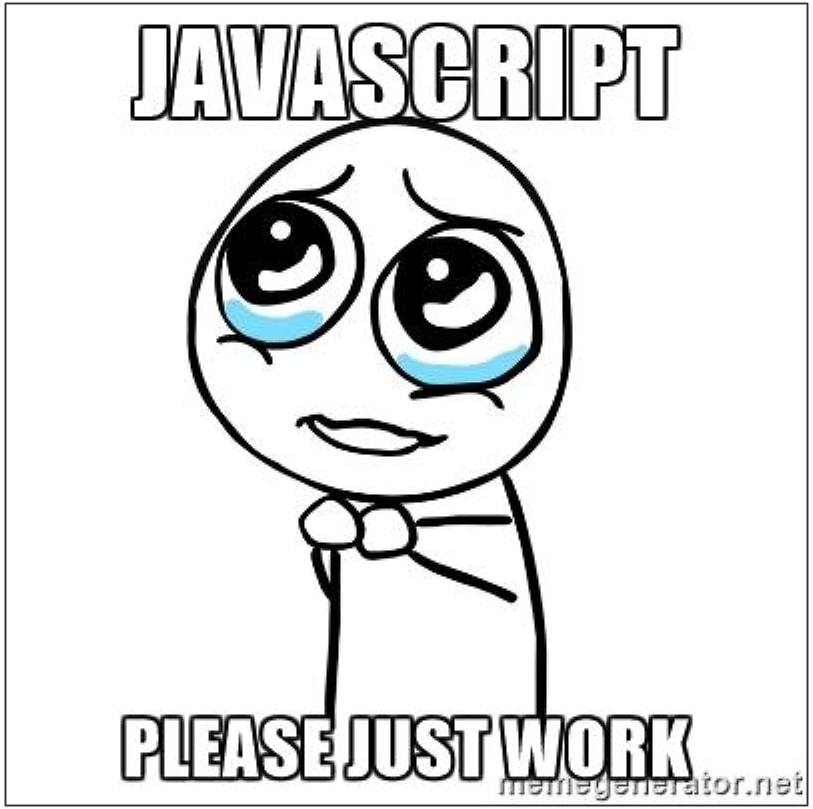
Tips and Tricks for Testing JavaScript


JAVASCRIPT
PLEASE JUST WORK
memegenerator.net

Use case 1: testing AJAX calls

UC: testing AJAX calls

UC: testing AJAX calls

Code

```
function getRecommendedMeal () {
    $.ajax({
        url: '/get/food/recommendation/' ,
        success (meal) {
            $('body').append(`<p>${meal.name} is rated ${meal.rating}</p>`);
        },
        error () {
            $('body').append('<p>Could not fetch recommendation.</p>' );
        }
    });
}
```

UC: testing AJAX calls

Test

```
const expect = require('expect');
const $ = require('jquery');

const { getRecommendedMeal } = require('./demo.js');
```

UC: testing AJAX calls

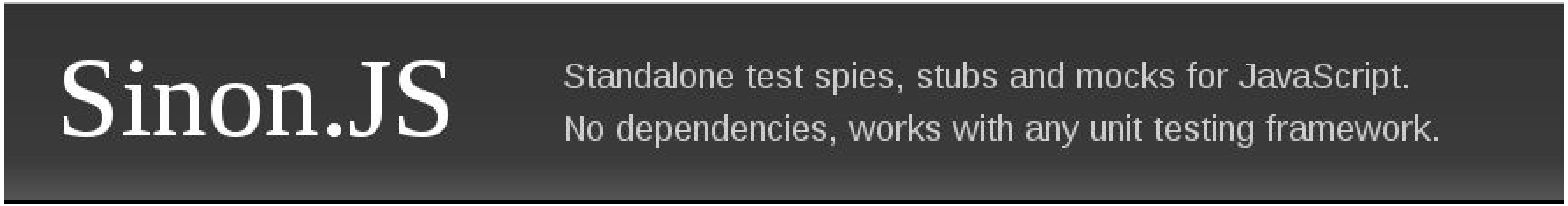
Test

```
const expect = require('expect');
const $ = require('../libs/jquery.js');

const { getRecommendedMeal } = require('./demo.js');


describe('AJAX example', function () {
    it('get food recommendation - success' , function () {




    });
});
```

UC: testing AJAX calls

Test

```
const expect = require('expect');
const $ = require('../libs/jquery.js');

const { getRecommendedMeal } = require('./demo.js');


describe('AJAX example',  function () {
    it('get food recommendation - success',  function () {
        getRecommendedMeal();
        const result = document.querySelector('p');
        expect(result.textContent).toBe( 'pizza is rated 4.2');
    });
});
```



Sinon.JS

Standalone test spies, stubs and mocks for JavaScript.
No dependencies, works with any unit testing framework.

UC: testing AJAX calls

Test

```
const sinon = require('sinon');

describe('AJAX example', function () {









});
```

UC: testing AJAX calls

Test

```
const sinon = require('sinon');

describe('AJAX example', function () {
    let ajaxStub;

    beforeEach(function () {
        document.body.innerHTML = '';
        ajaxStub = sinon.stub($, 'ajax');
    });

    afterEach(function () {
        $.ajax.restore();
    });
    ...
});
```

UC: testing AJAX calls

Test

```
const sinon = require('sinon');

describe('AJAX example', function () {
    ...
    it('get food recommendation - success', function () {

        ajaxStub.yieldsTo('success', {name: 'pizza', rating: 4.2});

        getRecommendedMeal();
        const result = document.querySelector('p');
        expect(result.textContent).toBe('pizza is rated 4.2');
    });
});
```

UC: testing AJAX calls

What about testing errors?

UC: testing AJAX calls

Test

```
const sinon = require('sinon');

describe('AJAX example', function () {
    ...
    it('get food recommendation - error', function () {

        ajaxStub.yieldsTo( 'error');

        getRecommendedMeal();
        const result = document.querySelector('p');
        expect(result.textContent).toBe('Could not fetch recommendation');
    });
});
```

UC: testing AJAX calls

What about multiple AJAX calls with different results?

UC: testing AJAX calls

UC: testing AJAX calls

Code

```
function getIngredientsOfRecommendation () {





    const getRecommendation = function () {
        $.ajax({
            url: '/get/food/recommendation/',
            success (meal) {
                $('body').append(`<h1>${meal.name}</h1>`);
                getIngredients(meal.id);
            },
        });
    };
    getRecommendation();
}
```

UC: testing AJAX calls

Code

```
function getIngredientsOfRecommendation () {
    const getIngredients = function (mealId) {
        $.ajax({
            url: `/ingredients/${mealId}/`,
            success (ingredients) {
                $('body').append(`<p>${ingredients.join(', ')}</p>`);
            },
        });
    };
    const getRecommendation = function () {
        $.ajax({
            url: '/get/food/recommendation/',
            success (meal) {
                $('body').append(`<h1>${meal.name}</h1>`);
                getIngredients(meal.id);
            },
        });
    };
    getRecommendation();
}
```

UC: testing AJAX calls

Test

```
const sinon = require('sinon');

describe('AJAX example', function () {
        ...
    it('get ingredients for recommended meal', function () {
        ajaxStub.onCall(0).yieldsTo('success', {name: 'Pizza Funghi', id: 3})
                .onCall(1).yieldsTo('success', ['mushrooms', 'cheese']);





    });
});
```

UC: testing AJAX calls

Test

```
const sinon = require('sinon');

describe('AJAX example', function () {
        ...
    it('get ingredients for recommended meal', function () {
        ajaxStub.onCall(0).yieldsTo('success', {name: 'Pizza Funghi', id: 3})
                .onCall(1).yieldsTo('success', ['mushrooms', 'cheese']);
        getIngredientsOfRecommendation();
        const headline = document.querySelector('h1');
        expect(headline.textContent).toBe('Pizza Funghi');
        const ingredients = document.querySelector('p');
        expect(ingredients.textContent).toBe('mushrooms, cheese');
    });
});
```

UC: testing AJAX calls

Test

```
const sinon = require('sinon');

describe('AJAX example', function () {
        ...
    it('get ingredients for recommended meal', function () {
        ajaxStub.onCall(0).yieldsTo('success', {name: 'Pizza Funghi', id: 3})
                .onCall(1).yieldsTo('success', ['mushrooms', 'cheese']);
        getIngredientsOfRecommendation();
        const args = ajaxStub.getCall(1).args;
        expect(args[0].url).toBe('/ingredients/3/');
    });
});
```

Use case 2: testing timeouts

UC: testing timeouts

UC: testing timeouts

Code

```
function getPieOutOfTheOven () {
    const pie = {
        state: 'too hot',
    };

    window.setTimeout(function () {
        pie.state = 'ready';
    }, 60000);

    return pie;
}
```

UC: testing timeouts

Test

```
describe('timeout example', function () {
    let clock;

    beforeEach(function () {
        clock = sinon.useFakeTimers();
    });

    afterEach(function () {
        clock.restore();
    });
    ...
});
```

UC: testing timeouts

Test

```
describe('timeout example', function () {

    it('get pie from the oven', function() {
        const pie = getPieOutOfTheOven();
        expect(pie.state).toBe( 'too hot');




    });
});
```

UC: testing timeouts

I'M NOT SAYING WE'RE GONNA TIME TRAVEL
BUT WE'RE GONNA TIME TRAVEL
H HD
HISTORY.COM
DIYLOL.COM

UC: testing timeouts

Test

```
describe('timeout example',  function () {

    it('get pie from the oven',  function() {
        const pie = getPieOutOfTheOven();
        expect(pie.state).toBe('too hot');
        clock.tick(30000);
        expect(pie.state).toNotBe( 'ready');



    });
});
```

UC: testing timeouts

Test

```
describe('timeout example', function () {

    it('get pie from the oven', function() {
        const pie = getPieOutOfTheOven();
        expect(pie.state).toBe('too hot');
        clock.tick(30000);
        expect(pie.state).toNotBe('ready');
        clock.tick(30000);
        expect(pie.state).toBe('ready');
    });
});
```

Use case 3: ajax calls + timeouts = polling

UC: testing polling

UC: testing polling

Code

```
function checkMuffin () {
    const muffin ={
        state: 'baking',
    };









    return muffin;
}
```

UC: testing polling

Code

```
function checkMuffin () {
    const muffin ={
        state: 'baking',
    };

    const checkState = function () {
        $.ajax({
            url: '/muffin/ready/',
            success (response) {
                if (response.ready) {
                    muffin.state = 'ready';
                } else {
                    window.setTimeout(checkState, 1000);
                }
            },
        });
    };
    checkState();
    return muffin;
}
```

UC: testing polling

Test

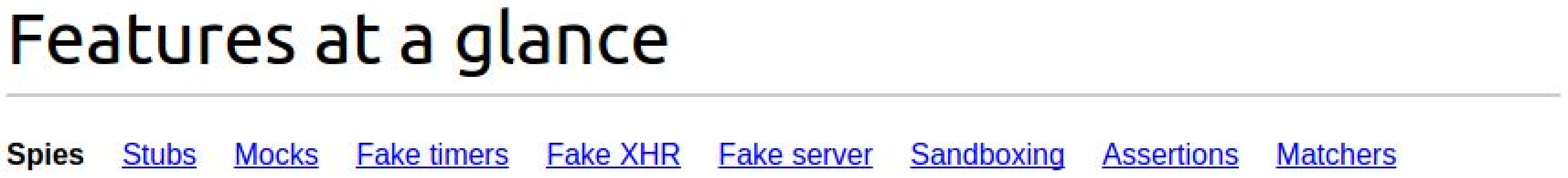
```
describe('polling example', function () {
    let clock;
    let ajaxStub;

    beforeEach(function () {
        clock = sinon.useFakeTimers();
        ajaxStub = sinon.stub($, 'ajax');
    });

    afterEach(function () {
        clock.restore();
        $.ajax.restore();
    });
    ...
});
```

UC: testing polling

Test

```
describe('polling example', function () {
    ...
    it('check if muffins are ready', function() {
        ajaxStub.onCall(0).yieldsTo('success', {ready: false})
                .onCall(1).yieldsTo('success', {ready: false})
                .onCall(2).yieldsTo('success', {ready: true});

        const muffin = checkMuffin();
        expect(muffin.state).toBe('baking');

        clock.tick(1000);
        expect(muffin.state).toBe('baking');

        clock.tick(1000);
        expect(muffin.state).toBe('ready');
    });
});
```

http://sinonjs.org/

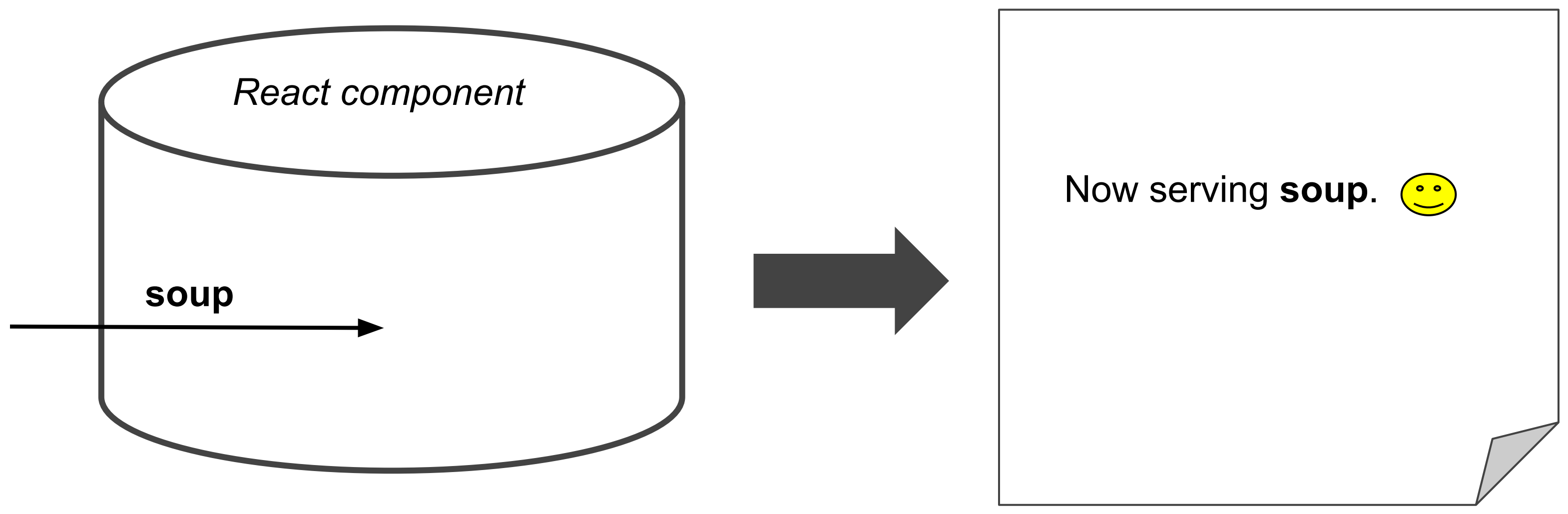


Features at a glance

Spies Stubs Mocks Fake timers Fake XHR Fake server Sandboxing Assertions Matchers

Use case 4: testing React components

UC: testing React components



UC: testing React components

Code

```
const React = require('react');


class StarterView extends React.Component {
    render () {
        return (
            <div>
                Now serving <span className="dish">{this.props.dish}</span>.
                <span className="icon-yummy"></span>
            </div>
        );
    }
}
```

UC: testing React components

Test

```
const TestUtils = require('react-addons-test-utils');

it('test StarterView', function () {









});
```

UC: testing React components

Test

```
const TestUtils = require('react-addons-test-utils');

it('test StarterView', function () {
    const view = TestUtils.renderIntoDocument(
        <StarterView dish='soup' />
    );










});
```

UC: testing React components

Test

```
const TestUtils = require('react-addons-test-utils');

it('test StarterView', function () {
    const view = TestUtils.renderIntoDocument(
        <StarterView dish='soup' />
    );

    const div = TestUtils.findRenderedDOMComponentWithTag(view, 'div');
    expect(div.textContent).toBe('Now serving soup.');




});
```

UC: testing React components

Test

```
const TestUtils = require('react-addons-test-utils');

it('test StarterView', function () {
    const view = TestUtils.renderIntoDocument(
        <StarterView dish='soup' />
    );

    const div = TestUtils.findRenderedDOMComponentWithTag(view, 'div');
    expect(div.textContent).toBe('Now serving soup.');

    const dish = TestUtils.findRenderedDOMComponentWithClass(view, 'dish');
    expect(dish.textContent).toBe('soup');



});
```

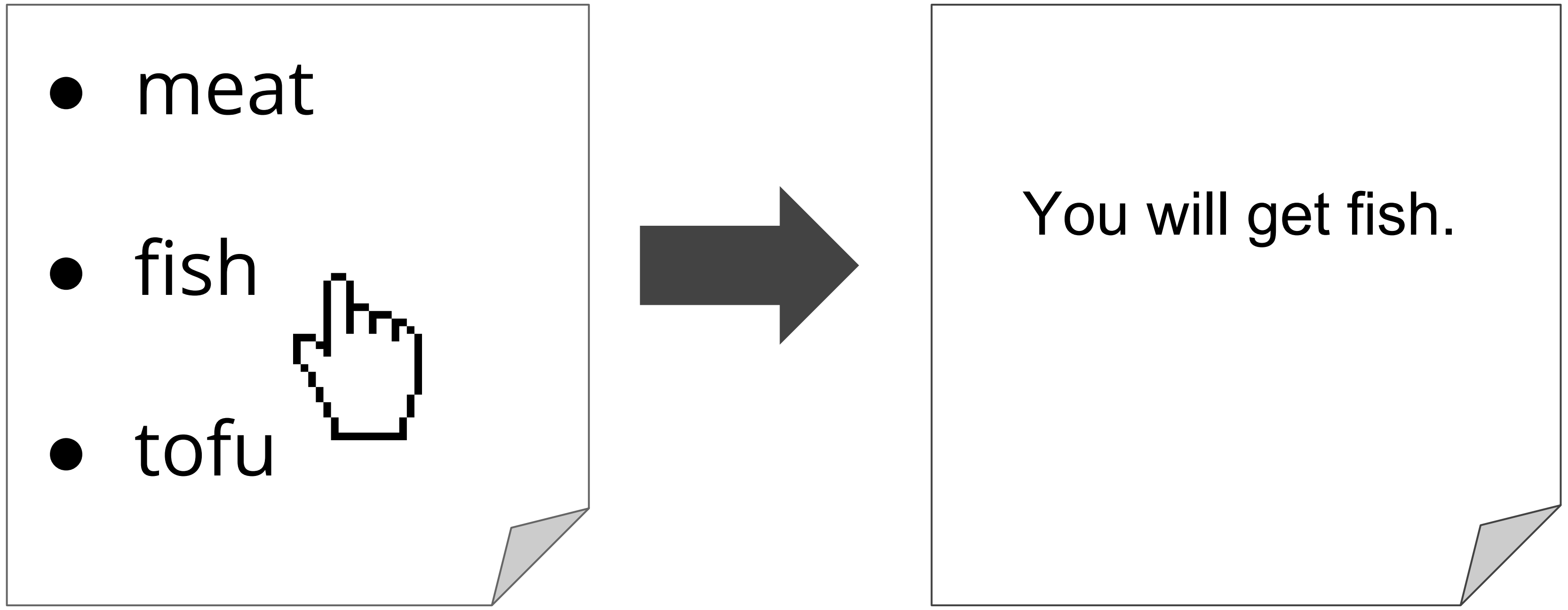
UC: testing React components

Test

```
const TestUtils = require('react-addons-test-utils');

it('test StarterView', function () {
    const view = TestUtils.renderIntoDocument(
        <StarterView dish='soup' />
    );

    const div = TestUtils.findRenderedDOMComponentWithTag(view, 'div');
    expect(div.textContent).toBe('Now serving soup.');

    const dish = TestUtils.findRenderedDOMComponentWithClass(view, 'dish');
    expect(dish.textContent).toBe('soup');

    const spans = TestUtils.scryRenderedDOMComponentsWithTag(view, 'span');
    expect(spans[1].className).toInclude('yummy');
});
```

Use case 5: testing React events

UC: testing React events



UC: testing React events

Code

```
const OPTIONS = ['meat', 'fish', 'tofu'];

class MainCourseView extends React.Component {

    constructor(props) {
        super(props);
        this.state = {};
    }

    ...

}
```

UC: testing React events

Code

```
const OPTIONS = ['meat', 'fish', 'tofu'];

class MainCourseView extends React.Component {
    ...

    renderOptions (opt) {
        return <input type="radio" key={opt} value={opt}
                onChange={this.chooseMeal.bind(this)} />;
    }

    render () {
        if (this.state.selected) {
            return <div className="selection">You will get {this.state.selected}.</div>;
        } else {
            return <div>{OPTIONS.map(this.renderOptions.bind(this))}</div>;
        }
    }
}
```

UC: testing React events

Code

```
const OPTIONS = ['meat', 'fish', 'tofu'];

class MainCourseView extends React.Component {
    ...

    chooseMeal (event) {
        const choice = event.target.value;
        this.setState({selected: choice});
    }

    renderOptions (opt) {...}

    render () {...}
}
```

UC: testing React events

Test

```
it('test MainCourseView', function () {
    const view = TestUtils.renderIntoDocument(
        <MainCourseView />
    );









});
```

UC: testing React events

Test

```
it('test MainCourseView', function () {
    const view = TestUtils.renderIntoDocument(
        <MainCourseView />
    );
    expect(view.state.selected).toNotExist();









});
```

UC: testing React events

Test

```
it('test MainCourseView', function () {
    const view = TestUtils.renderIntoDocument(
        <MainCourseView />
    );
    expect(view.state.selected).toNotExist();

    const radios = TestUtils.scryRenderedDOMComponentsWithTag(view, 'input');
    TestUtils.Simulate.change(radios[1]);




});
```

UC: testing React events

Test

```
it('test MainCourseView', function () {
    const view = TestUtils.renderIntoDocument(
        <MainCourseView />
    );
    expect(view.state.selected).toNotExist();

    const radios = TestUtils.scryRenderedDOMComponentsWithTag(view, 'input');
    TestUtils.Simulate.change(radios[1]);

    expect(view.state.selected).toBe('fish');
    const selection = TestUtils.findRenderedDOMComponentWithClass(view, 'selection');
    expect(selection.textContent).toBe('You will get fish.');
});
```

Use case 6: testing change of props for React components

UC: testing prop change

UC: testing prop change

Code

```
class RollingSushi extends React.Component {
    constructor(props) {
        super(props);
        this.state = {sushiType: 'maki'};
    }

    ...

    render () {
        return (
            <div>
                How about {this.state.sushiType}?
                <SushiView type={this.state.sushiType} />
            </div>
        )
    }
}
```

UC: testing prop change

Code

```
class RollingSushi extends React.Component {
   ...
    componentDidMount () {
        this.getSushi();
    }

    getSushi () {
        $.ajax({
            url: 'get/sushi/',
            success: (response) => {
                this.setState({sushiType: response.type});
                window.setTimeout(this.getSushi.bind(this), 5000);
            }
        });
    }
    ...
}
```

UC: testing prop change

Code

```
class SushiView extends React.Component {
    constructor (props) {
        super(props);
        this.state = {eatSushi: true};
    }











}
```

UC: testing prop change

Code

```
class SushiView extends React.Component {
    constructor (props) {
        super(props);
        this.state = {eatSushi: true};
    }











    render () {
        if (this.state.eatSushi) {
            return <p>Om nom nom nom</p>;
        } else {
            return <p>next please</p>;
        }
    }
}
```

UC: testing prop change

Code

```
class SushiView extends React.Component {
    constructor (props) {
        super(props);
        this.state = {eatSushi: true};
    }

    componentWillReceiveProps (nextProps) {
        if (nextProps.type != this.props.type) {
            this.setState({eatSushi: true});
        } else {
            this.setState({eatSushi: false});
        }
    }

    render () {
        if (this.state.eatSushi) {
            return <p>Om nom nom</p>;
        } else {
            return <p>next please</p>;
        }
    }
}
```

UC: testing prop change

Test

```
it('test SushiView', function () {
    const container = document.createElement('div');







});
```

UC: testing prop change

Test

```
const ReactDOM= require('react-dom');
it('test SushiView', function () {
    const container = document.createElement('div');
    const view = ReactDOM.render(<SushiView type="sashimi" />, container);









});
```

UC: testing prop change

Test

```
const ReactDOM= require('react-dom');
it('test SushiView', function () {
    const container = document.createElement('div');
    const view = ReactDOM.render(<SushiView type="sashimi" />, container);

    ReactDOM.render(<SushiView type="nigiri" />, container);







});
```

UC: testing prop change

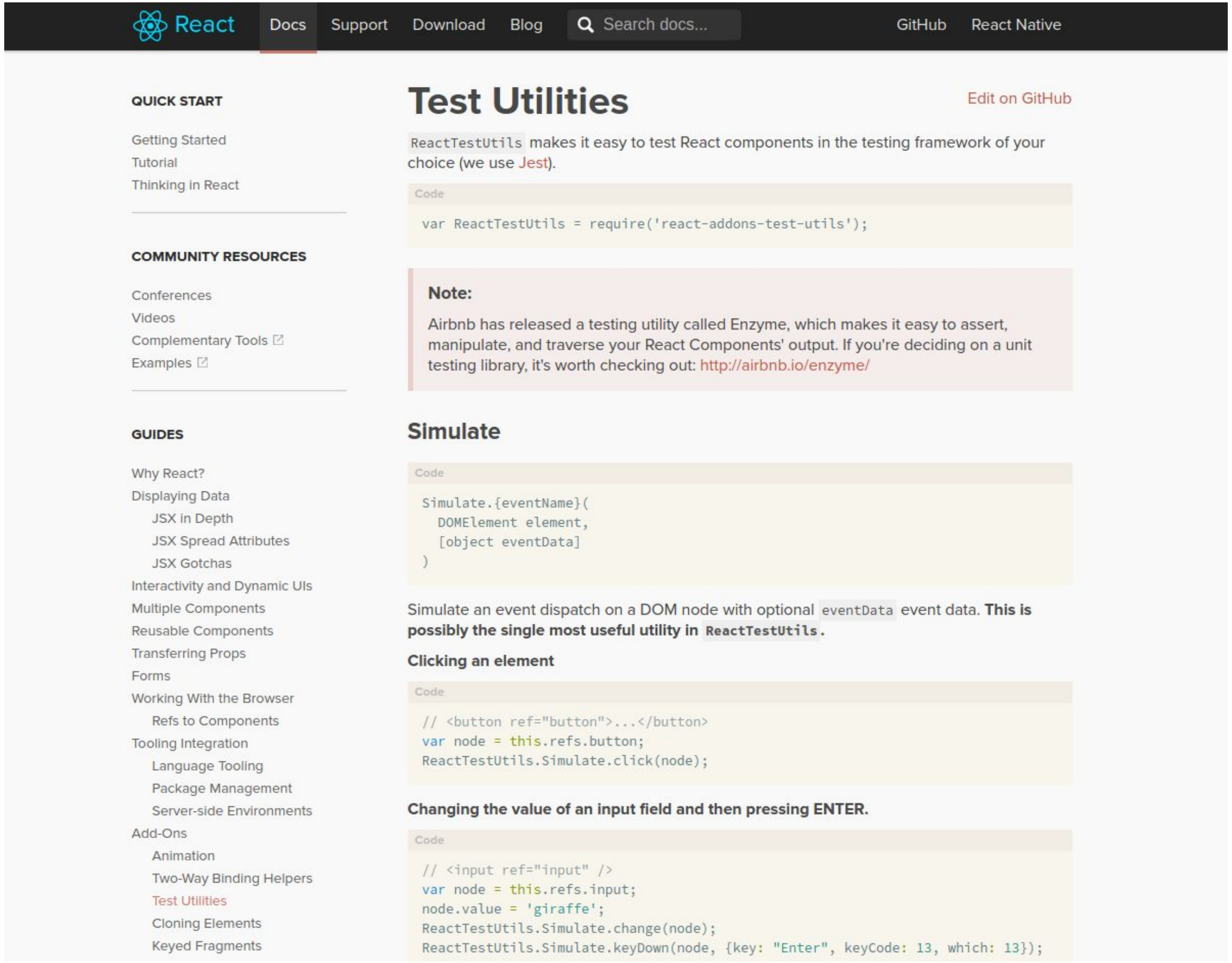
Test

```
const ReactDOM= require('react-dom');
it('test SushiView', function () {
    const container = document.createElement('div');
    const view = ReactDOM.render(<SushiView type="sashimi" />, container);

    ReactDOM.render(<SushiView type="nigiri" />, container);
    let answer = TestUtils.findRenderedDOMComponentWithTag(view, 'p');
    expect(answer.textContent).toBe('Om nom nom nom');




});
```

UC: testing prop change

Test

```
const ReactDOM= require('react-dom');
it('test SushiView', function () {
    const container = document.createElement('div');
    const view = ReactDOM.render(<SushiView type="sashimi" />, container);

    ReactDOM.render(<SushiView type="nigiri" />, container);
    let answer = TestUtils.findRenderedDOMComponentWithTag(view, 'p');
    expect(answer.textContent).toBe('Om nom nom nom');

    ReactDOM.render(<SushiView type="nigiri" />, container);
    answer = TestUtils.findRenderedDOMComponentWithTag(view, 'p');
    expect(answer.textContent).toBe('next please');
    container.remove();
});
```



Test Utilities

Edit on GitHub

`ReactTestUtils` makes it easy to test React components in the testing framework of your choice (we use Jest).

Code
```
var ReactTestUtils = require('react-addons-test-utils');
```

Note:

Airbnb has released a testing utility called Enzyme, which makes it easy to assert, manipulate, and traverse your React Components' output. If you're deciding on a unit testing library, it's worth checking out: http://airbnb.io/enzyme/

Simulate

Code
```
Simulate.{eventName}(
  DOMElement element,
  [object eventData]
)
```

Simulate an event dispatch on a DOM node with optional `eventData` event data. **This is possibly the single most useful utility in `ReactTestUtils`.**

Clicking an element

Code
```
// <button ref="button">...</button>
var node = this.refs.button;
ReactTestUtils.Simulate.click(node);
```

Changing the value of an input field and then pressing ENTER.

Code
```
// <input ref="input" />
var node = this.refs.input;
node.value = 'giraffe';
ReactTestUtils.Simulate.change(node);
ReactTestUtils.Simulate.keyDown(node, {key: "Enter", keyCode: 13, which: 13});
```

Questions?