How to Create a Lab on Unit Testing with Jasmine on Windows PCs

A step-by-step guide on how to install Jasmine, use GitHub Copilot, and integrate tests into a CICD pipeline

# Introduction

Jasmine is a popular framework for writing and running unit tests in JavaScript. It is easy to use, has a clear syntax, and supports asynchronous testing. In this document, you will learn how to create a lab on unit testing with Jasmine on Windows PCs. You will also learn how to use GitHub Copilot, an AI-powered code assistant, to generate Jasmine tests that can be integrated into a continuous integration and continuous delivery (CICD) pipeline.

# Prerequisites

Before you start, make sure you have the following:

* A Windows PC with an internet connection
* Visual Studio Code, a free and powerful code editor
* Node.js, a JavaScript runtime environment
* Git, a version control system
* A GitHub account

# Step 1: Install Jasmine

To install Jasmine, you need to use the Node Package Manager (npm), which comes with Node.js. Open a command prompt and type the following command:

npm **install** -g jasmine

This will install Jasmine globally on your PC. You can verify the installation by typing:

**jasmine -v**

You should see the version number of Jasmine displayed.

Initialize Jasmine in your project. This creates a spec directory and a configuration json file for your tests:

Open a terminal in VSC and run:

**npx jasmine init**

For later use, add Jasmine Reporters

npm **install jasmine-reporters** --save-dev

Create a test file in the spec directory. For example, you can create a file named `calculator.spec.js` and put your tests there.

Run your tests with the Jasmine command:

**npx jasmine**

# Step 2: Add Visual Studio Code Extensions for Jasmine

Visual Studio Code has a rich ecosystem of extensions that can enhance your coding experience. For this lab, you will need two extensions: Jasmine Test Explorer and GitHub Copilot.

To install Jasmine Test Explorer, open Visual Studio Code and go to the Extensions tab on the left sidebar. Search for "Jasmine Test Explorer" and click on the Install button.

A screenshot of a computer

Description automatically generated

This extension will allow you to run and debug your Jasmine tests from within Visual Studio Code.

To install GitHub Copilot, you need to sign up for the technical preview at https://copilot.github.com/. Once you receive an invitation, you can install the extension from the same website. GitHub Copilot is an AI-powered code assistant that can help you write code faster and better. It can suggest code completions, generate tests, and even write entire functions for you.

# Step 3: Use GitHub Copilot to Generate Jasmine Tests

Now that you have installed Jasmine and the extensions, you can start writing some tests. For this lab, you will use GitHub Copilot to generate some tests for a simple calculator function. The function takes two numbers as arguments and returns their sum.

First, create a new folder on your PC and name it "jasmine-lab". Open this folder in Visual Studio Code and create a new file called "calculator.js". In this file, write the following code:

**function** **calculator**(a, b) {  
**return** a + b;  
}  
    
module.exports = {  
  calculator: calculator  
};

Open the CoPilot Chat Window, make sure calculator.js is the active tab

**generate** jasmine tests besure to **include** the require directive **for** calculator.js

GitHub Copilot: Here is a Jasmine test suite for your `calculator` function:

const utils = require('../calculator.js');  
  
describe("Calculator", function() {  
 it("should add two numbers correctly", function() {  
 let result = calculator(5, 3);  
 expect(**result**).toEqual(8);  
 });  
  
 it("should return the first number when the second number is 0", function() {  
 let result = calculator(5, 0);  
 expect(**result**).toEqual(5);  
 });  
  
 it("should return the second number when the first number is 0", function() {  
 let result = calculator(0, 3);  
 expect(**result**).toEqual(3);  
 });  
  
 it("should return 0 when both numbers are 0", function() {  
 let result = calculator(0, 0);  
 expect(**result**).toEqual(0);  
 });  
});

Create a test file in the spec directory. For example, you can create a file named `calculator.spec.js` and put your tests there.

A screenshot of a computer

Description automatically generated

CoPIlot was close but we need to make a few changes for Jasmine to run

const utils = require('../calculator.js');  
  
describe("Calculator", function() {  
    it("should add two numbers correctly", function() {  
        let result = utils.calculator(5, 3);  
        expect(**result**).toEqual(8);  
    });  
  
    it("should return the first number when the second number is 0", function() {  
        let result = utils.calculator(5, 0);  
        expect(**result**).toEqual(5);  
    });  
  
    it("should return the second number when the first number is 0", function() {  
        let result = utils.calculator(0, 3);  
        expect(**result**).toEqual(3);  
    });  
  
    it("should return 0 when both numbers are 0", function() {  
        let result = utils.calculator(0, 0);  
        expect(**result**).toEqual(0);  
    });  
});

What is interesting when I add the module prefix, “utils.<some function> The Test Explorer Options become visible, and if you have the test explorer in focus, you see the Control Options

A screenshot of a computer

Description automatically generated

We can now also run this in the terminal

**npx jasmine**

Jasmine will automatically look for test files in the spec directory and run them. Make sure your test file name matches the pattern `\*spec.js` or `\*Spec.js`, as this is the pattern Jasmine uses to find test files.

Save the file and switch to the Test Explorer tab on the left sidebar. You should see a arrow button that says "Run All tests". Click on it and if the tests are successful, you should see the results of the tests displayed in the same tab.

A screenshot of a computer

Description automatically generated

# Step 4: Integrate the Tests into a CICD Pipeline

The final step of this lab is to integrate the tests into a CICD pipeline. A CICD pipeline is a process that automates the building, testing, and deploying of your code. For this lab, you will use GitHub Actions, a service that allows you to create workflows for your GitHub repositories.

First, you need to create a GitHub repository for your project. Go to https://github.com/ and sign in with your account. Click on the "New" button and create a new repository called "jasmine-lab". Make sure to check the "Add a README file" option and click on the "Create repository" button.

Add you Github Token as an Actions Secret (Settings | Secrets and Variables) | New repository secret button)

A screenshot of a computer

Description automatically generated

Next, you need to push your local code to the GitHub repository. Go back to Visual Studio Code and open the Source Control tab on the left sidebar. You should see your files listed as untracked changes. Click on the "+" button to stage them and then enter a commit message in the text box. Click on the check mark button to commit the changes. Then, click on the "Publish" button to push the changes to GitHub. Or you can use command line git commands. They both read the .git file.

Finally, you need to create a GitHub Actions workflow for your project. Go to your GitHub repository and click on the "Actions" tab. You should see a list of suggested workflows. Click on the "Node.js" workflow and edit the file as follows:

name: Run Jasmine Tests  
  
on: [workflow\_dispatch] # other options: push, pull\_request.  
  
jobs:  
 test:  
 runs-on: ubuntu-latest  
  
 steps:  
 - name: Checkout code  
 uses: actions/checkout@v2  
   
 - name: Use Node.js  
 uses: actions/setup-node@v2  
 with:  
 node-version: '16'  
 - name: Install dependencies  
 run: npm ci  
  
 - name: Run tests  
 run: npm test  
  
 - name: Publish Test Results  
 uses: dorny/test-reporter@v1  
 if: always()  
 with:  
 github\_token: ${{ secrets.GITHUB\_TOKEN }}  
 reporter: 'jest-junit'  
 test\_results: 'junit-results/\*.xml'

This part can be removed

      - name: Publish Test Results

        uses: dorny/test-reporter@v1

        if: always()

        with:

          github\_token: ${{ secrets.GITHUB\_TOKEN }}

          reporter: 'jest-junit'

          test\_results: 'junit-results/\*.xml'

Save the file and commit the changes. This will create a workflow that runs on every push and pull request to the main branch.

* Install Node.js
* Install the dependencies
* Run the tests on a Windows machine.

You can see the status of the workflow by clicking on the "Actions" tab again. You should see a green check mark if the workflow succeeds or a red cross if it fails.

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

# Conclusion

In this document, you learned how to create a lab on unit testing with Jasmine on Windows PCs. You learned how to install Jasmine, use GitHub Copilot to generate tests, and integrate the tests into a CICD pipeline. You can use these skills to write and run tests for any JavaScript project. Check this against the sample used in this document - [johndohoneyjr/jasmine-test (github.com)](https://github.com/johndohoneyjr/jasmine-test).