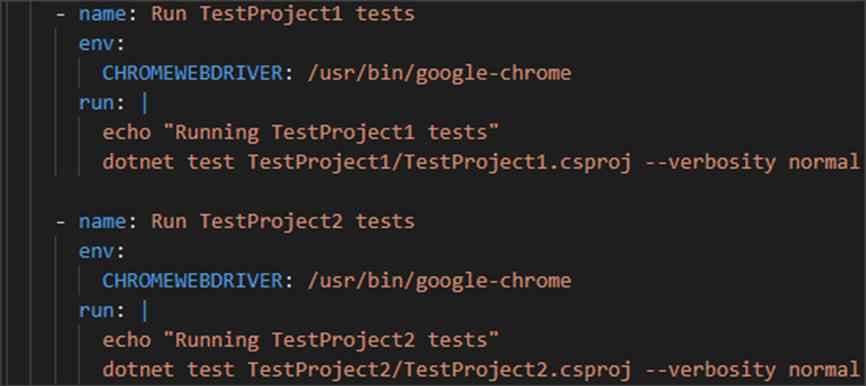
* **Install dependencies**
  + Give the step a meaningful name
  + Run the appropriate command to restore the dependencies specified in the solution file



* **Build the solution**
  + Give the step a meaningful name
  + Run the appropriate command to build the solution without restoring the dependencies again



* **Run each test project separately**
  + Give each step appropriate and meaningful name, describing which test project is being executed
  + Sets the environment variable **CHROMEWEBDRIVER** to the path of the Chrome executable
  + Run the tests in each project with normal verbosity



Now, commit the changes to the main branch of the repository.

Finally, **the workflow job should run after the commit.** Make sure that it is **successful**:

