Jest (testing)

Monday, March 5, 2018   11:02 AM

Installation / basics

Installation:

- Install dev dependency packages. For me, these were
    - babel-jest
    - eslint
    - eslint-plugin-jest
    - jest
    - If you didn't already have eslint:
        - eslint-plugin-import
    - If you don't already have it, Babel itself:
        - babel-plugin-transform-object-rest-spread
        - babel-preset-env
- I didn't have a babelrc, so I needed to make one. Also, because I export already-transpiled ES6 classes in @botland/shared and ran into issues originally with trying to extend those classes, so I needed to add "exclude": ["transform-es2015-classes"] as shown below:

```
{
    "plugins": [
        // "syntax-dynamic-import",
        "transform-object-rest-spread",
    ],
    "compact": false,
    "presets": [
        ["env", {
            // Try to take advantage of Webpack's scope-hoisting by not
            // transpiling modules in Babel.
            "modules": false,

            // This is needed so that I don't run into an issue on the client
            // with trying to create an already-transpiled class like
            // RestClient.
            "exclude": ["transform-es2015-classes"],

            // Use polyfills for something like Promise or Map for browsers like
            // IE. Note that this is an older version of babel-preset-env
            // (1.6.1), so "true" is indeed the correct argument rather than
            // "usage" or "entry", and core-js needs to be installed explicitly
            // for it to work due to using NPM2.
            "useBuiltIns": true,
            "targets": {
                "browsers": ["last 2 versions", "safari >= 7", "ie > 8"],
            },
        }],
        "react",
    ],
    "env": {
        "test": {
            "presets": [
                ["env", {
                    "exclude": ["transform-es2015-classes"],
                }],
                "react"
            ]
        }
    }
}
```

    - Note that I am not on Babel 7, so apparently I can't use browserslistrc and thus need to manually list out the browsers that I'm targeting.
    - Because I didn't have a babelrc already but I *did* have a webpack.config.js, I needed to remove my babel configuration from webpack.
- I needed a jest.config.js for ignoring LESS files (see below)
- Configure .eslintrc.json
    - Add "jest" to "plugins"
    - Add "plugin:jest/recommended" to "extends"
- Modified my client/package.json to add some scripts
    - "test": "jest"
    - "test:coverage": "jest --coverage && start ./coverage/lcov-report/index.html",

    - "test:watch": "jest --watch",
    - "test:debug": "node --inspect-brk ./node_modules/.bin/jest --runInBand --watch",
    - "test:debug:win": "node --inspect-brk ./node_modules/jest/bin/jest.js --runInBand --watch"
    - Note: the test:debug:win version actually works on all platforms.
- Consider installing jest-watch-typeahead (reference)
    - npm install --save-dev jest-watch-typeahead
    - If you're going to install this, you also need to modify your jest.config.js.
    - This also involves being on at least Jest v23, so you may need to do "npm install --save-dev jest@latest".
    - Once done, you can run "jest --watch", then press enter to cut off any future tests, and finally follow the filter help that shows.

Helpful setup guide (reference)
HiDeoo wrote up a nice starting guide for me (in the reference link above).

Testing using Jest in general (reference)
Typically, you want to write your unit tests before you move on to anything scenario- or integration-related. When you DO get to that point, if you have thunks dispatching other thunks, you can use FlushThunks from redux-testkit so that you can wait for everything to finish before checking the state.

jest.config.js and "extending" a parent config
The file is just a JavaScript file, so you can do something like this:
```
const baseConfig = require('../../jest.config.js');

module.exports = {
  ...baseConfig,
  // Override anything here that you want to specify
};
```

rejects (reference)
THIS IS NOT A FUNCTION. It is just ".rejects.toThrow()" or "toThrowErrorMatchingSnapshot":
```
await expect(
  dbFramework.sellItem(userId, cosmeticItemIdYouDontOwn, sellAmount)
).rejects.toThrow();
```

So if you have a function that you *would* have called with "await someFunction", you do what you see above:
```
await expect(someFunction).rejects.toThrowErrorMatchingSnapshot();
```

Async "expects" (reference)
Suppose you have a function that you know should fail and you care about its error message:
```
await expect(
  dbFramework.testSetLeaguesForSeason(leagueInfo)
).rejects.toMatchSnapshot();
```

You can do this with toThrowErrorMatchingSnapshot as well (I prefer this way):
```
await expect(
  dbFramework.testSetLeaguesForSeason(leagueInfo)
).rejects.toThrowErrorMatchingSnapshot();
```

You should *not* need expect.assertions unless you're using callbacks, but if you're using callbacks, then you have more to fix than just adding "expect.assertions".

Jest extensions (reference)

If you ever want extra matchers like "toBeArray", you can look at this package.

Mocks/mocking

See this note.

Debugging (reference)

Because I'm on Windows and I have an old version of Node, here's the command I have to run:

```
node --inspect-brk ./node_modules/jest/bin/jest.js --runInBand
```

If this doesn't work for you, then it could require an update to at least Node v8.4 (see issue #1652)

TIMEOUT ISSUE: keep in mind that while debugging, the Jest default timeout per test of 5000 ms (reference) could be hit, and it may not be obvious that it's happening in that case. To adjust the timeout of a test, the signature is "test(name, fn, timeout)", so just put 1e9 as the timeout.

Make sure not to check in the changed timeout!

I highly suggest reading on to "Sourcemaps" to find out how to get sourcemaps working, otherwise debugging could be very tough.

Sourcemaps

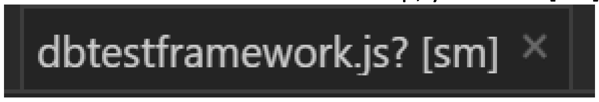
Turns out you don't need to specify sourceMaps yourself in your babelrc because babel-jest will inline them *for* you (but if you *did* have to, here's what it would look like):

```
{
  "env": {
    "test": {
      "presets": [
        ["env", {
          "exclude": ["transform-es2015-classes"],
        }],
      ],
      "sourceMaps": "inline"
    }
  }
}
```

However, this can lead to strange issues debugging. For example, I had ES6 (which I wrote) that looked like this:

```
initialize() {
  return this.setupDatabase().then(() => {
    return this.databaseWrapper.connect();
  });
}
```

However, the ES5 that it transpiled really turned "this" into "_this4", so it was nearly impossible for me to actually debug using the console (due to this Chromium issue). Ways around this behavior:

1. Turn off sourcemaps in Chrome (and refresh the page) so that you're looking at transpiled code (reference)
    a. Note: when code has a sourcemap, you'll see [sm] in the tab:
    b. 
    dbtestframework.js? [sm]
2. For this specific problem, arrow functions shouldn't even be transpiled since they're supported natively by Node, so this was a configuration problem. I needed to make sure babel-preset-env knew to target the current version of Node:

```
"env": {
  "test": {
    "presets": [
      ["env", {
        "exclude": ["transform-es2015-classes"],

        targets: {
          node: 'current',
        },
      }],
    ],
    "sourceMaps": false,
  }
}
```

From Lumie1337: @Adam13531 regarding the sourcemaps issue, apparently it is an open issue for chrome, tldr: mapping from source code without source maps to source mapped symbols works, but not the other way around https://bugs.chromium.org/p/chromium/issues/detail?id=327092

~~Custom "expect" logic (reference)~~

~~I was testing Joi and noticed that it was emitting "then" and "catch" properties everywhere even though they weren't relevant to my tests. Originally, I just called "delete" on both of those, but then HiDeoo showed me the reference link, and now I can do something like this:~~

> 14:28 HiDeoo: Ho nvm Adam13531, I remembered it wrong, it's not yet doable as-is in an expect.extend, we use a fork to have toMatchSnapshotAndIgnoreKeys() but they don't expose the original toMatchSnapshot() in expect.extend so we had to fork it to expose it. There is a PR coming up for this to expose it but not yet merged.

~~Note: if you're going to go this route, the location where you'd add the code would be based on setupTestFrameworkScriptFile in your jest.config.js (reference). I wrote a config like this:~~

```
~~module.exports = {~~
~~ setupTestFrameworkScriptFile: '<rootDir>/test/setuptestframework.js',~~
~~};~~
```

Coverage

All I had to do is put this in my package.json file under "scripts":

```
"test:coverage": "jest --coverage"
```

Then I did "npm run test:coverage" and got a coverage/lcov-report/index.html that I could open.

Collecting coverage from other files (reference)

All you have to do is make a jest.config.js that looks like this:

```
module.exports = {
  collectCoverageFrom: ['<rootDir>/whatever/**/*.js'],
};
```

Keep in mind that you can't cover a file in node_modules via any combination of options since Istanbul is just going to ignore all of node_modules for you in the end anyway (reference).

Using snapshots

When running a test with "toMatchSnapshot" for the first time, a __snapshots__ directory is going to get created alongside the test. This will contain some JavaScript objects/strings that represent what was returned by your test.

Subsequently, it will compare the output of the test against whatever's in the snapshot files. Suppose you run a test and it reports a mismatch against the snapshot (meaning the test has errors), but you don't want them to be considered as errors, you can press 'u' in "jest --watch" to update the snapshot files.

You should check in your __snapshots__ folders.

If you expect your test to fail, then instead of saying something like:

```
expect(passingFunction()).toMatchSnapshot()
```

say

```
expect(() => failingFunction()).toThrowErrorMatchingSnapshot();
```

Inline snapshots

To get snapshots directly into your test code without needing a new file, take a look at this:

> [13:53] HiDeoo: Btw adam13531 I don't know what Jest version you're using, but in 23.3 they introduced inline snapshots with toMatchInlineSnapshot() & toThrowErrorMatchingInlineSnapshot() and it's amazing, no more snapshot files & you can see the snapshot right from the test https://bit.ly/2mFmqs6

Dynamic data in snapshots ("property matchers") (reference)

If you have a generated user ID or date, you may only want to check that the type is correct. This is where property matchers come into play. However, there may be a time when you want to ignore a property altogether (e.g. a database property that could be null *or* a date); in those cases, you're not really testing anything (since the database will enforce that constraint for you just by virtue of columns having types), so just omit the property from the test:

```
    const dbSeasonLeaguesJestMatcher = {
      league_id: expect.any(Number),
      season_id: expect.any(Number),
    };

    _.map(dbSeasonLeagues, (dbSeasonLeague) => {
      delete dbSeasonLeague.created_at; // this could be null or a date,
so just delete it so that it doesn't end up in the snapshot

expect(dbSeasonLeague).toMatchSnapshot(dbSeasonLeaguesJestMatcher);
    });
```

Ignoring LESS files (reference)

The reference clearly describes two ways. If you want to go the config route, put it into jest.config.js:

```
module.exports = {
    "moduleNameMapper": {
        ".*\\.less$": "<rootDir>/pathToDummyFile/dummy.js"
    }
};
```

Note that <rootDir> is *not* just for the sake of example. Also, you'll need to restart Jest after doing this.

Troubleshooting

Babelrc issues

I am pretty sure I ran into this issue. It was frustrating enough where I just explicitly put my babelrc back into Webpack for the sake of deploys and kept a separate .babelrc file just for sake of Jest.

If I ever need to take another look at this, I could specify "debug: true" in babelrc so that I can figure out which plug-ins are being used.

console.time

There's a bug in console.time at the time of writing (6/12/2018) at these two lines of code: they divide by 1000 but then show the resulting time in ms. This means that if you use console.time and your times seem off by a factor 1000 that it's not you. ;)

UPDATE (6/20/2018): this has been fixed

Random failure that you can't chalk up to anything else

It's possible that you forgot to reset mocks. If running the test in isolation works, then this is a stronger possibility since it means you didn't clean up after a previous test.