JavaScript – Jest

# Overview

Jest is a lightweight, quick, front-end JavaScript testing framework. One of the top features jest provides is test files will run cocurrently, which speeds up testing considerably.

Install

Jest can be added via the yarn package manager:

yarn add --dev jest

Add Jest script to the package.json at base level:

{

"scripts": {

"test": "jest"

}

The package.json "jest" key is where the general Jest configuration will be placed.

Adding an Example Simple Test

Create a JavaScript module to be tested:

// sum.js

function sum(a, b) {

return a + b;

}

module.exports = sum;

Create a Jest test for the module:

// sum.test.js

const sum = require('./sum');

test('adds 1 + 2 to equal 3', () => {

expect(sum(1, 2)).toBe(3);

});

Run tests using yarn:

yarn test

Setup in Ruby on Rails

To setup in Ruby on Rails with webpack, the correct Jest configuration must be added. First create a new folder to contain the Jest test scripts in spec/javascript, for example:

// spec/javascript/sum.test.js

Then configure the Jest root to be the new specs folder:

"jest": {

"roots": [

"spec/javascript"

]

}

The rails application javascript files are typically placed in the webpack packs location: app/javascript/packs. Add this location to the Jest module lookups:

"moduleDirectories": [

"node\_modules",

"<rootDir>/app/javascript/packs"

]

Pack files should now be requirable by name, instead of relative path from test.js file.

Rails applications often use javascript files with .erb extensions to allow for the rails engine to add values prior to test, add the .erb extension to the allowed list of extensions:

"moduleFileExtensions": [

"js",

"js.erb"

],

### Add in erb loader to transform .erb files with ruby engine prior to test

To mock out any static assets with the module name mapper, add the following to the Jest configuration:

"moduleNameMapper": {

"\\.(jpg|jpeg|png|gif|eot|otf|webp|svg|ttf|woff|woff2|mp4|webm|wav|mp3|m4a|aac|oga)$": "<rootDir>/spec/javascript/\_\_mocks\_\_/fileMock.js",

"\\.(css|less)$": "<rootDir>/spec/javascript/\_\_mocks\_\_/styleMock.js"

}

Then add the relevant mocks to the spec/javascript/\_\_mocks\_\_/ directory:

// spec/javascript/\_\_mocks\_\_/fileMock.js

module.exports = 'test-file-stub'

// spec/javascript/\_\_mocks\_\_/styleMock.js

module.exports = {}

Babel

For ECMAScript modules, Babel must be used to transpile the javascript files to an earlier version of javascript. Add babel to the project using yarn:

yarn add --dev babel-jest @babel/core @babel/preset-env

Create a babel config file in the root directory of the project if there is not one already present:

// babel.config.js

module.exports = {

presets: [

[

'@babel/preset-env',

{

targets: {

node: 'current',

},

},

],

],

}

Add babel transform to the jest configuration in package.json:

transform: {

"^.+\\.js(?:\\.erb)?$": "babel-jest"

},

Jest should now automatically transpile javascript files with babel if required.

Plugin - Testing Library

A common jest plugin is testing-library. It allows for more complex queries and event firing in Jest and JSDOM (useful for StimulusJS). The main package is the @testing-library/dom which allows for dom manipulation and querying.

<https://testing-library.com/>

Add the plugin to the project using:

yarn add --save-dev @testing-library/dom

Then import the required components in the specs:

import { render, fireEvent, waitFor, screen } from '@testing-library/dom'

import { render, fireEvent, waitFor, screen } from '@testing-library/react'

## FireEvent

Fire a custom event on a dom element by selecting the node then using the fireEvent method:

fireEvent[eventName](node: HTMLElement, eventProperties: Object)

For example:

fireEvent.input(sliderInput, { target: { value: 25 } })

The full list of events which can be fired can be seen at: <https://github.com/testing-library/dom-testing-library/blob/master/src/event-map.js>

Integration Testing - Databases

## General

One of the top features jest provides is running each test file cocurrently, with the tests in each file being run sequentially, which speeds up testing considerably on multicore machines. However, this concurrency complexifies integration tests which interact with the database. When tests are run cocurrently, entries and queries to the database which are performed in by different workers may interact in different ways depending on the speed of each worker, leading to errors and unstable tests.

There are two ways to fix this issue:

* run the tests serially with the --runInBand option, causing a large performance hit
* dynamically generate separate test databases for each worker to avoid any interaction, performance hit should be minimal on small databases

## Dynamically Create Databases

Use the project database as a template for the test databases since this should be migrated. Set the ORM to delete any exisiting database with the dynamic name and create a new one from the project database. Then pass database connection using dependency injection through the application. The database can then be mocked as required and database connections are kept to a minimum allowing for scalability.

Sequelize setup:

if (env === 'test') {

module.exports = (async () => {

// connect to template database and migrate

const templateConnection = new Sequelize(config.database, config.username, config.password, config)

// generate new test db for worker

const workerDatabaseName = config.database + '-test-' + process.env.JEST\_WORKER\_ID

await templateConnection.query(`DROP DATABASE IF EXISTS "${workerDatabaseName}";`)

await templateConnection.query(`CREATE DATABASE "${workerDatabaseName}" TEMPLATE "${config.database}";`)

await templateConnection.close()

sequelize = new Sequelize(workerDatabaseName, config.username, config.password, config)

// add model to db object

fs.readdirSync(\_\_dirname)

.filter(file => {

return (file.indexOf('.') !== 0) && (file !== basename) && (file.slice(-3) === '.js')

})

.forEach(file => {

const model = require(path.join(\_\_dirname, file))(sequelize, Sequelize.DataTypes)

db[model.name] = model

})

Object.keys(db).forEach(modelName => {

if (db[modelName].associate) {

db[modelName].associate(db)

}

})

db.sequelize = sequelize

db.Sequelize = Sequelize

return db

})()

Server setup:

const debug = require('debug')('express:server')

const database = require('./app/models')

const app = require('./app')(database)

const port = process.env.PORT || 5000

const server = app.listen(port, () => {

debug(`Starting server in ${process.env.NODE\_ENV} mode`)

debug('Server is listening on port ' + port)

})