# Jest

# Jest Overview

- "a delightful JavaScript Testing Framework with a focus on simplicity"
  - https://jestjs.io/
- Supports unit tests
- "Automatically finds tests to execute in your repo"
  - by default, all `.test.js` and `.spec.js` files in project and all `.js` files in and under `__tests__` directory

    also `.jsx` files
- "Runs your tests with a fake DOM implementation (via jsdom) so that your tests can run on the command line"
- Watches source and test files and automatically reruns tests when they change
  - can run all tests or only ones that failed in last run
- Support snapshot tests
- Can use to test React components
- Default test framework of apps created with create-react-app

# Jest API Highlights

- **`describe(name, fn)`**
  - describes a "test suite"
- **`describe.only(name, fn)`**
  - alias `fdescribe`
- **`describe.skip(name, fn)`**
  - alias `xdescribe`
- **`beforeAll(fn)`**
  - run once before all tests in suite begin
- **`beforeEach(fn)`**
  - run before each test in suite begins
- **`afterEach(fn)`**
  - run after each test in suite begins
- **`afterAll(fn)`**
  - run once after all tests in suite finish

`test` calls are not required to be inside a `describe`

- **`test(name, fn)`**
  - alias `it`
- **`test.only(name, fn)`**
  - alias `fit`
- **`test.skip(name, fn)`**
  - alias `xit`

- **`expect(value)`**
  - chain a matcher call onto this
  - see next two slides

# Jest Matchers ...

https://jestjs.io/docs/en/using-matchers

- **.not** - can be prepended to any matcher
- **.toBe(*value*)** - uses ===
- **.toEqual(*value*)**
  - deep object comparison
- **.toBeTruthy()**
- **.toBeFalsy()**
- **.toBeDefined()**
- **.toBeUndefined()**
- **.toBeNull()**
- **.toMatch(*regexp*)**
- **.toMatchObject(*object*)**
  - all properties in ***object*** match those in receiver
- **.toBeCloseTo(*number, digits*)**
- **.toBeGreaterThan(*number*)**
- **.toBeGreaterThanOrEqual(*number*)**
- **.toBeLessThan(*number*)**
- **.toBeLessThanOrEqual(*number*)**
- **.toBeInstanceOf(*Class*)**
- **.toContain(*item*)**
- **.toContainEqual(*item*)**
  - deep object comparison
- **.toHaveLength(*number*)**

# ... Jest Matchers

- `.toHaveBeenCalled()`
  - alias is `.toBeCalled`
- `.toHaveBeenCalledTimes(number)`
- `.toHaveBeenCalledWith(arg1, arg2, ...)`
  - alias is `.toBeCalledWith`
- `.toHaveBeenLastCalledWith(arg1, arg2, ...)`
  - alias is `.lastCalledWith`

- `.toThrow()`
- `.toThrowError(error)`

- `.toMatchSnapshot()`
- `.toThrowErrorMatchingSnapshot()`

# Jest Snapshot Testing

- Snapshot tests assert that ...
  - a component will render same content as last successful test
- The first time snapshot tests are run ...
  - `toMatchSnapshot` matchers save a representation of rendered output in subdirectory of test file named `__snapshots__`
    - mostly, but not exactly HTML
- In subsequent runs ...
  - same representation is generated again and compared to what was saved in last successful run
- When snapshot tests fail ...
  - scroll back to review differences in rendered output
  - if changes are correct, press "u" to accept them
    - overwrites previous snapshot files with new ones
  - if changes are incorrect, fix code and run tests again

These `__snapshot__` directories should be checked into version control!

# Writing Snapshot Tests

- The long way ...

```
import renderer from 'react-test-renderer';

describe('some suite') {
  test('should do something') {
    const component = some-jsx;
    const tree = renderer.create(component);
    expect(tree.toJSON()).toMatchSnapshot();
  }
}
```

- The short way using our utility function ...

```
import snapshot from '../../../snapshot';

describe('some suite') {
  test('should do something') {
    snapshot(some-jsx);
  }
}
```

# Failing Snapshot Tests

- Failing tests prevent git pushes due to our use of **husky** and the **`prepush`** npm script
- When some of failing tests are snapshot tests ...
  - run "**`npm t`**"
  - press "**`a`**" or press enter to run all tests
  - scroll back to examine snapshot differences
  - if new output is correct, press "**`u`**" to update snapshot files
  - press "**`q`**" to exit
- Once all tests are passing, **`git push`** will work

# Can Test React Components

- Example files see https://github.com/mvolkmann/react-examples/tree/master/jest
  - **`webpack.config.js`** - configures webpack to use Babel (ES6 to ES5 transpilation) and ESLint (JavaScript linting) loaders
  - **`.babelrc`** - configures Babel "preset" configurations to be used
  - **`package.json`** - used by npm; specifies dependencies, defines test script, and configures jest
  - **`jest/preprocessor.js`** - required in order to test code that uses JSX
  - **`src/greeting.js`** - defines a React component to be tested
  - **`__tests__/greeting-test.js`** - defines tests
  - **`__tests__/dont-mock.js`** - work-around for import hoisting issue (see comments in test code)
- Steps to run tests
  - **`npm install`**
  - **`npm test`** - runs **`jest`** command

# webpack.config.js

- webpack is a JavaScript module bundler
- It follows dependencies expressed with
  ES 2015 imports, AMD, and CommonJS (require)
  to produce a single JavaScript file that combines many

```
module.exports = {
  entry: './src/demo.js',
  output: {
    path: __dirname,
    filename: 'build/bundle.js'
  },
  module: {
    loaders: [
     {test: /\.js$/, exclude: /node_modules/, loader: 'babel!eslint'},

    ]
  },
  resolve: {
    root: '.'
  }
};
```

uses the component being tested

# .babelrc

- Babel is a JavaScript transpiler
- It can be configured to
  transpile ES 2015 code to ES5
  and convert JSX to JavaScript functions (React)

```
{
  "presets": ["env", "react"]
}
```

# package.json

- Used by npm
- Specifies dependencies, defines **`test`** script, and configures Jest

```
{
  "name": "jest-demo",
  "version": "1.0.0",
  "description": "This demonstrates using jest to test React components.",
  "main": "index.html",
  "scripts": {
    "test": "jest"
  },
  "devDependencies": {
    "babel-core": "*",
    "babel-jest": "*",
    "babel-loader": "*",
    "babel-preset-env": "*",
    "babel-preset-react": "*",
    "eslint": "*",
    "eslint-loader": "*",
    "eslint-plugin-react": "*",
    "jest-cli": "*",
    "jest-webpack-alias": "*",
    "react-addons-test-utils": "*",
    "webpack": "*",
    "webpack-dev-server": "*"
  },
  "dependencies": {
    "react": "*",
    "react-dom": "*"
  },
  "jest": {
    "scriptPreprocessor": "jest/preprocessor.js",
    "unmockedModulePathPatterns": [
      "node_modules/react",
      "node_modules/react-dom",
      "node_modules/react-addons-test-utils",
      "node_modules/fbjs"
    ]
  }
}
```

Warning: Jest **`verbose`** mode can overwrite some **`console.log`** output!

don't mock React libraries or the fbjs utility library that React uses

# preprocessor.js

- Required in order to test code that uses JSX

```
const babelJest = require('babel-jest');
const webpackAlias = require('jest-webpack-alias');

module.exports = {
  process(src, filename) {
    if (filename.indexOf('node_modules') === -1) {
      src = babelJest.process(src, filename);
      src = webpackAlias.process(src, filename);
    }
    return src;
  }
};
```

processes all JavaScript files except those below the `node_modules` directory

# greeting.js

- The
  React component
  to be tested

```
import React from 'react';

class Greeting extends React.Component {
  constructor(props) {
    super(props);
    this.state = {name: 'World'}; // initial state
  }

  setName(event) {
    this.setState({name: event.target.value});
  }

  render() {
    return (
      <form>
        <div>
          <label>Name: </label>
          <input type="text" value={this.state.name}
            onChange={e => this.setName(e)}/>
        </div>
        <div>
          {this.props.greet}, {this.state.name}!
        </div>
      </form>
    );
  }
}

Greeting.defaultProps = {greet: 'Hello'};
```

# greeting-test.js ...

```
/* global describe, expect, it */

// All ES6 imports get hoisted to the top,
// so they get mocked before the line below runs!
//jest.dontMock('../src/greeting');
// This line was moved to dont-mock.js so it can be imported first.
// See explanation at https://github.com/babel/babel-jest/issues/16.
import './dont-mock';

import React from 'react';
import ReactDOM from 'react-dom';
import TestUtils from 'react-addons-test-utils';

import Greeting from '../src/greeting';

describe('greeting', () => {
  test('greets with defaults', () => {
    const component = TestUtils.renderIntoDocument(
      <Greeting/>
    );
    const form = ReactDOM.findDOMNode(component);
    const lastDiv = form.lastChild;
    expect(lastDiv.textContent).toBe('Hello, World!');
  });
```

This is the part to focus on. Notice how easy it is to write the actual test code!

# ... greeting-test.js

```
  test('greets with prop', () => {
    const component = TestUtils.renderIntoDocument(
      <Greeting greet="Hola"/>
    );
    const form = ReactDOM.findDOMNode(component);
    const lastDiv = form.lastChild;
    expect(lastDiv.textContent).toBe('Hola, World!');
  });

  test('greets with state', () => {
    const component = TestUtils.renderIntoDocument(
      <Greeting/>
    );
    component.setState({name: 'Mark'});
    const form = ReactDOM.findDOMNode(component);
    const lastDiv = form.lastChild;
    expect(lastDiv.textContent).toBe('Hello, Mark!');
  });
});
```

# Async Tests

- Function passed to tests of asynchronous functions should have a **`done`** parameter
- When all asynchronous actions have completed, call **`done()`**
- Can fail a test by calling **`done.fail(error)`**
- Ex.

```
try {
  expect(actual).toBe(expected);
  done();
} catch (e) {
  done.fail(e);
}
```

# no-mock.js

- Prevents the code being tested from being mocked

```
/* global jest */
jest.dontMock('../src/greeting');
```

# Running Tests

- Enter "npm test"
- Output

```
> jest-demo@1.0.0 test /Users/Mark/Documents/training/React/react-examples/jest
> jest

Using Jest CLI v0.8.0, jasmine1
 PASS  __tests__/dont-mock.js (0.339s)
 PASS  __tests__/greeting-test.js (0.753s)
1 test passed (1 total in 2 test suites, run time 1.668s)
```

# More Details

- Errors in tests
  - if code inside a test throws, the test fails
  - can be used as an alternative to explicit assertions
- Testing functions that return promises
  - if a test returns a promise,
    it will wait for the promise to resolve or reject
    and fail if it rejects
- Test data
  - consider using faker.js npm package to generate
  - https://github.com/Marak/Faker.js