

# Lab 6: Testing

Software Studio  
DataLab, CS, NTHU

# Notice

- This lab is about software development good practices
- Interesting for those who like software development and want to go deeper
- Good to optimize your development time

# Outline

- Introduction to Testing
- Simple testing in JS
- Testing with Karma
  - Configuring Karma
  - Karma demo: Simple test
  - Karma demo: Weather Mood
  - Karma demo: Timer App
- Conclusions

# Outline

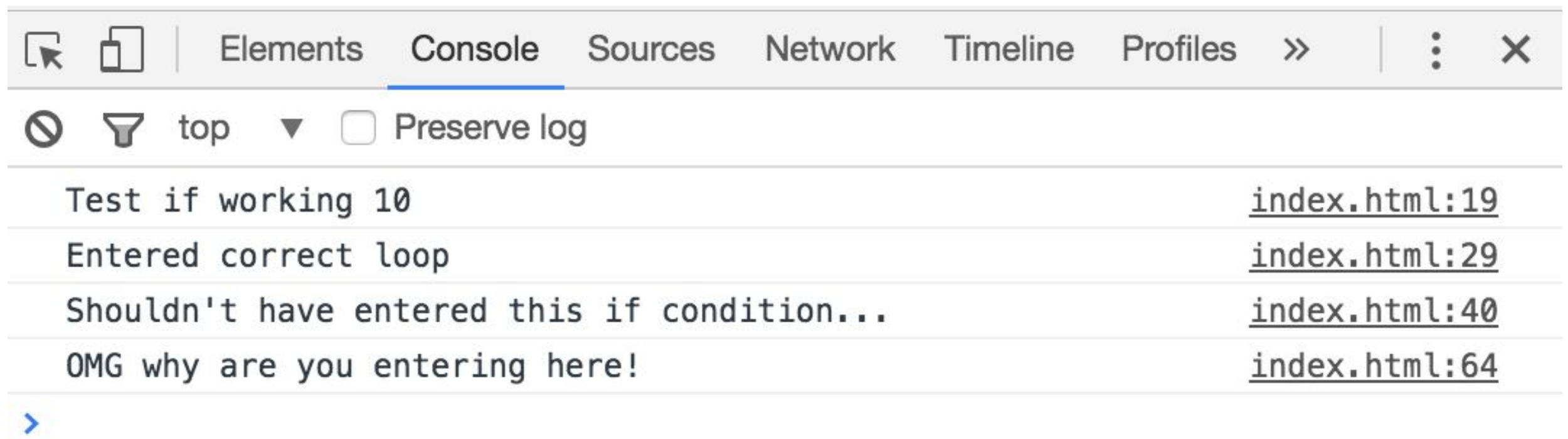
- Introduction to Testing
- Simple testing in JS
- Testing with Karma
  - Configuring Karma
  - Karma demo: Simple test
  - Karma demo: Weather Mood
  - Karma demo: Timer App
- Conclusions

What is testing?  
Is it important?

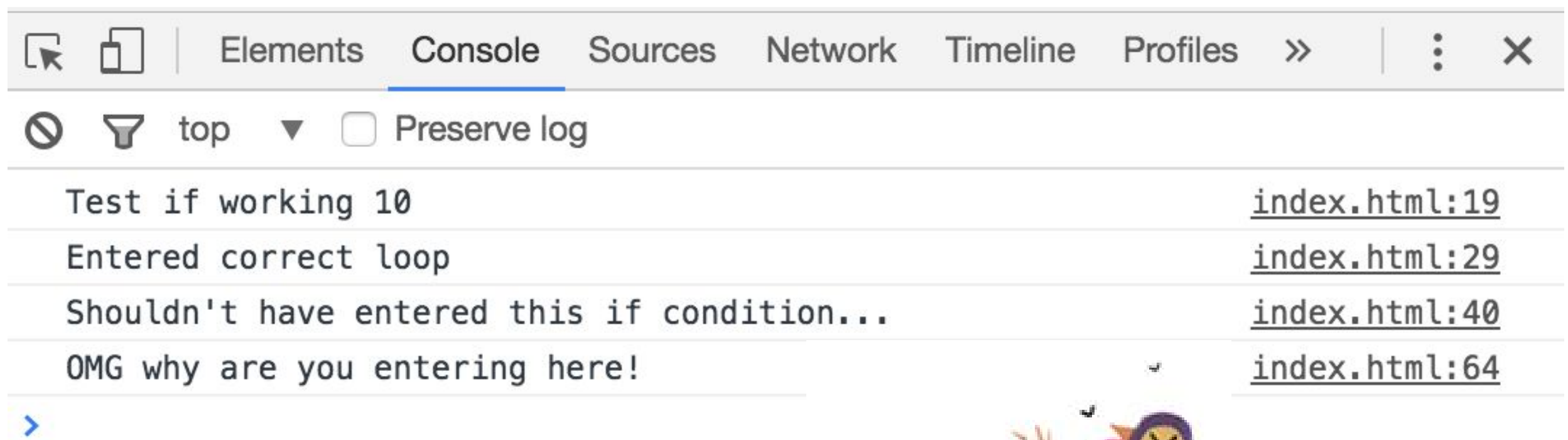
Actually, we are always  
doing some kind of  
testing..

```
<script>  
  var value = 10;  
  console.log('Test if working ' + value);  
</script>
```

# And expecting some result..

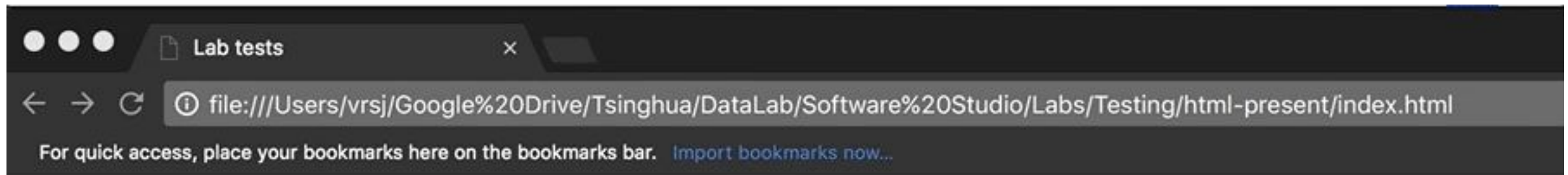


# And expecting some result..





Sometimes we just open  
our app and start clicking  
to see results



[Is this the correct link?](#)

Why test like a machine?  
We can write programs to  
automatically test our  
application!

Most common type of  
test: Unit test

# Unit test

- “Unit testing is a method by which individual units of source code are tested to determine if they are fit for use. A unit is the smallest testable part of an application.”

- Wikipedia

# Good unit test properties

- It should be automated and repeatable
- It should be easy to implement
- Once it's written, it should remain for future use
- Anyone should be able to run it
- It should run at the push of a button
- It should run quickly

The art of Unit Testing, Roy Oshero

# Many frameworks for unit tests in several languages

**JUnit**



# Common procedure

- Assertion: A statement that a predicate is expected to always be true at that point in the code.

-Wikipedia



# Common procedure

- We write our application and want it to do something, so we write a test to see if it's behaving as we expect

# Common procedure

- Pseudocode:

```
function sum2numbers(number1, number2)
{
    return number1 + number2;
}

function testsum2numbers()
{
    return assert(sum2numbers(1, 2) == 3)
}
```


But how to do that in JS?


# Outline


- Introduction to Testing
- Simple testing in JS
- Testing with Karma
  - Configuring Karma
  - Karma demo: Simple test
  - Karma demo: Weather Mood
  - Karma demo: Timer App
- Conclusions

Simple testing:  
expect library and jsbin

 [mjackson](#) / [expect](#)

 Watch ▾22

 Unstar1,392

 Fork80

- <> Code
-  Issues20
-  Pull requests30
-  Projects0
-  Pulse
-  Graphs

Write better assertions

 340 commits

 33 branches

 37 releases

 30 contributors

 MIT

Branch: master ▾New pull request

Create new fileUpload filesFind file

Clone or download ▾

Secure


https://jsbin.com/citudug/edit?html,output

☆

ABPoff

AppsHackMD - Collabor...Baron Schwartz's B...Free Code CampHanziCraft - Inform...Zika Virus Epidemic...Slido - Audience Int...Learn Git BranchingHome | Daniel Rodri...Other Bookmarks

×



New bin

Open bin...

JS Bin features »

Getting started

Keyboard Shortcuts

Exporting/importing gist

☐ Textarea editor mode

Pro features »

Private bins

Dropbox backup

Vanity URLs

Upgrade to pro now

Blog »

Live reloading CSS

Help »

Adding new pre-processors

When do revisions change?

Donate to JS Bin ♥ »

Support JS Bin to keep the project open source & MIT for all

Follow @js\_bin on twitter

By using JS Bin you agree to our legal terms

"Everyone should learn how to program a computer because it teaches you how to think" — Steve Jobs

FileAdd libraryShare

HTMLCSSJavaScriptConsoleOutput

AccountBlog1Help

HTML

```
<!DOCTYPE html>
<html>
<head>
  <meta charset="utf-8">
  <meta name="viewport" content="width=device-width">
  <title>JS Bin</title>
</head>
<body>
  <h1>Hi!</h1>
<script src="https://unpkg.com/expect/umd/expect.min.js"></script>
</body>
</html>
```

Output

Run with JSAuto-run JS

Hi!

## toBe

```
expect(object).toBe(value, [message])
```

Asserts that `object` is strictly equal to `value` using `===`.



# Test passed:



The screenshot shows a web development environment with a top navigation bar containing 'File', 'Add library', and 'Share'. Below this is a tabbed interface with 'HTML', 'CSS', 'JavaScript', 'Console', and 'Output'. The 'JavaScript' tab is active, displaying the following code:

```
JavaScript ▾  
console.log('Starting tests');  
  
function add(a,b){  
  return a+b;  
}  
  
// Assertions  
  
//toBe: we expect the result of adding 3 and 5 to be 8  
expect(add(3,5)).toBe(8);  
  
console.log('All tests have passed');
```

The 'Console' tab is also active, showing the output of the JavaScript code:

```
Console  
"Starting tests"  
"All tests have passed"  
>
```

# Test didn't pass:

The screenshot shows a web-based JavaScript editor interface. At the top, there is a navigation bar with links for 'File', 'Add library', 'Share', and tabs for 'HTML', 'CSS', 'JavaScript', 'Console', and 'Output'. On the right side of the navigation bar, there are links for 'Account', 'Blog' (with a red notification badge), and 'Help'. The main editor area is divided into two panels. The left panel, titled 'JavaScript', contains the following code:

```
console.log('Starting tests');  
  
function add(a,b){  
  return a+b+2;  
}  
  
// Assertions  
  
//toBe: we expect the result of adding 3 and 5 to be 8  
expect(add(3,5)).toBe(8);  
  
console.log('All tests have passed');
```

The right panel, titled 'Console', shows the output of the code execution. It contains the following log entries:

- "Starting tests"
- "error"
- "Error: Expected 10 to be 8  
 at a (https://unpkg.com/expect@1.20.2/umd/expect.min.js:1:1122)  
 at t.value (https://unpkg.com/expect@1.20.2/umd/expect.min.js:1:2273)  
 at citudug.js:10:18"

The 'Run' button in the top right corner of the console panel is highlighted with a blue border.

This is ok for testing some simple functions.. But what about our React app?

# Outline

- Introduction to Testing
- Simple testing in JS
- **Testing with Karma**
  - Configuring Karma
  - Karma demo: Simple test
  - Karma demo: Weather Mood
  - Karma demo: Timer App
- Conclusions

# Karma

- Karma is a JavaScript test runner created by the AngularJS team. Karma uses your test framework to run tests in various environments of your choice



# Karma

- Karma has several configurations for our tests and can run code using several JS testing frameworks such as Mocha, Jasmine and QUnit

# Karma

- Karma has several configurations for our tests and can run code using several JS testing frameworks such as Mocha, Jasmine and QUnit
- For this lab, we are going to focus on Mocha

# Mocha framework

- Mocha is a JavaScript test framework featuring browser support, asynchronous testing, test coverage reports, and use of any assertion library





# Karma + Mocha

- Use Karma to configure tests for application
- Use Mocha framework tools to write the tests
- Use Karma to run the tests

# Outline

- Introduction to Testing
- Simple testing in JS
- Testing with Karma
  - **Configuring Karma**
  - Karma demo: Simple test
  - Karma demo: Weather Mood
  - Karma demo: Timer App
- Conclusions

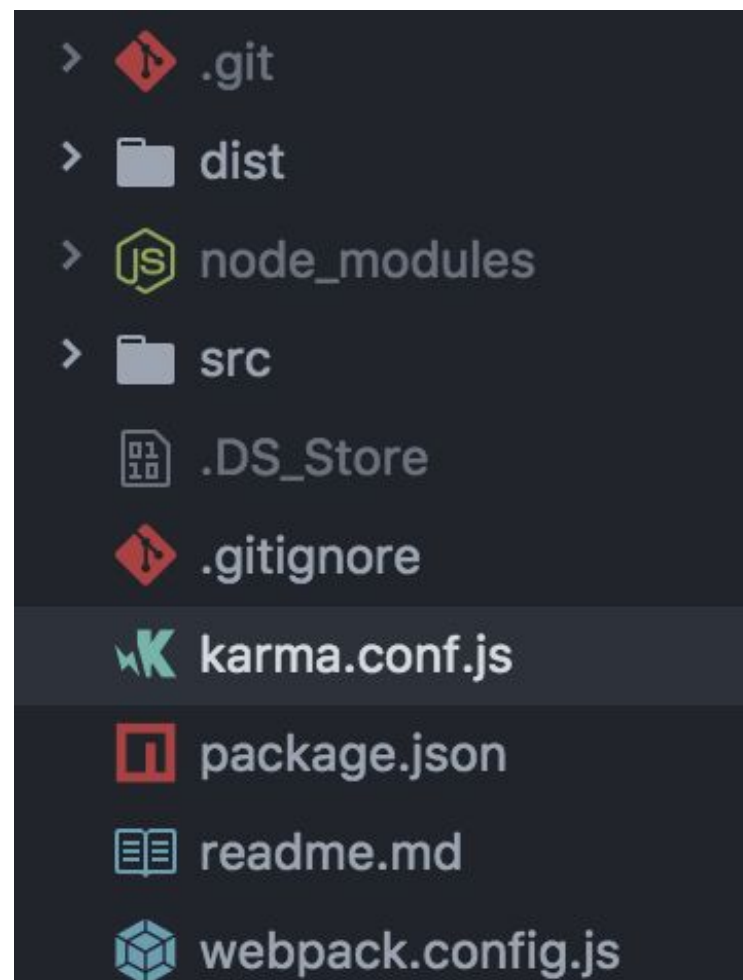
# Configuring Karma

- First, install required dependencies:

```
npm install mocha karma karma-chrome-launcher karma-mocha  
karma-mocha-reporter karma-sourcemap-loader karma-webpack  
expect react-addons-test-utils --save-dev
```

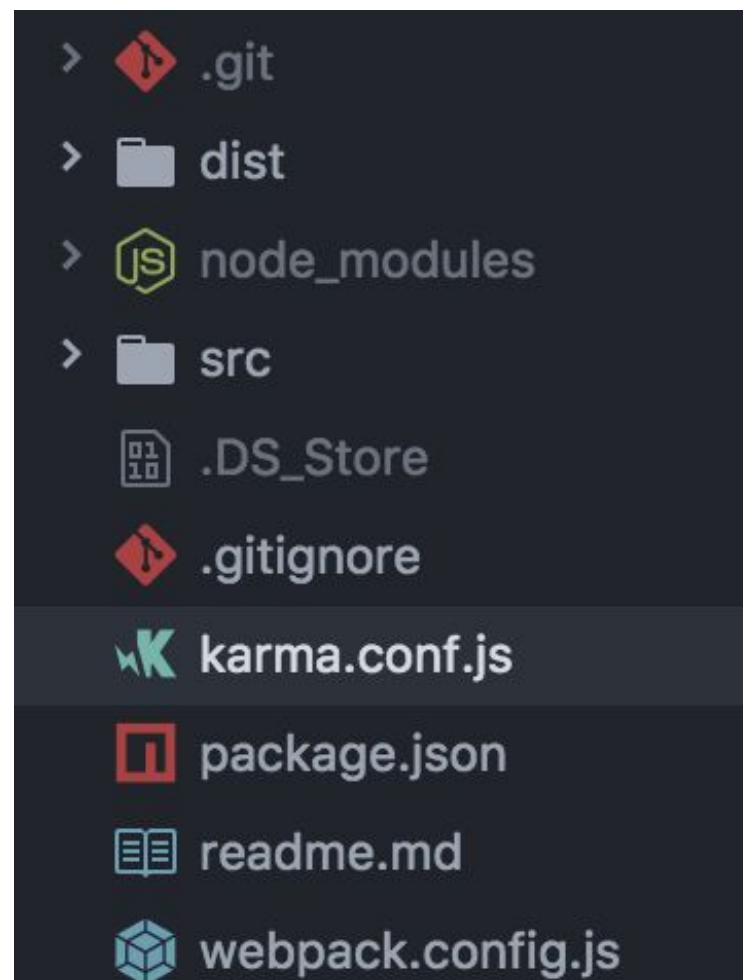
# Configuring Karma

- Create config file for Karma named “karma.config.js” – same folder as package.json and webpack.config.js



# Configuring Karma

- Create config file for Karma named “karma.conf.js”
  - same folder as package.json and webpack.config.js



```
var webpackConfig = require('./webpack.config.js');  
  
module.exports = function (config){  
  config.set({  
    // Which browsers we want to run on  
    browsers: ['Chrome'],  
    singleRun: true,  
    // Karma test runner will use mocha to run tests  
    frameworks: ['mocha'],  
    // Need to load common bundle files before the tests  
    // Run all files that end in .test.jsx in folders under app/tests  
    // files: ['dist/vendor.bundle.js', 'dist/index.bundle.js', 'src/tests/**/*.test.jsx'],  
    files: ['src/tests/**/*.test.jsx'],  
    // Here "webpack" enables to use webpack modules on the tests  
    // "sourcemap" enables to use the .jsx files and not the bundles  
    preprocessors: {  
      'src/tests/**/*.test.jsx': ['webpack', 'sourcemap']  
    },  
    // How to format the result of the tests  
    reporters: ['mocha'],  
    // Timeout condition for failure  
    client: {  
      timeout: '5000'  
    },  
    webpack: webpackConfig,  
    webpackServer: {  
      noInfo: true  
    }  
  });  
};
```

# Configuring Karma

- In package.json, change the script for “test” with “karma start”

```
"scripts": {  
  "build": "webpack -p",  
  "watch": "webpack -w",  
  "start": "webpack-dev-server",  
  "test": "echo \"Error: no test specified\" && exit 1",  
},
```



```
"scripts": {  
  "build": "webpack -p",  
  "watch": "webpack -w",  
  "start": "webpack-dev-server",  
  "test": "karma start",  
},
```

# Configuring Karma

- If you are having problems with the tests, it might be cached code
- To solve this, inside webpack.config.js, modify plugins:

```
plugins: [new webpack.optimize.CommonsChunkPlugin({name: 'vendor',  
filename: 'vendor.bundle.js', minChunks: 2})],
```



```
plugins: [],
```

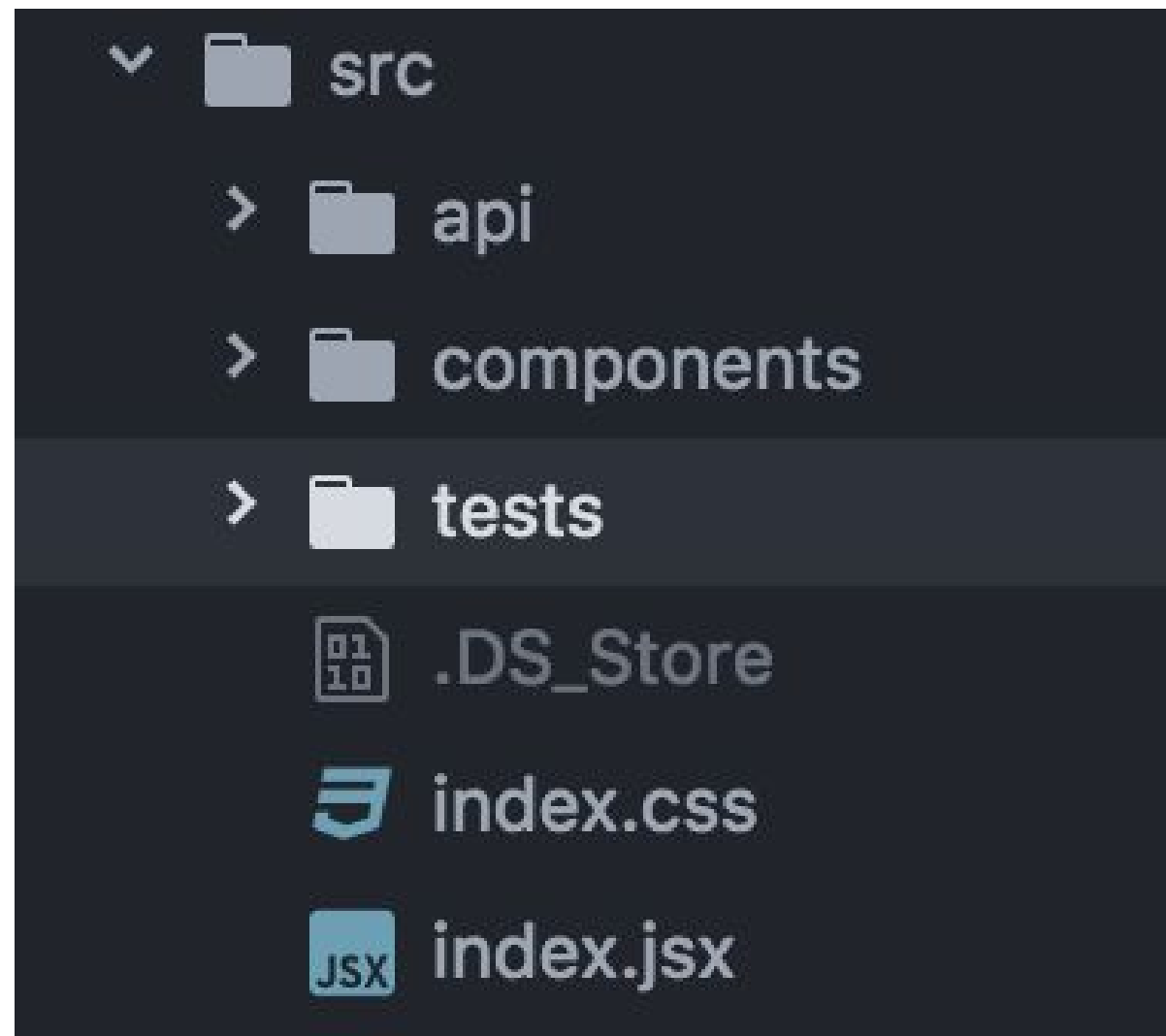


# Configuring Karma

- CommonChunks is a feature from webpack that caches code in chunks for faster reloading

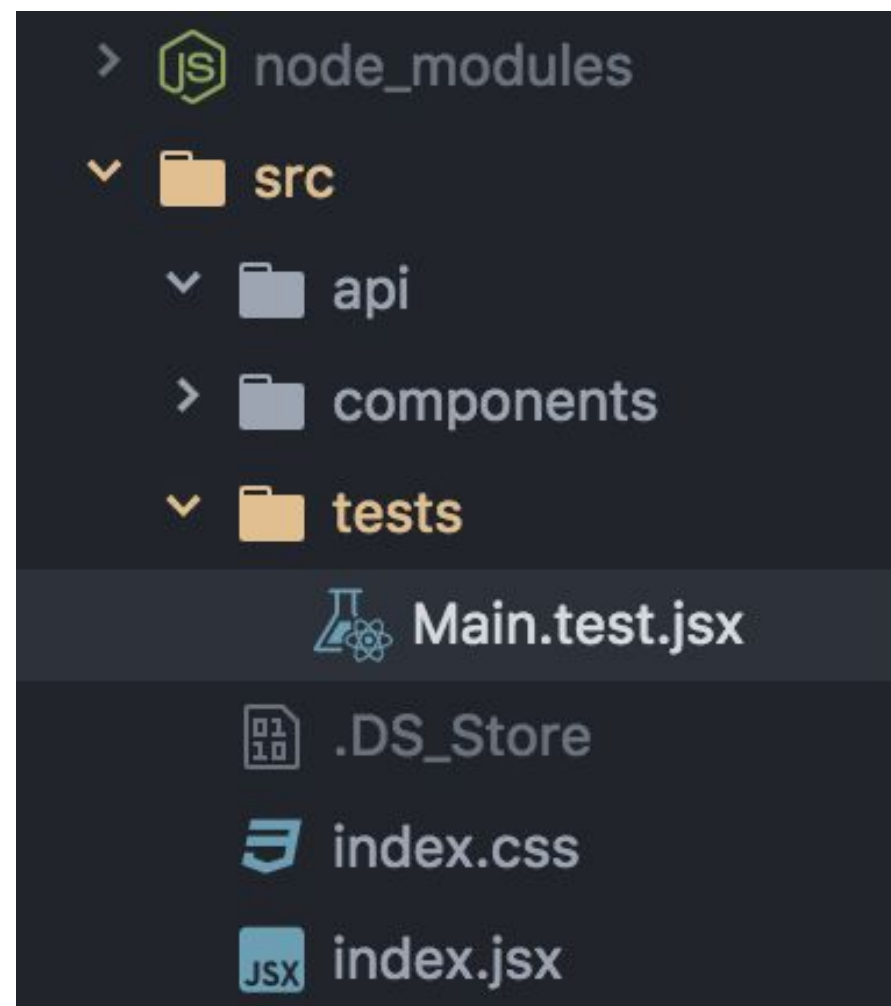
# Configuring Karma

- Create folder src/tests where our test files will be



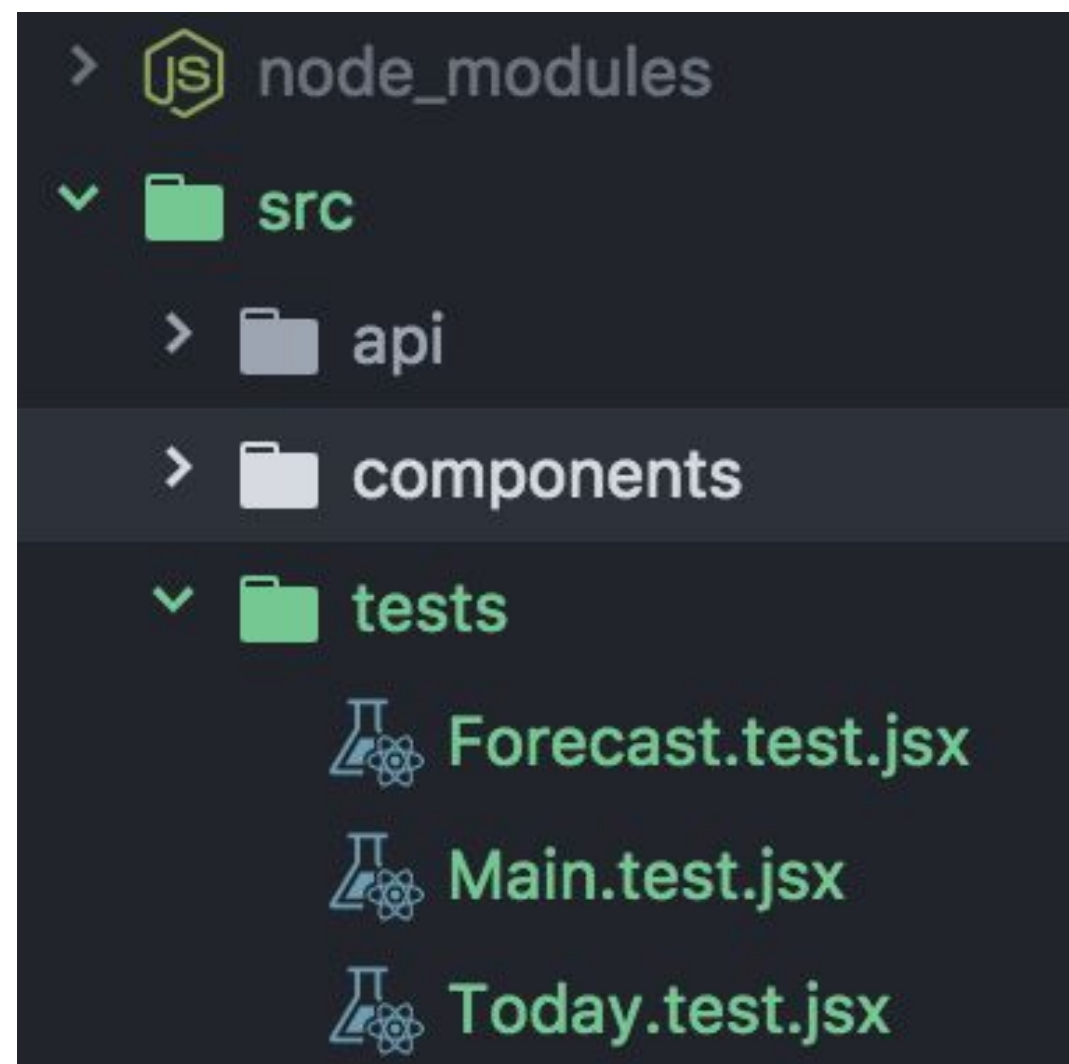
# Configuring Karma

- To test the Main.jsx component, we create a Main.test.jsx file in that folder



# Configuring Karma

- For other components that we want to test, we create similar files inside the tests folder



# Outline

- Introduction to Testing
- Simple testing in JS
- Testing with Karma
  - Configuring Karma
  - Karma demo: Simple test
  - Karma demo: Weather Mood
  - Karma demo: Timer App
- Conclusions

# Karma demo: Simple test

- Inside Main.test.jsx:

```
var expect = require('expect');  
  
// describe just breaks the tests in groups  
describe('Main', () => {  
  it('should properly run tests', () => {  
    expect(1).toBe(1);  
  });  
});
```

# Karma demo: Simple test

- Run tests on terminal with “npm test”
- Output:

```
START:
(node:35028) DeprecationWarning: loaderUtils.parseQuery() received a non-string value which can be problematic
parseQuery() will be replaced with getOptions() in the next major version of loader-utils.
05 04 2017 16:18:17.940:INFO [karma]: Karma v1.5.0 server started at http://0.0.0.0:9876/
05 04 2017 16:18:17.942:INFO [launcher]: Launching browser Chrome with unlimited concurrency
05 04 2017 16:18:17.949:INFO [launcher]: Starting browser Chrome
05 04 2017 16:18:19.194:INFO [Chrome 56.0.2924 (Mac OS X 10.12.4)]: Connected on socket HL74UbWLb1uyFvhzAAA
Main
  ✓ should properly run tests

Finished in 0.011 secs / 0.001 secs @ 16:18:19 GMT+0800 (CST)

SUMMARY:
✓ 1 test completed
vrsj@VRSJ-MacBook-Pro ~/Documents/Tsinghua-NotSync/DataLab/SoftwareStudio/lab-tests master !
```

# Outline

- Introduction to Testing
- Simple testing in JS
- Testing with Karma
  - Configuring Karma
  - Karma demo: Simple test
  - **Karma demo: Weather Mood**
  - Karma demo: Timer App
- Conclusions



# Karma demo: Weather mood

- We can write tests for the components of the weather app
- Example: Today.test.jsx

```
var React = require('react');  
var ReactDOM = require('react-dom');  
var expect = require('expect');  
var $ = require('jQuery');  
var TestUtils = require('react-addons-test-utils');  
  
import Today from 'components/Today.jsx';
```

# Karma demo: Weather mood

- Test if today component exists:

```
describe('Today', () => {  
  //Check if Today component exists  
  it('should exist', () => {  
    expect(Today).toExist();  
  });  
});
```

# Karma demo: Weather mood

- Test if today component renders default city:

```
// Tests for rendering of today~  
describe('render', () => {~  
  • it('should render today to output with default city', () => {~  
    • var today = TestUtils.renderIntoDocument(<Today/>);~  
    • expect(today.state.city).toBe('Hsinchu');~  
  • });~  
});~
```

# Karma demo: Weather mood

- Test if today component renders non default city:

```
describe('render', () => {  
  it('should render today to output with non default city', () => {  
    var today = TestUtils.renderIntoDocument(<Today />);  
    today.setState({city: 'Paris'});  
    expect(today.state.city).toBe('Paris');  
  });  
});
```

# Karma demo: Weather mood

- Test if today component renders metric unit:

```
describe('render', () => {  
  it('should render today to output with metric unit', () => {  
    var today = TestUtils.renderIntoDocument(<Today />);  
    today.setState({unit: 'imperial'});  
    expect(today.state.unit).toBe('imperial');  
  });  
});
```



# Karma demo: Weather mood

- Output of tests:

```
START:
(node:45830) DeprecationWarning: loaderUtils.parseQuery() received a non-string value which c
issues/56
parseQuery() will be replaced with getOptions() in the next major version of loader-utils.
05 04 2017 17:54:15.507:INFO [karma]: Karma v1.5.0 server started at http://0.0.0.0:9876/
05 04 2017 17:54:15.509:INFO [launcher]: Launching browser Chrome with unlimited concurrency
05 04 2017 17:54:15.516:INFO [launcher]: Starting browser Chrome
05 04 2017 17:54:16.834:INFO [Chrome 56.0.2924 (Mac OS X 10.12.4)]: Connected on socket vyvAi
05 04 2017 17:54:17.434:WARN [web-server]: 404: /images/w-na.png
Main
  ✓ should properly run tests
Today
  ✓ should exist
  render
    ✓ should render today to output with default city
    ✓ should render today to output with non default city
    ✓ should render today to output with metric unit

Finished in 0.008 secs / 0.077 secs @ 17:54:17 GMT+0800 (CST)

SUMMARY:
```

# Outline

- Introduction to Testing
- Simple testing in JS
- Testing with Karma
  - Configuring Karma
  - Karma demo: Simple test
  - Karma demo: Weather Mood
  - Karma demo: Timer App
- Conclusions

# Karma demo: Timer App





# Karma demo: Timer App

## Countdown App



Pause

Clear

# Karma demo: Timer App



# Karma demo: Timer App

- What can we test in this app?

# Karma demo: Timer App

- Test if seconds are rendered correctly

```
describe('formatSeconds', () => {  
  it('should format seconds', () => {  
    var clock = TestUtils.renderIntoDocument(<Clock/>);  
    var seconds = 615;  
    var expected = '10:15';  
    var actual = clock.formatSeconds(seconds);  
  
    expect(actual).toBe(expected);  
  });  
});
```

# Karma demo: Timer App

- Test if seconds are rendered correctly

```
it('should format seconds when min/sec are less than 10', () => {  
  var clock = TestUtils.renderIntoDocument(<Clock/>);  
  var seconds = 61;  
  var expected = '01:01';  
  var actual = clock.formatSeconds(seconds);  
  
  expect(actual).toBe(expected);  
});
```

# Karma demo: Timer App

- Rendering of controls

```
describe('render', () => {  
  it('should render pause when started', () => {  
    var controls = TestUtils.renderIntoDocument(<Controls countdownStatus="started" />);  
    var $el = $(ReactDOM.findDOMNode(controls));  
    var $pauseButton = $el.find('button:contains(Pause)');  
  
    expect($pauseButton.length).toBe(1);  
  });  
  
  it('should render start when paused', () => {  
    var controls = TestUtils.renderIntoDocument(<Controls countdownStatus="paused" />);  
    var $el = $(ReactDOM.findDOMNode(controls));  
    var $pauseButton = $el.find('button:contains(Start)');  
  
    expect($pauseButton.length).toBe(1);  
  });  
});
```

# Karma demo: Timer App

- Check if app functionalities are working (start/stop)

```
it('should start timer on started status', (done) => {  
  var timer = TestUtils.renderIntoDocument(<Timer />);  
  
  timer.handleStatusChange('started');  
  expect(timer.state.count).toBe(0);  
  
  setTimeout(() => {  
    expect(timer.state.timerStatus).toBe('started');  
    expect(timer.state.count).toBe(1);  
    done();  
  }, 1001);  
});
```

# Karma demo: Timer App

- Countdown functions (remember errors countdown from nightmare mode in color game?)

```
it('should never set count less than zero', (done) => {  
  var countdown = TestUtils.renderIntoDocument(<Countdown />);  
  countdown.handleSetCountdown(1);  
  
  setTimeout(() => {  
    expect(countdown.state.count).toBe(0);  
    done();  
  }, 3001)  
});
```



# Outline

- Introduction to Testing
- Simple testing in JS
- Testing with Karma
  - Configuring Karma
  - Karma demo: Simple test
  - Karma demo: Weather Mood
  - Karma demo: Timer App
- Conclusions

# Conclusions

- For this lab we used Karma runner with Mocha, but you could try other testing frameworks
- For each new language you learn there are testing frameworks that you can use to automate this work and reduce errors
- We are humans and sometimes we can forget to test something manually, it's better to let your testing framework do this work for you

# Today's mission

- Clone the Weather Mood repository and explore the written tests
- Clone the Timer App repository and explore the written tests
- Try writing tests for these applications or your own application
- Optional assignment!

# Optional assignment

- Bonus 10 points for those who want it
- This assignment will be due in 1 week (04-13) at 23:59

# Optional assignment

- Create a fork for the hello-react repository
- Create at least 1 test for this project
  - Not just `expect(1).toBe(1)`
  - Not just `expect().toExist()`
  - Actually test some component (Try to test some function)

# Optional assignment

- The created tests must run correctly
- Submit your merge request

# References

- [Karma](#)
- [Mocha](#)
- [Karma Tutorial - Unit Testing JavaScript](#)
- [Client side testing with Karma and Mocha](#)