JavaScript, React, and Tests, Oh My!

Daniel Irvine
8th Light

Instagram @craft_of_code
Twitter @d_ir
Web danielirvine.com

20 January 2018

What we'll cover today

Functional JavaScript
ECMAScript 6: functions, modules and immutable data structures

Test-driven development (TDD) with Jasmine
Arrange-Act-Assert
Test doubles and spies
jsdom

React and ReactTestUtils
Virtual DOM
State management
Composing components

Asynchronous behaviour
Promises
Fetch API
setImmediate

An example JavaScript function

```
function add(a, b) {
  return a + b
}
```

An example JavaScript function

```
export function formatIngredientRequirement(
  ingredientFinder, measureFinder, ingredient) {
  if(ingredient.measure) {
    return formatIngredientWithMeasure(measureFinder, ingredient)
  } else {
    return formatWholeIngredient(ingredientFinder, ingredient)
  }
}

function formatIngredientWithMeasure(measureFinder, ingredient) {
 const measure = measureFinder(ingredient.measure)
  if(ingredient.amount === 1) {
    return `1 ${measure.singular} ${ingredient.name}`
  } else {
    return `${ingredient.amount} ${measure.plural} ${ingredient.name}`
  }
}

function formatWholeIngredient(ingredientFinder, ingredient) {
  if(ingredient.amount === 1) {
    return `1 ${ingredient.name}`
  } else {
    const ingredientDetail = ingredientFinder(ingredient.name)
    return `${ingredient.amount} ${ingredientDetail.plural}`
  }
}
```

Testing with Jasmine

```
import { formatIngredientRequirement } from '~/ingredientLister'

it('formats a single whole item', () => {
  const actual = format({name: 'apple', amount: 1})
  expect(actual).toEqual('1 apple')
})


function format(item) {
  return formatIngredientRequirement(
    ingredientFinder, measureFinder, item)
}
```

Testing with Jasmine

```
it('formats multiple whole items', () => {
  const actual = format({name: 'apple', amount: 2})
  expect(actual).toEqual('2 apples')
})

function format(item) {
  return formatIngredientRequirement(
    ingredientFinder, measureFinder, item)
}
```

Testing with Jasmine

```
it('formats multiple whole items', () => {
  const ingredientDetail = {singular: 'apple', plural: 'apples'}
  ingredientFinder = _ => ingredientDetail
  const actual = format({name: 'apple', amount: 2})
  expect(actual).toEqual('2 apples')
})

function format(item) {
  return formatIngredientRequirement(
    ingredientFinder, measureFinder, item)
}
```

Those two tests in full

```
import { formatIngredientRequirement } from '~/ingredientLister'

describe('formatIngredientRequirement', () => {

  let ingredientFinder
  let measureFinder

  it('formats a single whole item', () => {
    const actual = format({name: 'apple', amount: 1})
    expect(actual).toEqual('1 apple')
  })

  it('formats multiple whole items', () => {
    const ingredientDetail = {singular: 'apple', plural: 'apples'}
    ingredientFinder = _ => ingredientDetail
    const actual = format({name: 'apple', amount: 2})
    expect(actual).toEqual('2 apples')
  })

  function format(item) {
    return formatIngredientRequirement(ingredientFinder, measureFinder, item)
  }
})
```

Breathe!

Arrange
Act
Assert

These are the essential three parts of every test.

Test description

```
it('formats multiple whole items', () => {
```

Arrange

```
  const ingredientDetail = {singular: 'apple',
plural: 'apples'}
  ingredientFinder = _ => ingredientDetail
```

Act

```
  const actual = format({name: 'apple', amount: 2})
```

Assert

```
  expect(actual).toEqual('2 apples’)


})
```

Jasmine API Reference

Assertions

```
expect(actual).toEqual(expected)
expect(actual).toBeDefined()
expect(actual).not.toEqual(unexpected)
expect(actual).not.toBeDefined()

expect(string).toContain(substring)
```

For use with spies:

```
expect(spy).toHaveBeenCalled()
expect(spy).toHaveBeenCalledWith(argumentOne, argumentTwo, ...)
```

Jasmine API Reference

Spies

```
spyOn(object, functionName).and.returnValue(returnValue)

e.g.

spyOn(window, 'fetch')
  .and.returnValue(Promise.resolve({json: () => value}))


const mySpy = jasmine.createSpy()
```

Working with the DOM

```
export function resetDom(windowUrl) {
  global.window = new JSDOM('', {url: windowUrl}).window
  global.document = global.window.document
  global.navigator = global.window.navigator
  window.fetch = () => {json: () => ''}
}
```

```
beforeEach(() => {
  resetDom()
})
```

An example React component

```
import React from 'react'
import ReactDOM from 'react'
import Recipe from './recipe'

export default class RecipeList extends React.Component {
  constructor(props) {
    super(props)
 }

  render() {
    return <div id='recipeList'>
      <ul>
      {this.props.recipes.map(this.renderRecipe)}
      </ul>
     </div>
  }

  renderRecipe(recipe) {
    return <li key={recipe}>{recipe}</li>
  }
}
```

Mounting a React component

```
import React from 'react'
import ReactDOM from 'react-dom'
import RecipeList from './recipeList'
import { loadAllRecipes } from './recipeRepository'

const recipes = loadAllRecipes()

ReactDOM.render(
  <RecipeList recipes={recipes} />,
  document.getElementById('main'))
```

Mounting a specific component in tests

```
let container
let component

beforeEach(() => {
  container = document.createElement('div')
})

function mountComponent() {
  component = ReactDOM.render(
    <RecipeList recipeRepository={repository} />,
    container)
}
```

Testing a React component

```
import React from 'react'
import ReactDOM from 'react-dom'
import ReactTestUtils from 'react-dom/test-utils'
import { resetDom } from '../specHelper'
import { loadRecipe } from '~/recipeRepository'
import RecipeList from '~/recipeList'
import Recipe from '~/recipe'

const recipes = ['recipe 1', 'recipe 2']

describe('recipeList', () => {
 let component
 const repository = Promise.resolve(recipes)


 beforeEach(() => {
  resetDom()
  spyOn(Recipe.prototype, 'componentDidUpdate')
 })
```

```
 it('initially displays an empty unordered list', () => {
  mountComponent()
  expect(ul()).toBeDefined()
  expect(ul().children.length).toEqual(0)
 })

it('renders all recipes once they are received', (done) => {
 mountComponent()
 setImmediate(() => {
  expect(ul().children.length).toEqual(2)
  expect(ul().children[0].textContent).toEqual('recipe 1')
  expect(ul().children[1].textContent).toEqual('recipe 2')
  done()
 })
})

function ul() {
 return ReactTestUtils.findRenderedDOMComponentWithTag(component, 'ul')
}
```

Daniel Irvine

Twitter: @d_ir

Testing a React component

```
import React from 'react'
import ReactDOM from 'react-dom'
import ReactTestUtils from 'react-dom/test-utils'
import { resetDom } from '../specHelper'
import { loadRecipe } from '~/recipeRepository'
import RecipeList from '~/recipeList'
import Recipe from ‘~/recipe'

const recipes = ['recipe 1', 'recipe 2']

describe('recipeList', () => {
 let component
 const repository = Promise.resolve(recipes)


 beforeEach(() => {
  resetDom()
  spyOn(Recipe.prototype, 'componentDidUpdate')
 })


 it('initially displays an empty unordered list', () => {
  mountComponent()
  expect(ul()).toBeDefined()
  expect(ul().children.length).toEqual(0)
 })
```

ReactTestUtils API Reference

```
import ReactTestUtils from 'react-dom/test-utils'

ReactTestUtils.findRenderedDOMComponentWithTag(component, 'ul')
ReactTestUtils.findRenderedComponentWithType(component, Recipe)
ReactTestUtils.findRenderedDOMComponentWithClass(component, cssClassName)

ReactTestUtils.scryRenderedDOMComponentsWithTag(component, 'ul')

ReactTestUtils.Simulate.change(selectBox, {target: {value: option}} )
ReactTestUtils.Simulate.click(link)


node.querySelector('ul')
node.querySelectorAll('li')
```

State

```
constructor(props) {
  super(props)
  this.handleChooseRecipe = this.handleChooseRecipe.bind(this)
  this.renderRecipe = this.renderRecipe.bind(this)
  this.state = {}
}

render() {
  return <div id='recipeList'>
    <ul>
    {this.state.recipes.map(this.renderRecipe)}
    </ul>
    <Recipe chosenRecipe={this.state.chosenRecipe} />
    </div>
}

renderRecipe(recipe) {
  return <li key={recipe}><a onClick={this.handleChooseRecipe}>{recipe}</a></li>
}

handleChooseRecipe(e) {
  this.setState({
    chosenRecipe: e.target.textContent
  })
}
```

Another use of state

```
constructor(props) {
  super(props)
  this.recipesReceived = this.recipesReceived.bind(this)
  this.props.recipeRepository.then(this.recipesReceived)
  this.state = {
    recipes: []
  }
}

recipesReceived(recipes) {
  this.setState({
    recipes: recipes
  })
}

render() {
  return <div id='recipeList'>
    <ul>
    {this.state.recipes.map(this.renderRecipe)}
    </ul>
    <Recipe chosenRecipe={this.state.chosenRecipe} />
    </div>
}
```

We never test state!

Function-only React component

Shielding ourselves from the framework

Understanding the virtual DOM

Daniel Irvine

Twitter: @d_ir

Promises

```
componentDidUpdate(oldProps, oldState) {
  if(oldProps.chosenRecipe !== this.props.chosenRecipe) {
    this.doLoad()
  }
}

doLoad() {
  if(this.props.chosenRecipe) {
    this.props.recipeLoader(this.props.chosenRecipe).then(recipe => {
      this.setState({
        recipe: recipe
      })
    })
  }
}
```

Testing promises

```
it('loads the given recipe using the recipe loader', (done) => {
  mountComponent('Avocado bagel')
  setImmediate(() => {
    expect(recipeLoader).toHaveBeenCalledWith('Avocado bagel')
    done()
  })
})
```

Default properties

```
import React from 'react'
import ReactDOM from 'react-dom'
import { loadRecipe } from './recipeRepository'

export default class Recipe extends React.Component {
  // ...
}

Recipe.defaultProps = {
  recipeLoader: loadRecipe
}
```

Component hierarchies

```
RecipeList > Recipe > Ingredient
```

Stubbing out children

recipeListSpec.js

```
beforeEach(() => {
  resetDom()
  spyOn(Recipe.prototype, 'componentDidUpdate')
})
```

Exercises

See exercises.md in the repo

What we haven’t covered much yet: inputting data.

8th Light is hiring!

Especially senior developers