Getting Started with IBM Bluemix

Hands-On Workshop

Exercise 5b: Setting up Development Tooling

Setting up development tooling

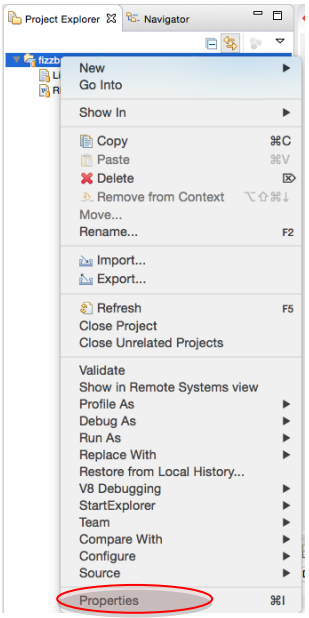
Using appropriate developer tooling with features to automatically check for errors, provide code assist, and so on can make you more productive and help identify possible problems.

You'll use the new project that you created in the previous section.

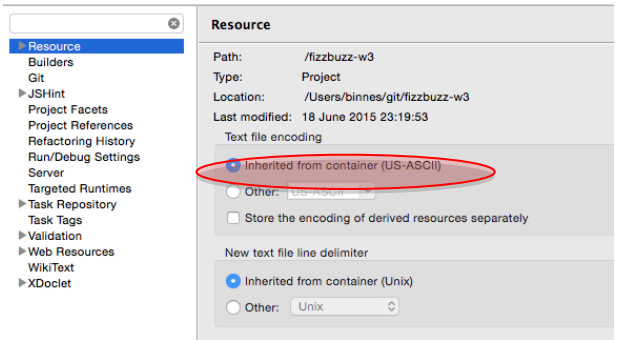
The first step is to ensure the JavaScript tooling is enabled and the test framework is created.

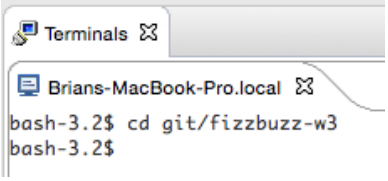
1. Create the JavaScript project:
   a. In the Terminals view, change to the directory that contains the project:
      - **Mac and Linux default**: `cd <user home>/git/<project name>`
      - **Windows default**: `cd <user home>\git\<project name>`

      If you want to verify where your project is located, right-click the project name in the Project Explorer view and then click **Properties**.



Note the project location on the **Resources** tab:





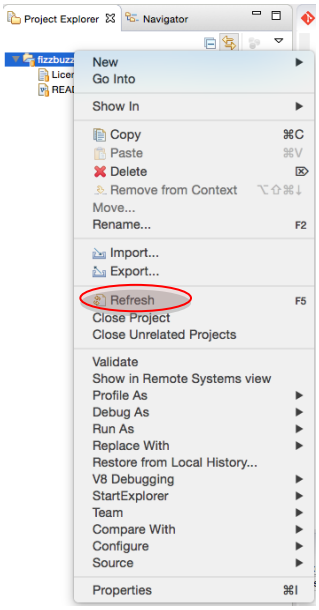
b. In the Terminals window, run the **`npm init`** command and answer the prompts as shown below.
   - Name: provide a unique name for your project
   - Version: `0.0.0`
   - Description: `REST API to calculate FizzBuzz value for a give range`
   - Entry point: `server.js`
   - Test command: `node_modules/.bin/mocha`
   - Git repository: press enter to accept the default value
   - Keywords: leave this blank
   - Author: enter your name
   - License: press enter to accept the default value

   Windows: Use the forward slash (/) when specifying the test command.

   If the Terminals window is too small, double-click the tab to enlarge it to a full screen. Double-clicking again will return it to its original size and location.

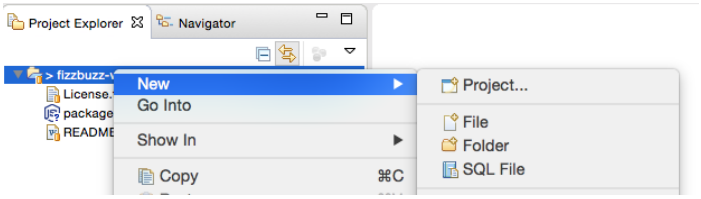
```
bash-3.2$ npm init
This utility will walk you through creating a package.json file.
It only covers the most common items, and tries to guess sane defaults.

See `npm help json` for definitive documentation on these fields
and exactly what they do.

Use `npm install <pkg> --save` afterwards to install a package and
save it as a dependency in the package.json file.

Press ^C at any time to quit.
name: (fizzbuzz-w3)
version: (0.0.0)
description: REST API to calculate FizzBuzz values for a given range
entry point: (index.js) server.js
test command: node_modules/.bin/mocha
git repository: (https://hub.jazz.net/git/binnes/fizzbuzz-w3)
keywords:
author: Brian Innes
license: (ISC)
About to write to /Users/binnes/git/fizzbuzz-w3/package.json:

{
  "name": "fizzbuzz-w3",
  "version": "0.0.0",
  "description": "REST API to calculate FizzBuzz values for a given range",
  "main": "server.js",
  "scripts": {
    "test": "node_modules/.bin/mocha"
  },
  "repository": {
    "type": "git",
    "url": "https://hub.jazz.net/git/binnes/fizzbuzz-w3"
  },
  "author": "Brian Innes",
  "license": "ISC"
}


Is this ok? (yes) yes
bash-3.2$
```

c. To move the created files into Eclipse, right-click the project name in the Project Explorer view and click **Refresh** or press F5.

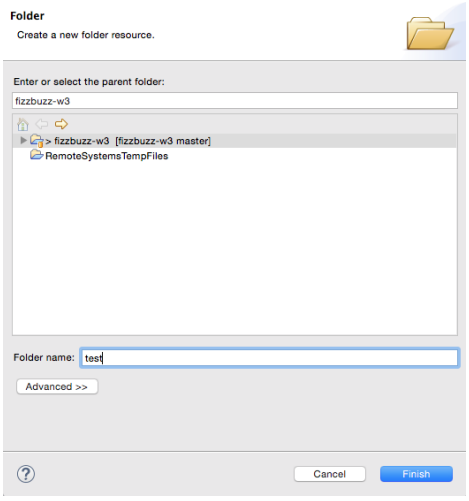


d. If the wizard truncated the test script to `mocha`, open the `package.json` file in the Eclipse editor and change it back to `node_modules/.bin/mocha`.

e. Tests are expected to be in a directory named `test`, so create a test directory:
   i. Right-click the project name in the Project Explorer view and click **New > Folder.**



   ii. Name the folder `test` and then click **Finish**.



f. To be able to run tests, install the Mocha framework:
   i. In the Terminals view, enter `npm install mocha --save-dev`

ii. Select the project name in the Project Explorer and click **Refresh** the project in Eclipse to display the new files.

g. Test that everything works as expected by running a test:
   i. In the Terminals view, enter `npm test`

```
bash-3.2$ npm test

> fizzbuzz-w3@0.0.0 test /Users/binnes/git/fizzbuzz-w3
> mocha



  0 passing (4ms)
```

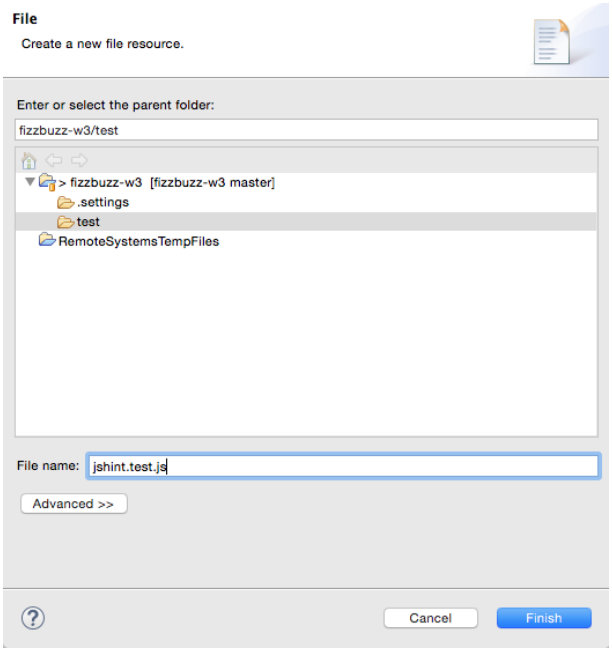
Note that the test framework ran even though there are no tests defined.

JSHint is a lint program for JavaScript. It provides static analysis of code. It's a good practice to have this running in your tooling. Many development teams also include it as part of their automated test suite.

h. Run JSHint from Mocha:
   i. In the Terminals view, enter the following command:

```
npm install mocha-jshint --save-dev
```

   ii. Select the test directory of the project in the Project Explorer view and then right-click and click **New > File**. Name the file `jshint.test.js`.



   iii. In the file, enter the following content and then save the file by clicking **File > Save** or use these commands:
   Windows and Linux: Ctrl+S
   Mac: Cmd+S:

```
require('mocha-jshint')();
```

iv. Rerun the tests by entering the following command in the Terminals view:

```
npm test
```

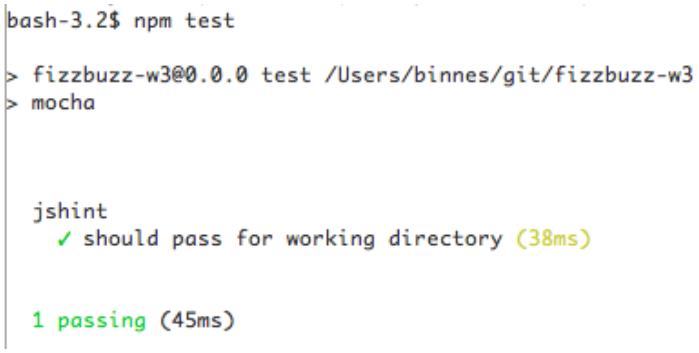
There are many errors from the imported modules in the `node_modules` directory. You do not control this is code, so it should be excluded from tests.

i. Configure JSHint to ignore the files by using a `.jshintignore` file:
   i. Create a `.jshintignore` file by right-clicking the project name in the Project Explorer view and clicking **New > File**.

   ii. Add the following content to the file:

```
node_modules
test
```

   iii. In the Terminals view, rerun the tests: `npm test`



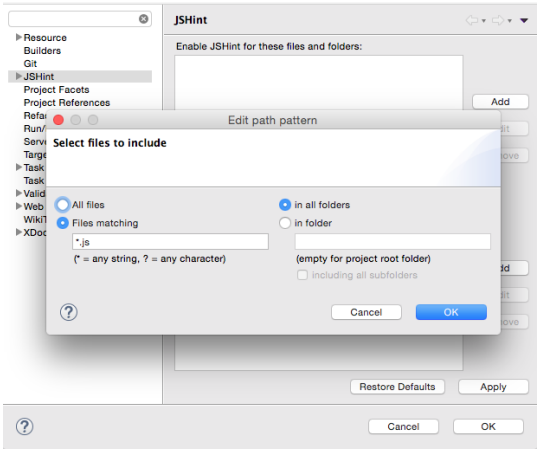
JSHint is now enabled in the test suite, but ideally the code editor should provide feedback from the problems.

JSHint is included in Eclipse, but you must configure it.

j. Enable JSHint in Eclipse:
   i. Right-click the project name in the Project Explorer view and click **Properties**.

   ii. In the dialog, click **JSHint**.

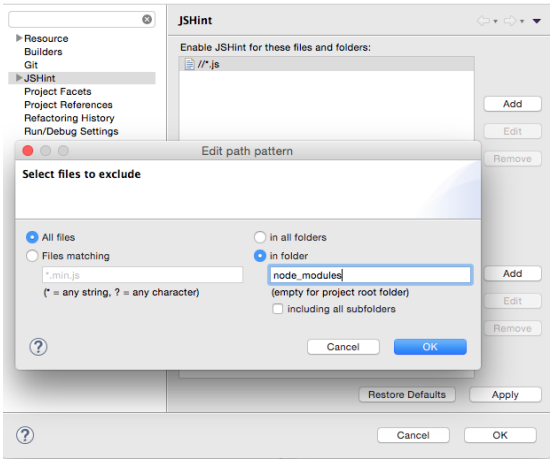
iii. Click **Add** next to the **Enable JSHint for these files and folders** box.



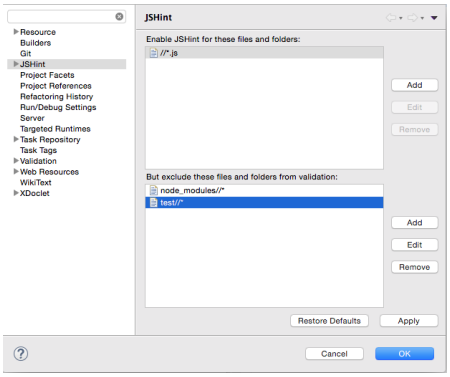
iv. Accept the defaults as shown above and click **OK**.

v. Click **Add** next to the **But exclude these files and folders from validation** box.

vi. Click **All files** and then select **in folder** and add the node_modules folder. Select the **include all subfolders** checkbox.



vii. Repeat the previous step to exclude the content of the test folder.



viii. Click **OK** to make the changes and close the dialog.

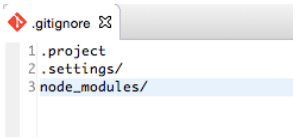
The project is now ready to implement the REST API, so now is a good time to commit changes to the Git repository (repo).

There is some debate about whether dependencies should be checked in or re-fetched at build time. To reduce the amount of traffic that you need to push to the Git repo for this exercise, you will exclude the dependencies in `node_modules`.

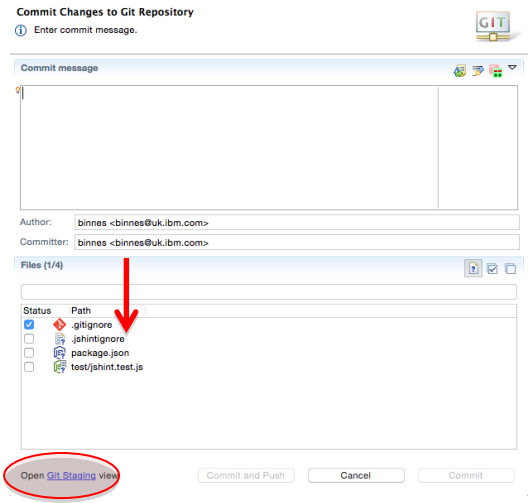
k. Update the Git configuration and then commit and push changes to the Git repo:

I. Switch to the Navigator view to see files that start with a period (.)

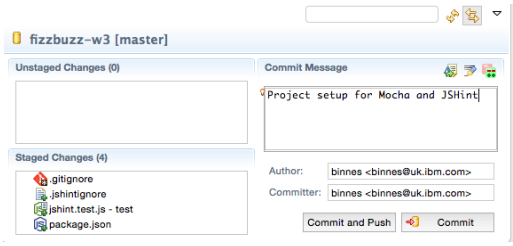
II. Double-click the `.gitignore` file to edit it and add the `node_modules` directory to the file. Then, save the file: Ctrl+S or Cmd+S.



III. Right-click the project name and click **Team > Commit**. Click the link at the bottom right of the dialog to open the Git Staging view.



IV. Drag the 4 files from the Unstaged Changes window to the Staged Changes window. Then, enter a commit message.



V. Click **Commit and Push**.

VI. Verify that the commit completed successfully and then close the confirmation dialog.