**React TDD Example: Unit Testing and Building a React Component With Jest, Gulp and React Test Utils**

ReactJs has attained lot of momentum since its initial release in 2013, and became the best JavaScript library for developing rich UI interface. React also popularised different concepts like Virtual DOM, Uni-directional data flow and Componentization in the front-end community. As of today, React has more than 20,000 stars in [Github repo](https://github.com/facebook/react), and actively maintained by the Facebook team.

As I said already, React is a solid piece of work from Facebook. Any solid software will be built with testability in mind; And React is not an exception to that. Facebook built React with complete testing support — They even released their own testing framework named “**Jest**” and **React Test Utils** for unit testing React components.

Here in this post, we’ll see how to build a react component in TDD approach.

**What we’re going to build ?**

We will build a simple Accordion component in React. If you’re not sure what an Accordion is, have a look at the twitter bootstrap page:

[Accordion Widget](http://getbootstrap.com/2.3.2/javascript.html#collapse) Or, see [jQuery UI Accordion](https://jqueryui.com/accordion/)

We’ll be creating a simple Accordion component in React.

**Setting up the tools**

We’re going to build our component in TDD fashion. Hence, we need to set up our environment with a test runner (for running unit tests), and local server ( with automatic reloading enabled ).

These are the tools I am going to use:

1. [BrowserSync](http://www.undefinednull.com/2015/05/03/react-tdd-example-unit-testing-and-building-a-react-component-with-jest-gulp-and-react-test-utils/www.browsersync.io) for live development ( browser reloading ).
2. [Jest](https://facebook.github.io/jest/) for unit testing.
3. [JSPM](http://jspm.io/) and [npm](https://www.npmjs.org/) for package management.
4. [Gulp](http://gulpjs.com/) for running build and test tasks.
5. Finally, [ReactJS](https://facebook.github.io/react/) for developing our component

Setting up all these takes time. Therefore, I have created a [seed project](https://github.com/shidhincr/react-jest-gulp-jspm-seed) for bootstrapping the set-up and the initial boiler plate code. So, as the first step, clone the seed project to your system:

|  |  |
| --- | --- |
| 1  2  3 | git clone git@github.com:shidhincr/react-jest-gulp-jspm-seed.git react-seed  cd react-seed  npm install |

Once you download the seed and install the node modules, we’re good to start.

**Developing the Component in TDD**

We’re using *Gulp* as our build and task tool. If you open the **gulpfile.js** in the seed folder, you can see there are two main tasks for test driven development ( as explained in the project README file ). One task will build the project and open the development server in browser, while the other task run the unit tests and watch for any changes the source files and test files.

Open two terminal windows, and run **gulp develop** in one and **gulp test** in the other one.

There are two example files provided in the **scripts** folder and **\_\_tests\_\_** folder. When you run the **gulp test** task at first, you should see those tests passing.

**Accordion React Component**

Let’s add the file **accordionComponent.js** in **scripts** and **accordionComponent-spec.js** in the **\_\_tests\_\_** folders respectively.

From here onwards, we’re going to build our component by writing each functionality as a test case and then implementing them. When adding the test case, you should be able to see the tests failing in the terminal ( gulp test ). Hence, once we see the failing test, the next step is to implement the code to make them pass.

**Test Case 1:**

*The accordion should exist as a React Component*

Write the first test case in **accordionComponent-spec.js**

accordionComponent-spec.js

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19 | jest.dontMock('../scripts/accordionComponent.js');  describe('Accordion', function() {  var React = require('react');  var ReactDOM = require('react-dom');  var TestUtils = require('react-addons-test-utils');  var Accordion;  beforeEach(function() {  Accordion = require('../scripts/accordionComponent');  });  it('should exists', function() {  // Render into document  var accordion = TestUtils.renderIntoDocument( <Accordion /> );  expect(TestUtils.isCompositeComponent(accordion)).toBeTruthy();  });  }); |

Note: By using **jest.dontMock**, we get the original react component in the scripts folder ( as Jest tries to mock every component by default ).

Let’s make the test pass:

accordionComponent.js

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13 | 'use strict';  var React = require('react');  var Accordion = React.createClass({  render: function(){  return (  <div>Accordion component</div>  );  }  });  module.exports = Accordion; |

Okay, the tests are passing now. Let’s also make sure that our component renders properly in the browser. Modify the **app.js** to render the **accordionComponent**.

app.js

|  |  |
| --- | --- |
| 1  2  3  4  5  6 | 'use strict';  var React = require('react');  var ReactDOM = require('react-dom');  var Accordion = require('components/accordionComponent');  ReactDOM.render(<Accordion/>, document.getElementById('view')); |

If it works, proceed to next test case.

**Test Case 2:**

*Accordion should build the layout from an array of objects passed as prop.*

This time let’s make our component configurable. We have to pass an array data to the component, and expect it to build the layout for the Accordion.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23 | it('should build the layout from an array of objects passed as prop', function() {  var input = [{  name: 'Title 1',  content: 'Content belongs to title 1'  },{  name: 'Title 2',  content: 'Content belongs to title 2'  }];  var accordion = TestUtils.renderIntoDocument( <Accordion data={input}/> );  var headers = TestUtils.scryRenderedDOMComponentsWithClass(accordion, 'accordion-header');  var contents = TestUtils.scryRenderedDOMComponentsWithClass(accordion, 'accordion-content');  expect(headers.length).toEqual(2);  expect(contents.length).toEqual(2);  expect(headers[0].textContent).toEqual('Title 1');  expect(headers[1].textContent).toEqual('Title 2');  expect(contents[0].textContent).toEqual('Content belongs to title 1');  expect(contents[1].textContent).toEqual('Content belongs to title 2');  }); |

To make everything green:

accordionComponent.js

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31 | 'use strict';  var React = require('react');  var Accordion = React.createClass({  render: function(){  var panes = [];  var data = this.props.data || [];  data.forEach(function(item, index){  panes.push(  <div key={index}>  <div className="accordion-header">  {item.name}  </div>  <div className="accordion-content">  {item.content}  </div>  </div>  );  });  return (  <div className="accordion">  {panes}  </div>  );  }  });  module.exports = Accordion; |

All the tests pass now. To see the component works in browser, edit the **app.js** and pass the configuration:

app.js

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14 | 'use strict';  var React = require('react');  var ReactDOM = require('react-dom');  var Accordion = require('components/accordionComponent');  var input = [{  name: 'Title 1',  content: 'Content belongs to title 1'  },{  name: 'Title 2',  content: 'Content belongs to title 2'  }];  ReactDOM.render(<Accordion data={input}/>, document.getElementById('view')); |

You should be able to see that our Accordion component renders the new layout.

**Test Case 3:**

*By default, all the content section should be hidden unless specified by a flag.*

Add the following test case:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15 | it('should hide all the contents by default unless specified by a flag', function(){  var input = [{  name: 'Title 1',  content: 'Content belongs to title 1'  },{  name: 'Title 2',  content: 'Content belongs to title 2',  showOnLoad: true  }];  var accordion = TestUtils.renderIntoDocument( <Accordion data={input}/> );  var contents = TestUtils.scryRenderedDOMComponentsWithClass(accordion, 'accordion-content');  expect(contents.length).toEqual(1);  expect(contents[0].textContent).toEqual('Content belongs to title 2');  }); |

And make it pass:

accordionComponent.js

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37 | 'use strict';  var React = require('react');  var Accordion = React.createClass({  render: function(){  var panes = [];  var data = this.props.data || [];  var getContent = function(item){  return !item.showOnLoad ? '': (  <div className="accordion-content">  {item.content}  </div>  );  };  data.forEach(function(item, index){  panes.push(  <div key={index}>  <div className="accordion-header">  {item.name}  </div>  { getContent(item) }  </div>  );  });  return (  <div className="accordion">  {panes}  </div>  );  }  });  module.exports = Accordion; |

This code will make the above test case pass, but you can see that it makes the previous test case ( Test case 2 ) fail.

This is not a problem at all, because in TDD this is what we expect. We need to continuously refactor both source and test codes while we build the component. Let’s modify the **input** in the previous test case to make the tests pass.

TestCase2

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9 | var input = [{  name: 'Title 1',  content: 'Content belongs to title 1',  showOnLoad: true  },{  name: 'Title 2',  content: 'Content belongs to title 2',  showOnLoad: true  }]; |

This is the beauty of TDD; designing the component step by step. By now, we made our component configurable also.

**Test Case 4:**

*It should be able to toggle the content by clicking on the respective title.*

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22 | it('should be able to toggle the content by clicking on the respective title.', function(){  var input = [{  name: 'Title 1',  content: 'Content belongs to title 1'  },{  name: 'Title 2',  content: 'Content belongs to title 2'  }];  var accordion = TestUtils.renderIntoDocument( <Accordion data={input}/> );  var headers = TestUtils.scryRenderedDOMComponentsWithClass(accordion, 'accordion-header');  var contents = TestUtils.scryRenderedDOMComponentsWithClass(accordion, 'accordion-content');  expect(contents.length).toEqual(0);  TestUtils.Simulate.click(headers[0]);  contents = TestUtils.scryRenderedDOMComponentsWithClass(accordion, 'accordion-content');  expect(contents.length).toEqual(1);  expect(contents[0].textContent).toEqual('Content belongs to title 1');  TestUtils.Simulate.click(headers[0]);  contents = TestUtils.scryRenderedDOMComponentsWithClass(accordion, 'accordion-content');  expect(contents.length).toEqual(0);  }); |

The tests are RED again. Before we move on, it’s time to think about the code design again. If you obeserve, we have a stateful component now. It means, each of the *pane* in the Accordion can have its own closed or open states.

While designing your applications, it’s a bad practice to put whole logic into one single component. We should think about small components — which handles single tasks — and create composite components for big features. Here, in our code we should move the state-full code into a separate component; Let’s call it as a**Pane** component.

See the code below:

accordionComponent.js

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53  54  55  56  57  58  59 | 'use strict';  var React = require('react');  var Pane = React.createClass({  getInitialState: function(){  return {  show: this.props.data.showOnLoad  };  },  toggle: function(){  this.setState({  show: !this.state.show  });  },  render: function(){  var getContent = (function(item){  return this.state.show ? (  <div className="accordion-content">  {item.content}  </div>  ) : '';  }).bind(this);  var item = this.props.data;  return (  <div>  <div className="accordion-header" onClick={this.toggle}>  {item.name}  </div>  { getContent(item) }  </div>  );  }  });  var Accordion = React.createClass({  render: function(){  var panes = [];  var data = this.props.data || [];  data.forEach(function(item, index){  panes.push(  <Pane data={item} key={index}/>  );  });  return (  <div className="accordion">  {panes}  </div>  );  }  });  module.exports = Accordion; |

All tests are GREEN now ! If you check your browser window ( where the server is running ), you can see that our Accordion component is functional.

Before we summarise, see the final source code and tests:

accordionComponent-spec.js

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53  54  55  56  57  58  59  60  61  62  63  64  65  66  67  68  69  70  71  72  73  74  75  76  77  78  79  80  81  82  83 | jest.dontMock('../scripts/accordionComponent.js');  describe('Accordion', function() {  var React = require('react');  var TestUtils = require('react-addons-test-utils');  var Accordion;  beforeEach(function() {  Accordion = require('../scripts/accordionComponent');  });  it('should exists', function() {  // Render into document  var accordion = TestUtils.renderIntoDocument( <Accordion /> );  expect(TestUtils.isCompositeComponent(accordion)).toBeTruthy();  });  it('should build the layout from an array of objects passed as prop', function() {  var input = [{  name: 'Title 1',  content: 'Content belongs to title 1',  showOnLoad: true  },{  name: 'Title 2',  content: 'Content belongs to title 2',  showOnLoad: true  }];  var accordion = TestUtils.renderIntoDocument( <Accordion data={input}/> );  var headers = TestUtils.scryRenderedDOMComponentsWithClass(accordion, 'accordion-header');  var contents = TestUtils.scryRenderedDOMComponentsWithClass(accordion, 'accordion-content');  expect(headers.length).toEqual(2);  expect(contents.length).toEqual(2);  expect(headers[0].textContent).toEqual('Title 1');  expect(headers[1].textContent).toEqual('Title 2');  expect(contents[0].textContent).toEqual('Content belongs to title 1');  expect(contents[1].textContent).toEqual('Content belongs to title 2');  });  it('should hide all the contents by default unless specified by a flag', function(){  var input = [{  name: 'Title 1',  content: 'Content belongs to title 1'  },{  name: 'Title 2',  content: 'Content belongs to title 2',  showOnLoad: true  }];  var accordion = TestUtils.renderIntoDocument( <Accordion data={input}/> );  var contents = TestUtils.scryRenderedDOMComponentsWithClass(accordion, 'accordion-content');  expect(contents.length).toEqual(1);  expect(contents[0].textContent).toEqual('Content belongs to title 2');  });  it('should be able to toggle the content by clicking on the respective title.', function(){  var input = [{  name: 'Title 1',  content: 'Content belongs to title 1'  },{  name: 'Title 2',  content: 'Content belongs to title 2'  }];  var accordion = TestUtils.renderIntoDocument( <Accordion data={input}/> );  var headers = TestUtils.scryRenderedDOMComponentsWithClass(accordion, 'accordion-header');  var contents = TestUtils.scryRenderedDOMComponentsWithClass(accordion, 'accordion-content');  expect(contents.length).toEqual(0);  TestUtils.Simulate.click(headers[0]);  contents = TestUtils.scryRenderedDOMComponentsWithClass(accordion, 'accordion-content');  expect(contents.length).toEqual(1);  expect(contents[0].textContent).toEqual('Content belongs to title 1');  TestUtils.Simulate.click(headers[0]);  contents = TestUtils.scryRenderedDOMComponentsWithClass(accordion, 'accordion-content');  expect(contents.length).toEqual(0);  });  }); |

accordionComponent.js

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53  54  55  56  57  58  59 | 'use strict';  var React = require('react');  var Pane = React.createClass({  getInitialState: function(){  return {  show: this.props.data.showOnLoad  };  },  toggle: function(){  this.setState({  show: !this.state.show  });  },  render: function(){  var getContent = (function(item){  return this.state.show ? (  <div className="accordion-content">  {item.content}  </div>  ) : '';  }).bind(this);  var item = this.props.data;  return (  <div>  <div className="accordion-header" onClick={this.toggle}>  {item.name}  </div>  { getContent(item) }  </div>  );  }  });  var Accordion = React.createClass({  render: function(){  var panes = [];  var data = this.props.data || [];  data.forEach(function(item, index){  panes.push(  <Pane data={item} key={index}/>  );  });  return (  <div className="accordion">  {panes}  </div>  );  }  });  module.exports = Accordion; |

app.js

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14 | 'use strict';  var React = require('react');  var ReactDOM = require('react-dom');  var Accordion = require('components/accordionComponent');  var input = [{  name: 'Title 1',  content: 'Content belongs to title 1'  },{  name: 'Title 2',  content: 'Content belongs to title 2'  }];  ReactDOM.render(<Accordion data={input}/>, document.getElementById('view')); |

**Summary**