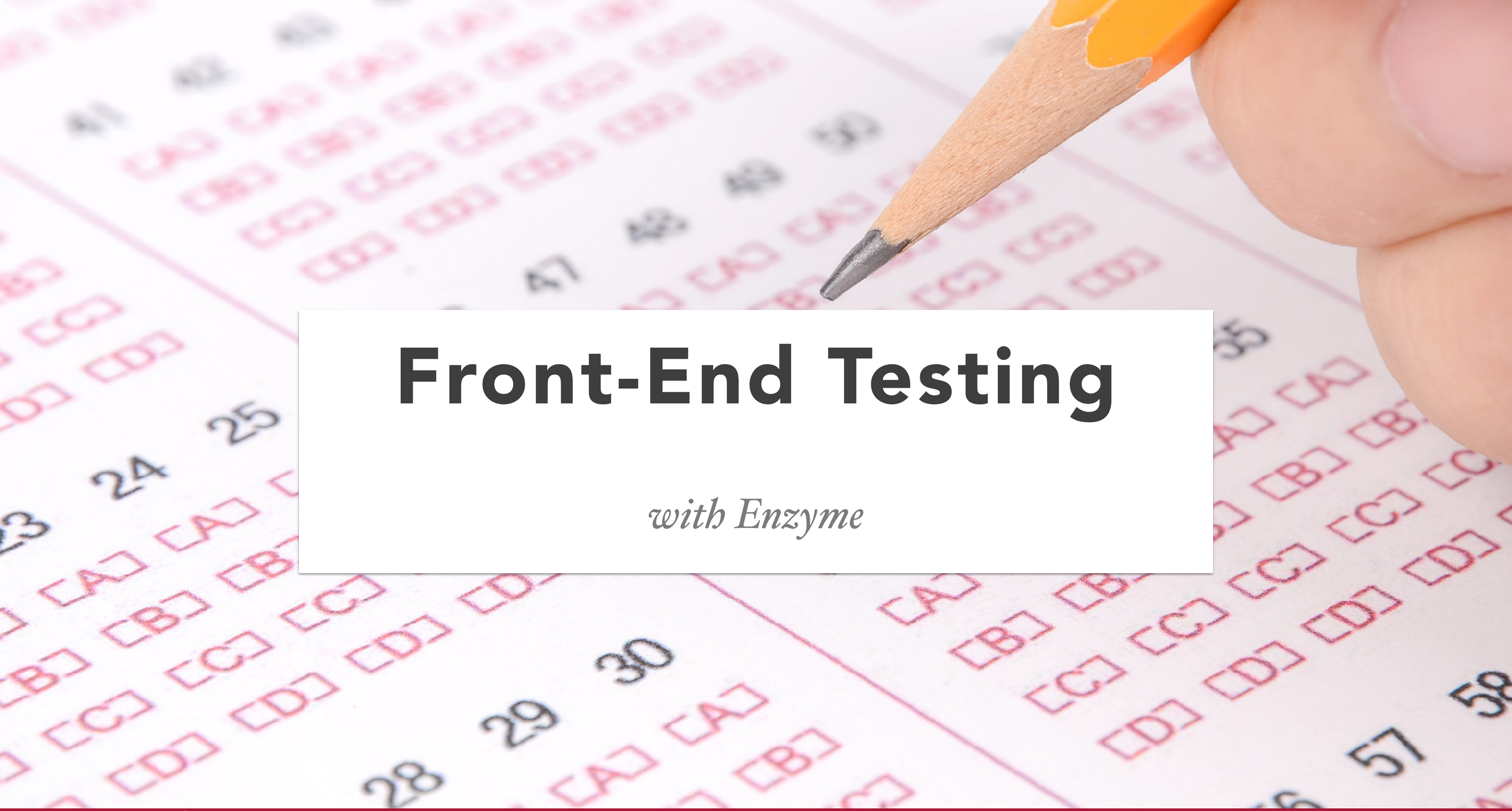


# Front-End Testing

*with Enzyme*

# Roadmap

- Front-end vs. back-end testing issues
- Orientation: the Full(test)stack
  - code · React · Enzyme · Webpack · Chai ·
- Details: Testing React Components
- Demo
  - Desig.ny

# How does front-end testing differ from back-end?

# Front vs Back

- Back end we control the environment; front end may be almost anything
- How do we test processes that extend multiple components?
- How do isolate various elements of the Redux loop?

# Enzyme

- Developed by AirBnB
- Flexible React testing framework focused on “shallow rendering” of components.

# Why we like it

- **Testing Isomorphism:**
  - Tests look similar on the front and back ends.
- **chai for assertions**
- **sinon for spies**

# Roadmap

- Front-end vs. back-end testing issues
- Orientation: the Full(test)stack
  - code · React · Enzyme · Webpack · Chai ·
- Details: Testing React Components
- Demo

/

chai

Sinon.JS

# Install all the things

```
#  mocha, chai, sinon
npm install mocha --save-dev
npm install chai sinon --save-dev # must `require` each in tests
# enzyme
npm install enzyme --save-dev
# if using React 15.x
npm i --save-dev react-addons-test-utils
```

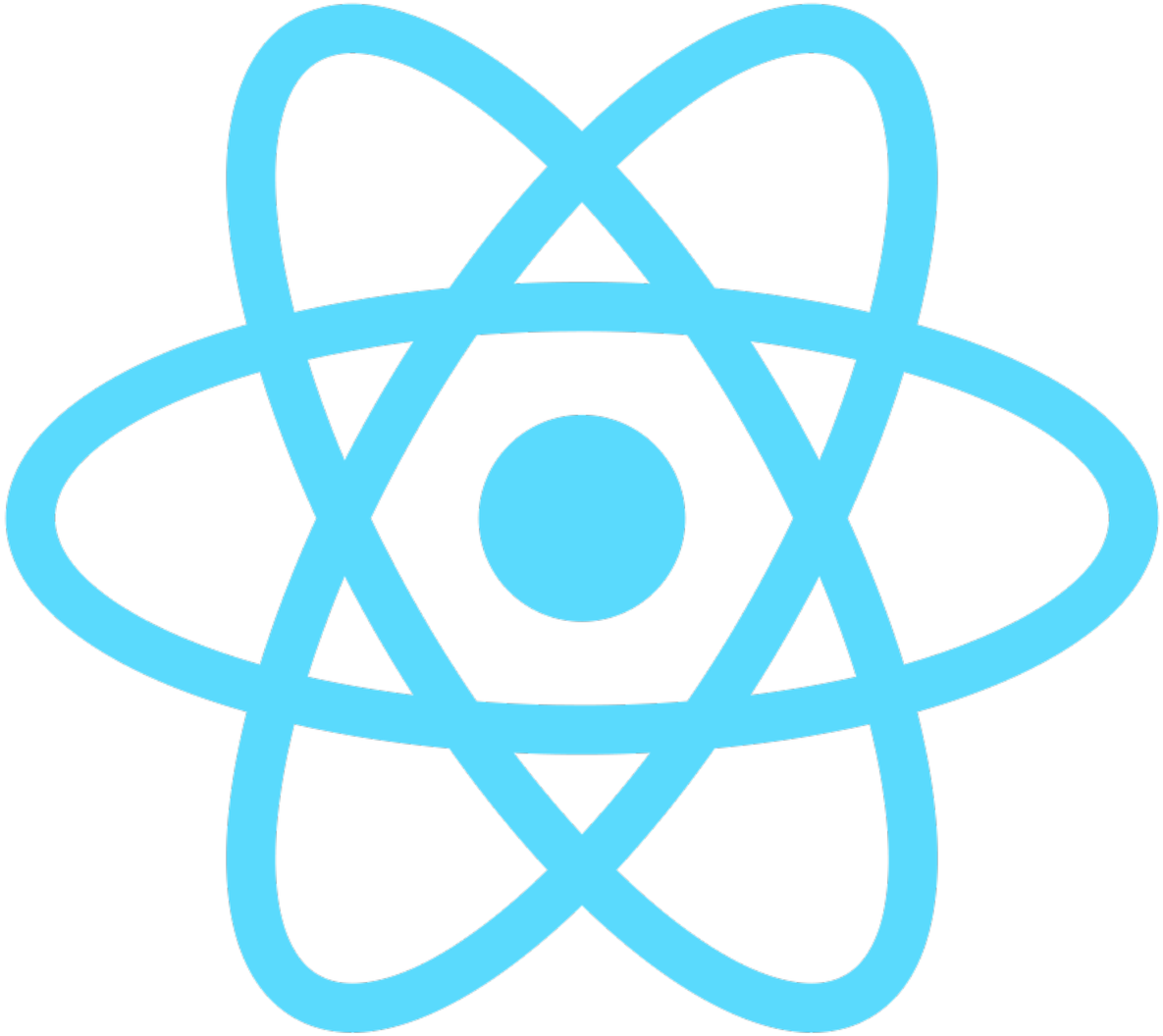

# Review of Assertions in Mocha/Chai

```
describe('A suite', function(){ // `describe` & `it` from mocha

  var myString;
  beforeEach(function(){
    myString = 'testing is fun!';
  });

  it('uses Mocha, Chai, and Sinon', function(){
    var spy = sinon.spy(); // sinon spy
    spy();
    expect(spy.called).to.be.ok; // `expect` & matcher from chai
  });

});
```



# Roadmap

- Front-end vs. back-end testing issues
- Orientation: the Full(test)stack
  - code · React · Enzyme · Webpack · Chai ·
- Details: Testing React Components
- Resources
  - Fullstack Academy Capstone Project Guidelines
  - Docs · Articles

# Setup tools & child components via import

```
import React from 'react';
import {expect} from 'chai';
import {shallow} from 'enzyme';
import {spy} from 'sinon';

import ChildComponent from '../../src/components/Child';

describe(‘This component', () => { … })
```

# Setup Example: a Component

```
describe('The Kitten Component', () => {

    beforeEach('Create component and onChange spy', () => {
       let clickSpy = spy();
       let kitty = shallow(<Kitten onClick ={clickSpy} />);
      });

     it(‘calls click fn’,()=> {
       kitty.simulate('click', {});
       expect(clickSpy.called).to.be.true;
    });
});
```

# Redux

# Redux is important to test too

- Mainly Action Creators and the Reducer
- In React/Redux your app's state = your app's view and behavior

# Action Creator

```
// suppose we have this action creator

export const sayMeow = volume => {
    return {
        type: ‘SAYS_MEOW’,
        volume
    };
};
```

# The test

```
import {expect} from 'chai';

import {sayMeow} from '../../src/store/action-creators';


describe('saysMeow', () => {

    it('saysMeow loudly', () => {

        const loudMeow = '12dbs'

        expect(sayMeow(volume)).to.be.deep.equal({
            type: 'SAY_MEOW',
            volume: loudMeow
        });

    });

});
```

# Reducer

```
const initialState = {
    volume: '0'
};

export default (state = initialState, action) => {

    const newState = Object.assign({}, state);

    switch (action.type) {

        case 'SAY_MEOW':
            newState.volume = action.volume;
            break;

        default:
            return state;
    }

    return newState;

};
```

```
import {expect} from 'chai';

import {createStore} from 'redux';
import mainReducer from '../../src/store/reducers/main';

describe('Main reducer', () => {

    let testStore;
    beforeEach('Create testing store', () => {
        testStore = createStore(mainReducer);
    });

    it('has expected initial state', () => {
        expect(testStore.getState()).to.be.deep.equal({
            volume : '0dbs'
        });
    });

    describe('Says Meow', () => {

        it('says Meow', () => {
            testStore.dispatch({ type: 'SAY_MEOW', volume: '99dbs' });
            const newState = testStore.getState();
            expect(newState.volume).to.be.deep.equal('99dbs');
        });

    });


});
```

# Priorities

- **Tests are valuable when they:**
  - Help drive writing the code in the first place
  - Cover heavily re-used code, especially for multiple contexts
  - Reveal breaking changes
  - Act as documentation
- **Tests are less valuable when they:**
  - Get in the way of meaningful progress
  - Are not idempotent / are reliant on a specific order
  - Are so brittle/specific that they discourage refactoring
  - Focus on implementation instead of behavior

# Roadmap

- Front-end vs. back-end testing issues
- Orientation: the Full(test)stack
- Details: Testing Remo Components
- Demo