JavaScript Development Environments

JavaScript

### Specified Single File

#### Run

Open a terminal and enter the command.

node hello.js

#### Run and Watch

Setup package.json if you have not already

npm init --yes

Install the nodemon node package as a development dependency.

npm install --save-dev nodemon

If we want to run the dev dependency from the terminal we use the npx command

npx nodemon hello.js

#### Run as Script

As we install it as a dev dependency, we can only run it from the scripts section of package.json

{

"name": "JS",

"version": "1.0.0",

"description": "",

"main": "test.js",

"scripts": {

"test": "echo \"Error: no test specified\" && exit 1",

**"watch" : "nodemon hello.js"**

},

"keywords": [],

"author": "",

"license": "ISC",

"devDependencies": {

"install": "^0.13.0",

"nodemon": "^2.0.4",

"npm": "^6.14.8"

}

}

Run the script npm run watch

#### Debug

{

"version": "0.2.0",

"configurations": [

{

"type": "node",

"request": "launch",

"name": "Launch Program",

"skipFiles": [

"<node\_internals>/\*\*"

],

**"program": "${workspaceFolder}\\hello.js"** }

]

}

You can now run or debug the file which has focus by using the command Ctrl-F5 or F5 respectively on windows.

#### Debug with Watch

Setup a launch.json target as follows. Make sure nodemon is installed globally

        {

            "name": "Launch server.js via nodemon",

            "type": "node",

            "request": "launch",

            "runtimeExecutable": "nodemon",

            "program": "${workspaceFolder}/hello.js",

            "restart": true,

            "console": "integrated

Terminal",

            "internalConsoleOptions": "neverOpen"

          }

Now run or debug it using Ctrl-F5 or F5 respectively

For more details see

<https://code.visualstudio.com/docs/nodejs/nodejs-debugging>

### Currently Selected File

#### Debug

Add the following to your launch.json

{

"version": "0.2.0",

"configurations": [

{

"type": "node",

"request": "launch",

"name": "Launch Program",

"skipFiles": [

"<node\_internals>/\*\*"

],

**"program": "${file}"** }

]

}

Now use Ctrl-F5 or F5 to run or debug the currently selected file

#### Unit Test All Files

### Tests

#### Run All Tests

First, we install jest

npm install --save-dev jest

Now we can run all the tests as

npx jest

#### Run single test file

npx jest myModule.test

#### Run Specified Test

npx jest myModule.test -t=<TestName>

#### Run all tests in Debug Mode

Add the following to vs code on Mac and run debug from the VS Code console. You will need something else on windows.

{

"name": "Debug tests single run",

    "type": "node",

    "request": "launch",

    "env": { "CI": "true" },

    "runtimeExecutable": "${workspaceRoot}/node\_modules/.bin/jest",

    "args": ["test", "--runInBand", "--no-cache"],

    "cwd": "${workspaceRoot}",

    "protocol": "inspector",

    "console": "integratedTerminal",

    "internalConsoleOptions": "neverOpen"

}

#### Run Single Test File in Debug Mode

{

"name": "Debug single tests single run",

    "type": "node",

    "request": "launch",

    "env": { "CI": "true" },

    "runtimeExecutable": "${workspaceRoot}/node\_modules/.bin/jest",

    "args": ["--runInBand", "--no-cache"],

    "cwd": "${workspaceRoot}",

    "program": "${fileBasenameNoExtension}",

    "protocol": "inspector",

    "console": "integratedTerminal",

    "internalConsoleOptions": "neverOpen"

}

#### Run Single Test File in Debug Mode With Watch

{

"name": "Debug single tests single run",

    "type": "node",

    "request": "launch",

    "env": { "CI": "true" },

    "runtimeExecutable": "${workspaceRoot}/node\_modules/.bin/jest",

    "args": ["--runInBand", "--no-cache", "--watchAll"],

    "cwd": "${workspaceRoot}",

    "program": "${fileBasenameNoExtension}",

    "protocol": "inspector",

    "console": "integratedTerminal",

    "internalConsoleOptions": "neverOpen"

}

JavaScript and React (CRA)

#### Run all tests with Filewatch

If you create you react app using npx create-react-app my-react-app then this all tests with filewatch is the default for the npm test script. From the terminal just enter the following command

npm test

#### Debug All Tests

Add the following code to your launch.config

{

"name": "Debug CRA Tests",

"type": "node",

"request": "launch",

"runtimeExecutable": "${workspaceRoot}/node\_modules/.bin/react-scripts",

"args": ["test", "--runInBand", "--no-cache", "--watchAll=false"],

"cwd": "${workspaceRoot}",

"protocol": "inspector",

"console": "integratedTerminal",

"internalConsoleOptions": "neverOpen",

"env": { "CI": "true" },

"disableOptimisticBPs": true

}

#### Debug Single Test File

## Typescript

### Setup

#### Node Commands

|  |  |
| --- | --- |
| File/Folder/Command | Details |
| npm install | Install all packages specified in package.json |
| npm list | Show all local packages and their dependencies |
| npm run | Run a script specified in package.json |

#### JavaScript/Typescript Project Structure

|  |  |
| --- | --- |
| File/Folder/Command | Details |
| package.json | Describes a project’s top-level dependencies. These are packages that have been added to a project using npm install |
| package-lock.json | All package dependencies for the project |
| tsconfig.json | TypeScript compiler configuration |

#### Node Packages

npm init –yes

npm install –-save-dev typescript ➊

npm install –save-dev tsc-watch ➋

npm install --save-dev jest ➌

npm install --save-dev @types/jest ➍

npm install --save-dev ts-jest ➎

|  |  |
| --- | --- |
| ➊ typescript | The typescript compiler |
| ➋ tsc-watch | Watches typescript files for changes. When it sees a change, it compiles. It can be configured to run a resulting JavaScript file after compilation |
| ➌ jest | JavaScript testing framework |
| ➍ @types/jest | Typescript types for the jest framework |
| ➎ ts-test | Test utilities for TypeScript |

#### TypeScript Compiler Options

Listing tsconfig.json

{

    "compilerOptions": {

        "target": "ES2018", ➋

        "outDir": "./dist",

        "rootDir": "./src",

        "noEmitOnError": true,

        "sourceMap": true,

        "module": "commonjs" ➊

    }

}

|  |  |
| --- | --- |
| ➊ module format | Some environments such as node do not support ES2015 modules so specifying commonjs tells the compiler to generate older module code |
| ➋ target | The version of JavaScript to target |

#### Package.json

{

  "name": "tools",

  "version": "1.0.0",

  "description": "",

  "main": "index.js",

  "scripts": {

**"test": "npx jest --watchAll",**

**"start": "tsc-watch --onsuccess \" node dist/index.js\""**

  },

  "keywords": [],

  "author": "",

  "license": "ISC",

  "devDependencies": {

    "tsc-watch": "^4.2.3",

    "typescript": "^3.8.3"

  }

}

The bold lines specify scripts that can be run by npm. We have added a script called start that monitors files for change and executes the index.js when changed files have been compiled

### Debugging

If we want to debug in VSCode we need to add a folder called .vscode into which we add a file called launch.json

{

    // Use IntelliSense to learn about possible attributes.

    // Hover to view descriptions of existing attributes.

    // For more information, visit: https://go.microsoft.com/fwlink/?linkid=830387

    "version": "0.2.0",

    "configurations": [

        {

            "type": "node",

            "request": "launch",

            "name": "Launch Program",

            "program": "${workspaceFolder}\\dist\\index.js",

        }

    ]

}

We can then run our debugger using F5 in visual studio code

### Unit Testing

Unit testing with Jest consists of two parts. The first part is to setup a configuration file called jest.config.js at the root level of our project. The following is a good example.

module.exports = {

    "roots": ["src"],

    "transform":{"^.+\\.tsx?$": "ts-jest"}

}

Then we simply add tests in our source code folder. If we have a module called adder.ts as follows

export function add(a: number, b: number): number {

    return a+b;

}

We can create a test called adder.test.ts as follows

import {add} from "./adder";

test("do a test", () => {

    let result = add(10,5);

    expect(result).toBe(15);

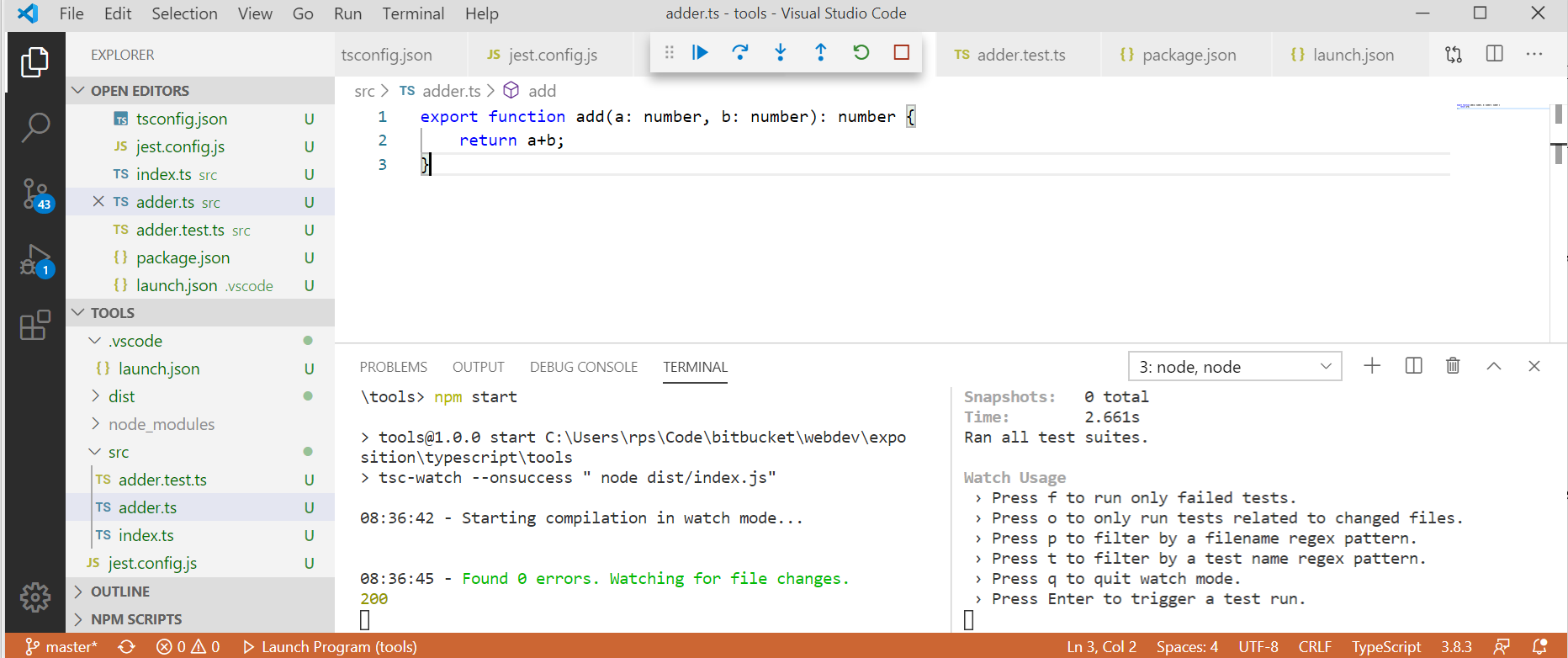
})

### Putting it together

Often it is useful to have two terminal windows: one with a file watcher compiling and running our application and one running the tests.

npm start

npm test



### Specified Single File

We need to make sure we have the typescript compiler installed

npm init –yes

npm install –-save-dev typescript

We need to create a typescript compiler configuration file

Listing tsconfig.json

{

    "compilerOptions": {

        "target": "ES2018", ➋

        "outDir": "./dist",

        "rootDir": "./src",

        "noEmitOnError": true,

        "sourceMap": true,

        "module": "commonjs" ➊

    }

}

|  |  |
| --- | --- |
| ➊ module format | Some environments such as node do not support ES2015 modules so specifying commonjs tells the compiler to generate older module code |
| ➋ target | The version of JavaScript to target |

#### Compile

When we run tsc from the command line with no arguments it will compile TypeScript source files in the rootDir to JavaScript files in the outDir

npx tsc

#### Run specified single File

We run JavaScript and not TypeScript, so the command is then

node dist/hello.js

#### Run specified single File with watch

To run typescript in watch mode we need an extra package called tsc-watch

npm install –save-dev tsc-watch

We then need to add a line to the scripts section in our package.json

"scripts": {

"test": "npx jest --watchAll",

**"start": "tsc-watch --onsuccess \" node dist/hello.js\""**

  },

### Selected File

#### Run/Debug Specified File No Watch

Setup the lauch.json with a configuration as follows.

{

    "version": "0.2.0",

    "configurations": [

        {

            "type": "node",

            "request": "launch",

            "name": "Run/Debug Open File",

            "skipFiles": [

                "<node\_internals>/\*\*"

            ],

            "program": "${file}",

            "preLaunchTask": "tsc: build - tsconfig.json",

            "outFiles": [

                "${workspaceFolder}/\*\*/\*.js"

            ]

        }

    ]

}

You can then run/debug the current file using Ctrl-F5 or F5

### Jest

#### run all tests no watch

To use jest with typescript we need the following

npm install --save-dev jest

npm install --save-dev @types/jest

npm install --save-dev @babel/preset-typescript

We also need a file called babel.config.js

module.exports = {

    presets: [

      ['@babel/preset-env', {targets: {node: 'current'}}],

  +    '@babel/preset-typescript',

    ],

  };

Finally, we run the tests as follows in the terminal

npx jest

#### run all tests With watch

npx jest --watchAll

#### Run Single File Test No Watch

Run the test in hello2.test.ts Note we miss off the .ts from the filename

npx jest hello2.test

#### Run Single File Test Watch

npx jest hello2.test--watch

#### Debug/Run Single Test File No Watch

Add the following configuration to launch.json

        {

            "name": "Run/Debug Open Test",

            "type": "node",

            "request": "launch",

            "runtimeExecutable": "${workspaceRoot}/node\_modules/.bin/jest.cmd",

            "args": [

                "--runInBand",

                "--watchAll=false",

                "${fileBasenameNoExtension}"

            ],

            "cwd" : "${workspaceFolder}",

            "protocol": "inspector",

            "console": "integratedTerminal",

            "internalConsoleOptions": "neverOpen"

        }

#### Debug/Run Single Test File With Watch

Add the following configuration to launch.json

        {

            "name": "Run/Debug Open Test",

            "type": "node",

            "request": "launch",

            "runtimeExecutable": "${workspaceRoot}/node\_modules/.bin/jest.cmd",

            "args": [

                "--runInBand",

                "--watchAll=true",

                "${fileBasenameNoExtension}"

            ],

            "cwd" : "${workspaceFolder}",

            "protocol": "inspector",

            "console": "integratedTerminal",

            "internalConsoleOptions": "neverOpen"

        }

#### CodeCoverage

npx jest --coverage

### React

The following sections assume the project was setup using

npx create-react-app my-app --template typescript

#### run all tests no watch

To use jest with typescript we need the following

npm install --save-dev jest

npm install --save-dev @types/jest

npm install --save-dev @babel/preset-typescript

We also need a file called babel.config.js

module.exports = {

    presets: [

      ['@babel/preset-env', {targets: {node: 'current'}}],

  +    '@babel/preset-typescript',

    ],

  };

Finally, we run the tests as follows in the terminal

npx jest

#### run all tests With watch

npx jest --watchAll

#### Run Single File Test No Watch

Run the test in hello2.test.ts Note we miss off the .ts from the filename

npx jest hello2.test

#### Run Single File Test Watch

npx jest hello2.test--watch

#### Debug/Run Single Test File No Watch

Add the following configuration to launch.json

        {

            "name": "Run/Debug Open Test",

            "type": "node",

            "request": "launch",

            "runtimeExecutable": "${workspaceRoot}/node\_modules/.bin/jest.cmd",

            "args": [

                "--runInBand",

                "--watchAll=false",

                "${fileBasenameNoExtension}"

            ],

            "cwd" : "${workspaceFolder}",

            "protocol": "inspector",

            "console": "integratedTerminal",

            "internalConsoleOptions": "neverOpen"

        }

#### Debug/Run Single Test File With Watch

Add the following configuration to launch.json

        {

            "name": "Run/Debug Open Test",

            "type": "node",

            "request": "launch",

            "runtimeExecutable": "${workspaceRoot}/node\_modules/.bin/jest.cmd",

            "args": [

                "--runInBand",

                "--watchAll=true",

                "${fileBasenameNoExtension}"

            ],

            "cwd" : "${workspaceFolder}",

            "protocol": "inspector",

            "console": "integratedTerminal",

            "internalConsoleOptions": "neverOpen"

        }

#### Code Coverage

npx jest --coverage

### React

This section assumes react with typescript was installed with

npx create-react-app my-app --template typescript

#### Run all tests with watch

Simply run the following from the VSCode terminal

npm run tests

#### Run Single test with watch

If the test file is called maths.test.js we enter the command on the terminal

npm run test maths.test

#### Debug single test