



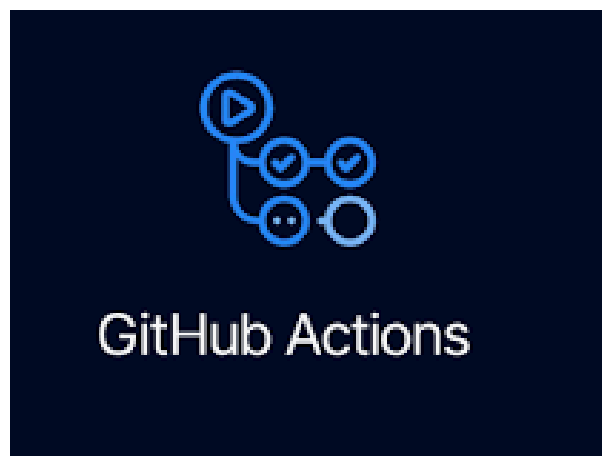
⚠ SMS 2FA method is prone to fraud and may be unreliable as it depends on delivery success rates in your region. To ensure consistent access to your account, please configure an additional 2FA method. [View 2FA settings](#) ×

Continuous Integration Testing Tool - Github Actions

[Jump to bottom](#)

Sa-dia edited this page 2 weeks ago · [5 revisions](#)

GitHub Actions CI Setup for JavaScript Project



Author: Trisha Sarkar

What is GitHub Actions?

GitHub Actions is an automation tool integrated with GitHub that allows you to build, test, and deploy code automatically based on events like pushes or pull requests. It is perfect for setting up Continuous Integration (CI) and Continuous Deployment (CD) pipelines.

What is Continuous Integration (CI)?

Continuous Integration (CI) is a software development practice where code changes are automatically tested and integrated into the shared codebase frequently. This ensures that errors or conflicts are caught early.

Installation

Prerequisites

- **Node.js:** Ensure you have Node.js installed. You can download it from [here](#).

Navigate to your project directory using the terminal.

```
PS F:\4-1\Academic Books> cd .\CI-CD-mocha-chai\  
PS F:\4-1\Academic Books\CI-CD-mocha-chai> 
```

Install npm dependencies (specified in your package.json file):

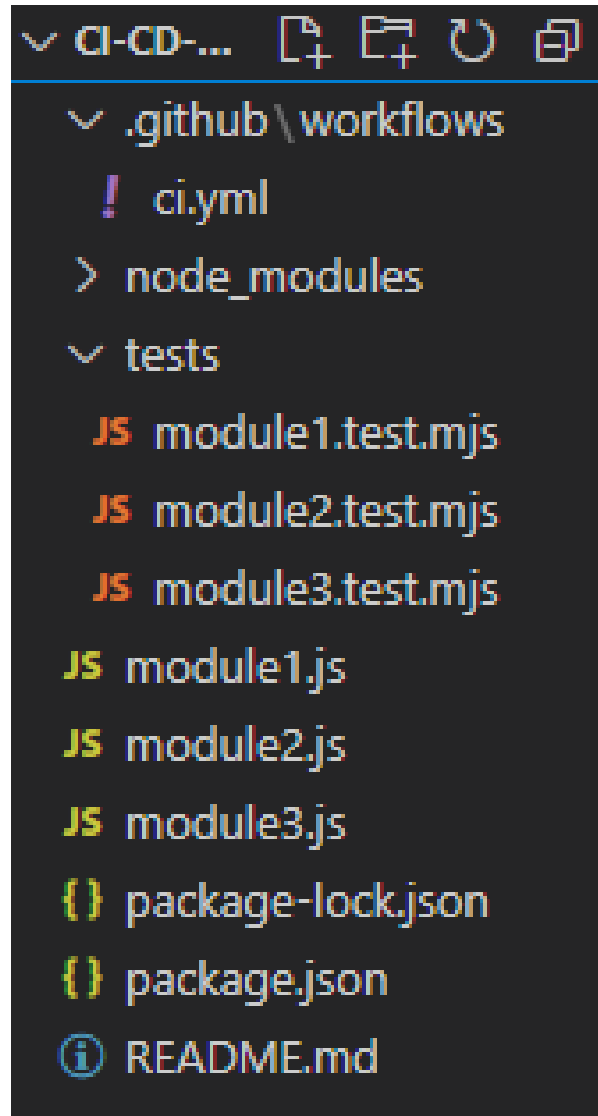
```
PS F:\4-1\Academic Books\CI-CD-mocha-chai> npm install  
  
up to date, audited 78 packages in 2s  
  
20 packages are looking for funding  
  run `npm fund` for details  
  
found 0 vulnerabilities
```

Install mocha-chai for testing

```
PS F:\4-1\Academic Books\CI-CD-mocha-chai> npm install --save-dev mocha chai  
npm warn deprecated glob@8.1.0: Glob versions prior to v9 are no longer supp  
  
added 77 packages, and audited 78 packages in 11s  
  
20 packages are looking for funding  
  run `npm fund` for details  
  
found 0 vulnerabilities
```

Project Structure

Your project should have the following structure:



- `module1.test.mjs` : Contains tests for the features and functions of `module1.mjs` .
- `module2.test.mjs` : Contains tests for the features and functions of `module2.mjs` .
- `module3.test.mjs` : Contains tests for the features and functions of `module3.mjs` .
- `package.json` : Project's package configuration file.
- `.github/workflows/ci.yml` : The GitHub Actions CI configuration file.

2. Create Module Files

You need to create three JavaScript module files: `module1.js` , `module2.js` , and `module3.js` .

module1.js

```
JS module1.js X
JS module1.js > ...
1  // module1.js
2  function someFunction() {
3      return "Hello from Module 1!";
4  }
5
6  module.exports = {
7      someFunction
8  };
9  |
```

module2.js

```
JS module2.js X
JS module2.js > ...
1  // module2.js
2  function performAction() {
3      return "Action performed in Module 2!";
4  }
5
6  module.exports = {
7      performAction
8  };
9  |
```

module3.js

```
JS module3.js X
JS module3.js > ...
1 // module3.js
2 function calculate() {
3   return 42; // The answer to life, the universe, and everything
4 }
5
6 module.exports = {
7   calculate
8 };
9
```

3. Set up tests for each module

Next, create the test files in the tests/ directory to ensure each module works correctly using Jest.

tests/module1.test.mjs

```
JS module1.test.mjs X
tests > JS module1.test.mjs > ...
1 // tests/module1.test.mjs
2 import pkg from '../module1.js'; // Import the entire module
3 const { someFunction } = pkg; // Destructure the required function
4
5 import { expect } from 'chai';
6
7 describe('Module 1', () => {
8   it('should return the correct value', () => {
9     expect(someFunction()).to.equal('Hello from Module 1!');
10   });
11 });
12
```

tests/module2.test.mjs

```
JS module2.test.mjs X
tests > JS module2.test.mjs > ...
1 // tests/module2.test.mjs
2 import pkg from '../module2.js'; // Import the entire module
3 const { performAction } = pkg; // Destructure the required function
4
5 import { expect } from 'chai';
6
7 describe('Module 2', () => {
8   it('should perform correctly', () => {
9     expect(performAction()).to.equal('Action performed in Module 2!');
10   });
11 });
12
```

tests/module3.test.mjs

```
JS module3.test.mjs X
tests > JS module3.test.mjs > ...
1 // tests/module3.test.mjs
2 import pkg from '../module3.js'; // Import the entire module
3 const { calculate } = pkg; // Destructure the required function
4
5 import { expect } from 'chai';
6
7 describe('Module 3', () => {
8   it('should return correct output', () => {
9     expect(calculate()).to.equal(42);
10   });
11 });
12
```

4. Configure package.json file

Make sure your package.json contains the necessary scripts to run each module's tests individually as well as collectively.

```
{ package.json X
{ package.json > ...
1 {
2   "name": "my-js-project",
3   "version": "1.0.0",
4   "description": "JavaScript project with 3 modules",
5   "main": "index.js",
6   "scripts": {
7     "test": "mocha 'tests/**/*.mjs'",
8     "test:module1": "npx mocha 'tests/module1.test.mjs'",
9     "test:module2": "npx mocha 'tests/module2.test.mjs'",
10    "test:module3": "npx mocha 'tests/module3.test.mjs'"
11  },
12  "devDependencies": {
13    "chai": "^5.1.2",
14    "mocha": "^10.7.3"
15  }
16 }
17
```

5. Set Up GitHub Actions for CI

Create the GitHub Actions configuration file to set up continuous integration. The file is located at `.github/workflows/ci.yml`.

```
! ci.yml X
.github > workflows > ! ci.yml
1  name: CI for JavaScript Modules
2
3  on:
4    push:
5      branches:
6        - main
7    pull_request:
8      branches:
9        - main
10
11  jobs:
12    build-and-test:
13      runs-on: ubuntu-latest
14
15      steps:
16        - name: Checkout code
17          uses: actions/checkout@v3
18
19        - name: Set up Node.js
20          uses: actions/setup-node@v3
21          with:
22            node-version: '16'
23
24        - name: Install dependencies
25          run: npm install
26
27        - name: Fix Jest permission issue
28          run: chmod +x ./node_modules/.bin/mocha || true
29
30        - name: Run Tests for Module 1
31          run: npm run test:module1
32
33        - name: Run Tests for Module 2
34          run: npm run test:module2
35
36        - name: Run Tests for Module 3
37          run: npm run test:module3
38
```

6. Testing CI Success

1. Run test for module1:

```
PS F:\4-1\Academic Books\CI-CD-mocha-chai> npm run test:module1
> my-js-project@1.0.0 test:module1

Module 1
  ✓ should return the correct value

1 passing (8ms)
```

2. Run test for module2:

```
PS F:\4-1\Academic Books\CI-CD-mocha-chai> npm run test:module2
> my-js-project@1.0.0 test:module2

Module 2
  ✓ should perform correctly

1 passing (10ms)
```

3. Run test for module3:

```
PS F:\4-1\Academic Books\CI-CD-mocha-chai> npm run test:module3
> my-js-project@1.0.0 test:module3

Module 3
  ✓ should return correct output

1 passing (7ms)
```

1. Run all tests:


```
PS F:\4-1\Academic Books\CI-CD-mocha-chai> npm run test

> my-js-project@1.0.0 test
> mocha 'tests/**/*.mjs'


Module 1
  ✓ should return the correct value

Module 2
Module 2
  ✓ should perform correctly

Module 3
  ✓ should return correct output
  ✓ should perform correctly

Module 3
  ✓ should return correct output

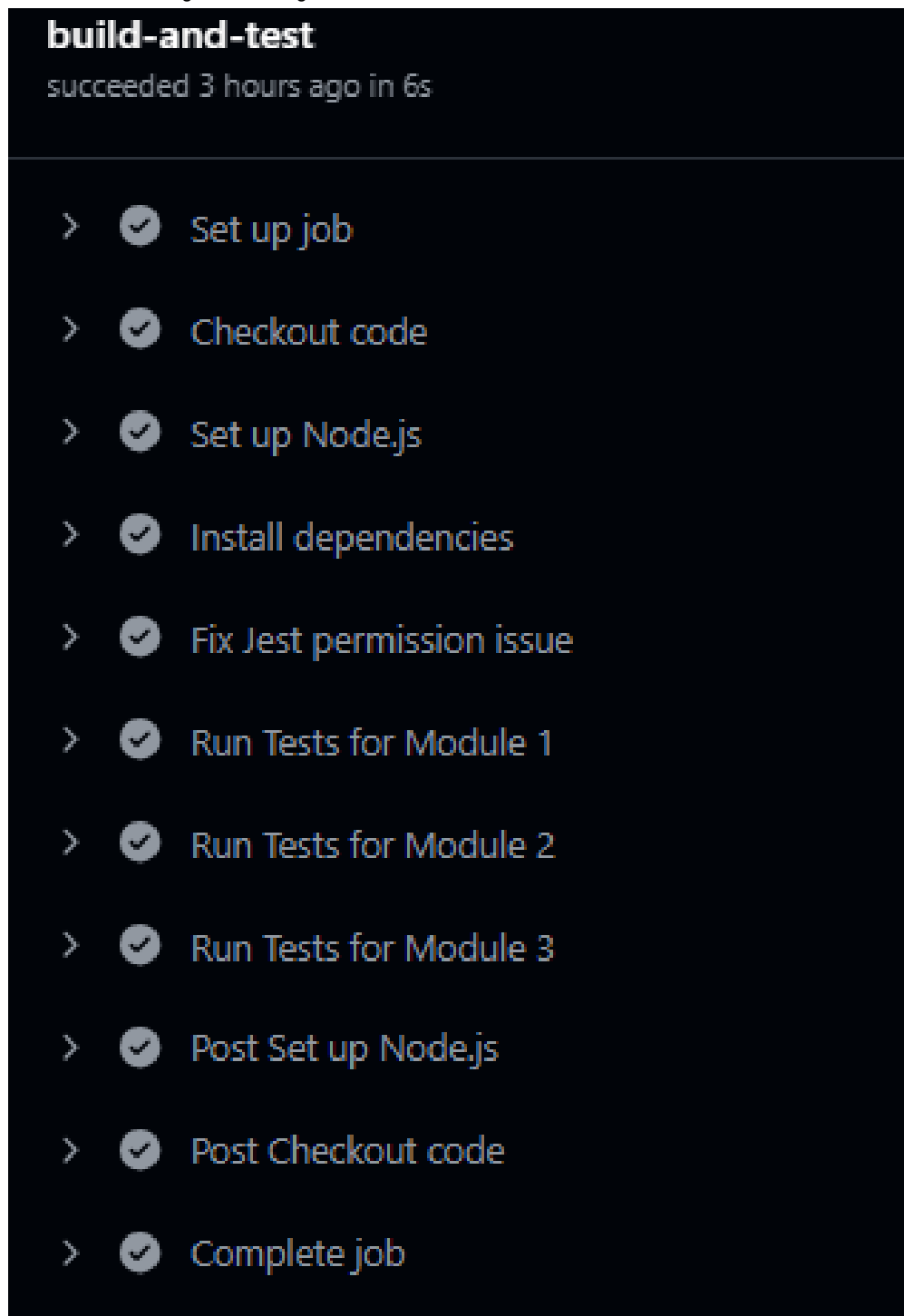
Module 3
  ✓ should return correct output
Module 3
  ✓ should return correct output

3 passing (14ms)
  ✓ should return correct output

3 passing (14ms)
```

CI Testing

Once you've pushed your changes to the main branch, GitHub Actions will automatically run the tests. You can view the results in the Actions tab of your GitHub repository.



Why Github Actions?

Advantages of Github Actions

1. **Tightly integrated with GitHub:** Easy setup and workflow creation within the GitHub ecosystem.
2. **No external tools needed:** Unlike CircleCI or Semaphore, you don't need to rely on third-party CI tools.

3. **Free for public repositories:** GitHub Actions offers generous free-tier minutes for open-source projects.
4. **Parallel workflows:** Supports running multiple jobs (like tests) simultaneously.

Disadvantages of GitHub Actions

- **Limited Free Minutes for Private Repositories:** GitHub Actions provides only 2,000 minutes/month for private repositories, which may not be sufficient for projects with heavy CI/CD usage.
- **Complexity for Advanced Workflows:** Managing complex workflows with multiple jobs and conditional logic can become challenging, especially without detailed documentation.
- **Lack of Advanced Analytics:** Compared to CI platforms like CircleCI, GitHub Actions does not provide detailed build insights or performance analytics.
- **Slower Support Response:** Official GitHub support can be slow, making it harder to resolve issues quickly compared to CI platforms with dedicated support teams.
- **Learning Curve for YAML Configurations:** YAML-based workflow files, though flexible, can be hard to set up for beginners, especially for advanced configurations.

CI/CD Tools Comparison: GitHub Actions vs CircleCI vs Semaphore

Below is a comparison table outlining the key differences between **GitHub Actions**, **CircleCI**, and **Semaphore** based on various factors such as pricing, ease of use, integration, and more.

Feature	GitHub Actions	CircleCI	Semaphore
Integration	Built-in with GitHub, seamless integration with GitHub repositories and PRs	Integrates with GitHub and Bitbucket	Integrates with GitHub
Free-Tier (Public Repos)	Unlimited usage for public repositories	Unlimited builds for open-source repositories	Unlimited for open-source projects


Feature	GitHub Actions	CircleCI	Semaphore
Free-Tier (Private Repos)	2,000 minutes per month (combined across jobs)	2,500 credits per week for free-tier	3,000 minutes per month for free-tier
Paid Plans	\$0.008 per minute after free-tier minutes used	Paid based on credits; custom plans available	Paid based on minutes used; custom plans
Ease of Use	Simple for basic workflows, more complex for advanced workflows	Easy to use for both simple and advanced pipelines	Simple setup for parallel and sequential workflows
Workflow Complexity	Can get complex for advanced multi-job workflows	Good for complex workflows with detailed documentation	Easy to manage complex workflows with clear documentation
Documentation	Good for basic workflows, lacks details for advanced cases	Extensive documentation and tutorials available	Clear documentation, especially for advanced setups
Parallelism	Supports parallel jobs with custom workflows	Supports parallel jobs and builds across multiple containers	Excellent parallelism support with clear scaling options
Speed	Fast for simple builds, can slow down with complex workflows	Generally fast, especially with container caching	Very fast and optimized for parallel jobs
Customization	High level of customization with YAML workflows	Highly customizable with workflow visualizer	Highly customizable with intuitive YAML syntax
Docker Support	Full Docker support	Full Docker support	Full Docker support
Third-Party Integrations	Integrates with popular third-party apps via GitHub Marketplace	Supports many integrations via API and orbs	Limited third-party integrations


Feature	GitHub Actions	CircleCI	Semaphore
Analytics and Insights	Basic insights with no in-depth analytics	Detailed analytics, build time, and performance insights	Good insights and performance data
Community and Support	Large GitHub community, but support can be slow	Extensive community support and faster response times	Smaller community but dedicated support
Best For	Developers using GitHub for both code and CI/CD	Teams looking for robust, scalable CI/CD options	Teams seeking fast, parallel, and highly customizable pipelines


Key Takeaways:


- **GitHub Actions** is best suited for developers already using GitHub who want seamless CI/CD integration. It offers a good free-tier for public repositories but has limitations for private repos.
- **CircleCI** is ideal for teams that need scalability and detailed analytics for complex workflows, with great support for Docker and parallel builds.
- **Semaphore** excels in fast build times and is highly optimized for parallelism, making it a good choice for teams with complex, performance-focused CI/CD pipelines.

▸ Pages 52

 [SRS](#)

 [DB Schema](#)

 [DFD](#)

 [Coding Standard](#)



Unit Testing Tool

Clone this wiki locally

<https://github.com/JUCSE49-Mavericks/Smart-Class-Routine-Management-System.wiki.git>

