

# Tooling and Testing

SWE 432, Fall 2019

Web Application Development

# Quiz

Go to:

[b.socrative.com](https://b.socrative.com), Click student login

Room name: SWE432

Student Name: Your G-number (Including the G)

**Reminder:** Survey can only be completed if you are in class. If you are not in class and do it you will be referred directly to the honor code board, no questions asked, no warning.

# Review: JSON: JavaScript Object Notation

Open standard format for transmitting *data* objects.

No functions, only key / value pairs

Values may be other objects or arrays

```
var profHacker = {  
  firstName: "Alyssa",  
  lastName: "P Hacker",  
  teaches: "SWE 432",  
  office: "ENGR 6409",  
  fullName: function(){  
    return this.firstName + " " + this.lastName;  
  }  
};
```

**Our Object**

```
var profHacker = {  
  firstName: "Alyssa",  
  lastName: "P Hacker",  
  teaches: "SWE 432",  
  office: "ENGR 6409",  
  fullName: {  
    firstName: "Alyssa",  
    lastName: "P Hacker"}  
};
```

**JSON Object**

# Logistics

- HW1 due next Monday before class

# Today

- Web Development Tools
- What's behavior driven development and why do we want it?
- Some tools for testing web apps - focus on Jest

# An (older) way to export modules

- Prior to ES6, was no language support for exposing modules.
- Instead did it with libraries (e.g., node) that handled exports
- Works similarly: declare what functions / classes are publicly visible, import classes

- Syntax:

In the file exporting a function or class sum:

```
module.exports = sum;
```

In the file importing a function or class sum:

```
const sum = require('./sum');
```

Where sum.js is the name of a file which defines sum.

# Options for executing JavaScript

- Browser
  - Pastebin—useful for debugging & experimentation
- Outside of the browser (focus for now)
  - node.js—runtime for JavaScript

# Demo: Pastebin

```
var course = { name: 'SWE 432' };  
  
console.log('Hello ' + course.name + '!');
```

<https://jsbin.com/?js,console>



# Node.js

- Node.js is a *runtime* that lets you run JS outside of a browser
- We're going to write backends with Node.js
- Download and install it: <https://nodejs.org/en/>
- We recommend LTS (LTS -> Long Term Support, designed to be super stable)

# Demo: Node.js

```
var course = { name: 'SWE 432' };  
console.log('Hello ' + course.name + '!');
```

# Node Package Manager

# Working with libraries

“The old way”

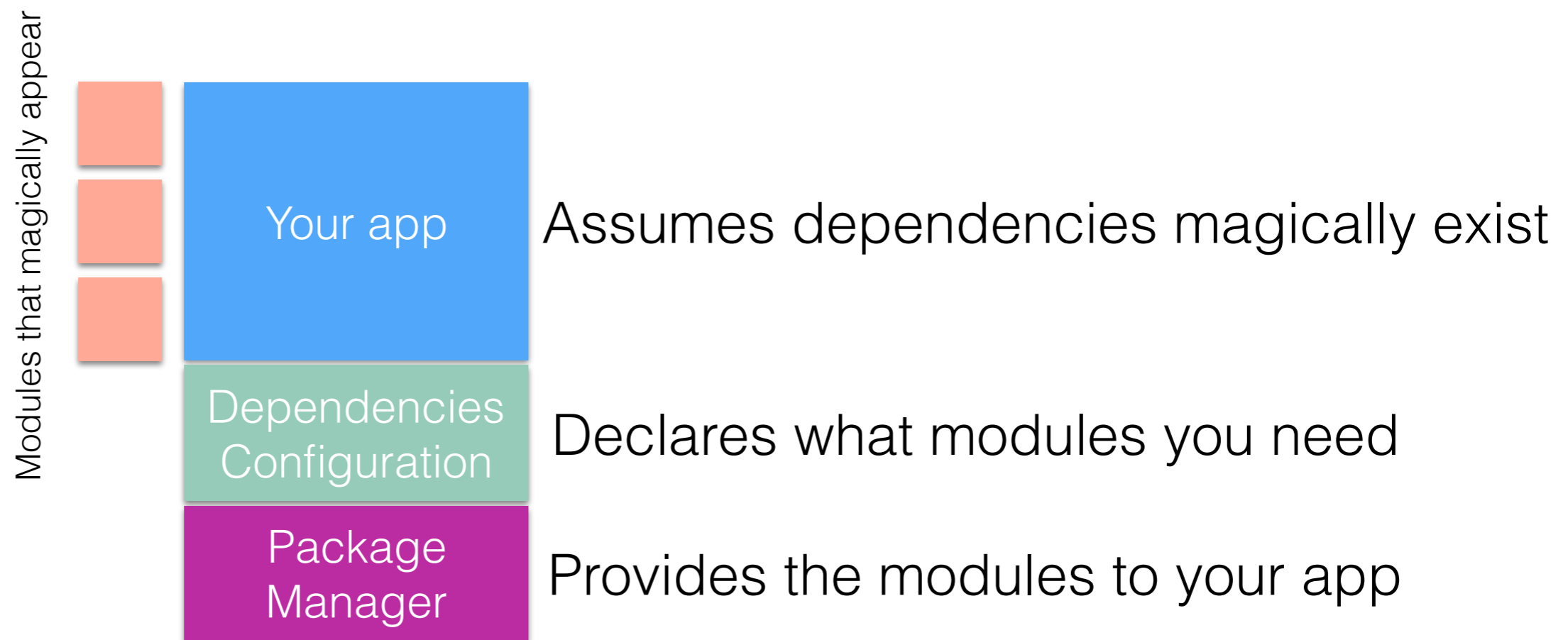


```
<script src="https://fb.me/react-15.0.0.js"></script>  
<script src="https://fb.me/react-dom-15.0.0.js"></script>  
<script src="https://cdnjs.cloudflare.com/ajax/libs/babel-core/5.8.34/browser.min.js"></script>
```

- What’s wrong with this?
  - No standard format to say:
    - What’s the name of the module?
    - What’s the version of the module?
    - Where do I find it?
  - Ideally: Just say “Give me React 15 and everything I need to make it work!”

# A better way for modules

- Describe what your modules are
- Create a central repository of those modules
- Make a utility that can automatically find and include those modules



# NPM: Not an acronym, but the Node Package Manager

- Bring order to our modules and dependencies

- Declarative approach:

- “My app is called helloworld”
- “It is version 1”
- You can run it by saying “node index.js”
- “I need express, the most recent version is fine”

- Config is stored in json - specifically package.json

**Generated by npm commands:**

```
{
  "name": "helloworld",
  "version": "1.0.0",
  "description": "",
  "main": "index.js",
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1"
  },
  "author": "",
  "license": "ISC",
  "dependencies": {
    "express": "^4.14.0"
  }
}
```

# Installing packages with NPM

- ``npm install <package> --save`` will download a package and add it to your `package.json`
- ``npm install`` will go through all of the packages in `package.json` and make sure they are installed/up to date
- Packages get installed to the ``node_modules`` directory in your project

# Using NPM

- Your “project” is a directory which contains a special file, package.json
- Everything that is going to be in your project goes in this directory
- Step 1: Create NPM project  
`npm init`
- Step 2: Declare dependencies  
`npm install <packagename> --save`
- Step 3: Use modules in your app  
`var myPkg = require(“packagename”)`
- Do NOT include node\_modules in your git repo! Instead, just do  
`npm install`
  - This will download and install the modules on your machine given the existing config!

<https://docs.npmjs.com/index>



# NPM Scripts

- Scripts that run at specific times.
- For starters, we'll just worry about *test* scripts

<https://docs.npmjs.com/misc/scripts>

```
{
  "name": "starter-node-react",
  "version": "1.1.0",
  "description": "a starter project structure for react-app",
  "main": "src/server/index.js",
  "scripts": {
    "start": "babel-node src/server/index.js",
    "build": "webpack --config config/webpack.config.js",
    "dev": "webpack-dev-server --config config/webpack.config.js --
devtool eval --progress --colors --hot --content-base dist/"
  },
  "repository": {
    "type": "git",
    "url": "git+https://github.com/wwsun/starter-node-react.git"
  },
  "author": "Weiwei SUN",
  "license": "MIT",
  "bugs": {
    "url": "https://github.com/wwsun/starter-node-react/issues"
  },
  "homepage": "https://github.com/wwsun/starter-node-react#readme",
  "dependencies": {
    "babel-cli": "^6.4.5",
    "babel-preset-es2015-node5": "^1.1.2",
    "co-views": "^2.1.0",
    "history": "^2.0.0-rc2",
    "koa": "^1.0.0",
    "koa-logger": "^1.3.0",
    "koa-route": "^2.4.2",
    "koa-static": "^2.0.0",
    "react": "^0.14.0",
    "react-dom": "^0.14.0",
    "react-router": "^2.0.0-rc5",
    "swig": "^1.4.2"
  },
  "devDependencies": {
    "babel-core": "^6.1.2",
    "babel-loader": "^6.0.1",
    "babel-preset-es2015": "^6.3.13",
    "babel-preset-react": "^6.1.2",
    "webpack": "^1.12.2",
    "webpack-dev-server": "^1.14.1"
  },
  "babel": {
    "presets": [
      "es2015-node5"
    ]
  }
}
```

Demo: NPM

# Unit Testing

- Unit testing is testing some program unit in isolation from the rest of the system (which may not exist yet)
- Usually the programmer is responsible for testing a unit during its implementation (even though this violates the rule about a programmer not testing own software)
- Easier to debug when a test finds a bug (compared to full-system testing)

# Integration Testing

- Motivation: Units that worked in isolate may not work in combination
- Performed after all units to be integrated have passed all unit tests
- Reuse unit test cases that cross unit boundaries (that previously required stub(s) and/or driver standing in for another unit)

# Unit vs Integration Tests



# Writing good tests

- How do we know when we have tested “enough”?
  - Did we test all of the features we created?
  - Did we test all possible values for those features?

# Behavior Driven Development

- Establish *specifications* that say what an app should do
- We write our spec *before* writing the code!
- Only write code if it's to make a spec work
- Provide a mapping between those specifications, and some observable application functionality
- This way, we can have a clear map from specifications to tests

# Investment Tracker

- Users make investments by entering a ticker symbol, number of shares, and the price that the user paid per share
- Once the investment is inputted, the user can see the current status of their investments
- How do we test this?

The image shows two screenshots of an investment tracker application. The top screenshot displays the input form with the following values: Symbol: PETO, Shares: 100, and Share price: 35. The bottom screenshot shows the application's state after adding investments. It features a list of two investments: AOUE with a 101.80% gain and PETO with a -42.34% loss. Each investment card includes a 'remove' button.

Symbol:	Shares:	Share price:	Action
PETO	100	35	Add
	0	0	Add

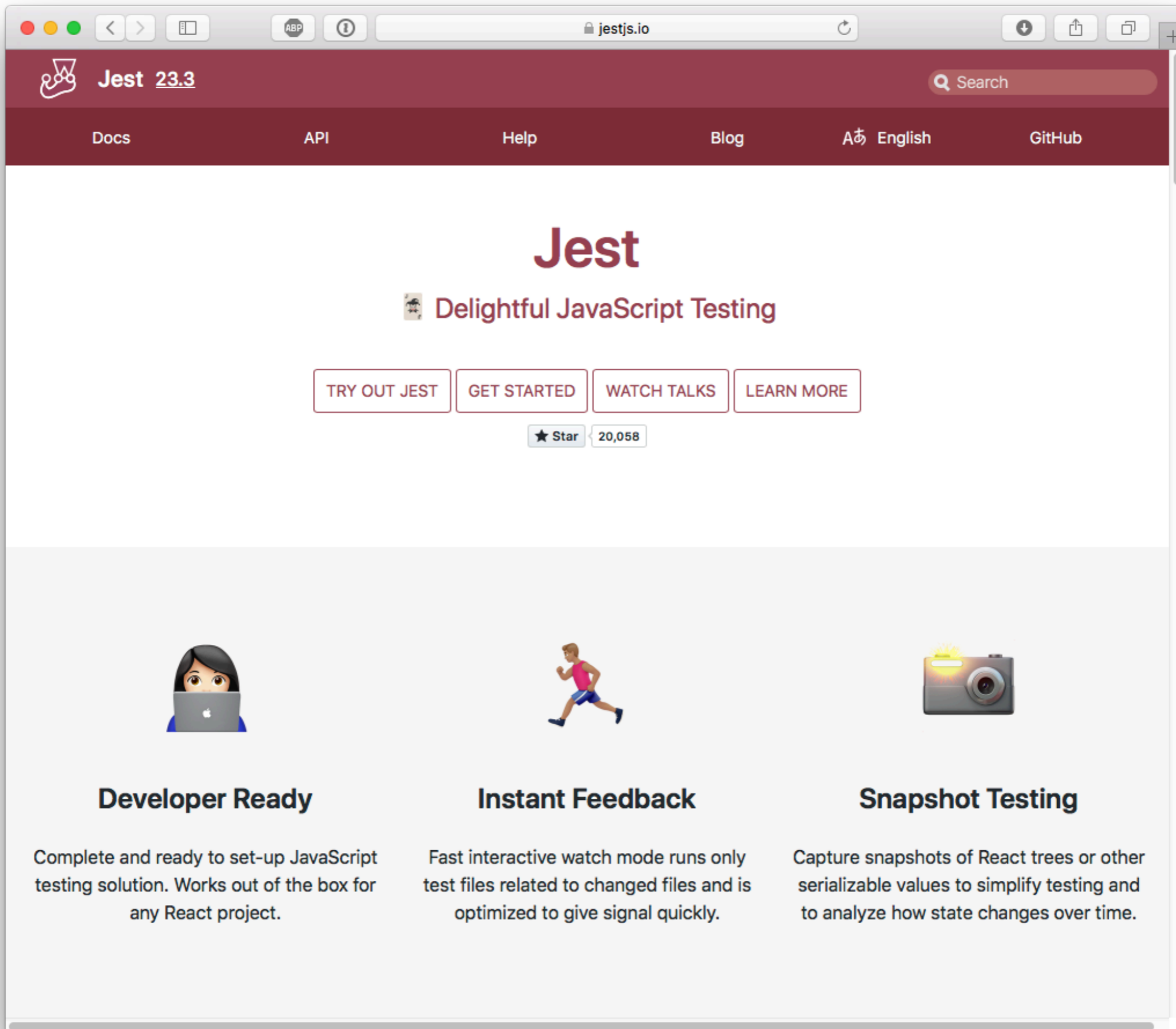
  

Symbol	Gain/Loss	Action
AOUE	101.80%	remove
PETO	-42.34%	remove



# Investment Tracker

- What's an investment for our app?
  - Given an investment, it:
    - Should be of a stock
    - Should have the invested shares quantity
    - Should have the share paid price
    - Should have a current price
  - When its current price is higher than the paid price:
    - It should have a positive return of investment
    - It should be a good investment



# Jest

 Delightful JavaScript Testing

- [TRY OUT JEST](#)
- [GET STARTED](#)
- [WATCH TALKS](#)
- [LEARN MORE](#)

★ Star 20,058



## Developer Ready

Complete and ready to set-up JavaScript testing solution. Works out of the box for any React project.



## Instant Feedback

Fast interactive watch mode runs only test files related to changed files and is optimized to give signal quickly.



## Snapshot Testing

Capture snapshots of React trees or other serializable values to simplify testing and to analyze how state changes over time.

# Jest lets you specify behavior in *specs*

- Specs are written in JS
- Key functions:
  - `describe`, `test`, `expect`
- **Describe** a high level scenario by providing a name for the scenario and function(s) that contains some **tests** by saying what you **expect** it to be
- Example:

```
describe("Alyssa P Hacker tests", () => {  
  test("Calling fullName directly should always work", () => {  
    expect(profHacker.fullName()).toEqual("Alyssa P Hacker");  
  });  
}
```

# Writing Specs

- Can specify some code to run before or after checking a spec

```
var profHacker;  
beforeEach(() => {  
  profHacker = {  
    firstName: "Alyssa",  
    lastName: "P Hacker",  
    teaches: "SWE 432",  
    office: "ENGR 6409",  
    fullName: function () {  
      return this.firstName + " " + this.lastName;  
    }  
  };  
});
```

# Making it work

- Add jest to your project (npm install --save-dev jest)
- Configure NPM to use jest for test in package.json

```
"scripts": {  
  "test": "jest"  
},
```

- For file x.js, create x.test.js
- Run npm test

# Multiple Specs

- Can have as many tests as you would like

```
test("Calling fullName directly should always work", () => {
    expect(profHacker.fullName()).toEqual("Alyssa P Hacker");
});

test("Calling fullName without binding but with a function ref is undefined",
() => {
    var func = profHacker.fullName;
    expect(func()).toEqual("undefined undefined");
});

test("Calling fullName WITH binding with a function ref works", () => {
    var func = profHacker.fullName;
    func = func.bind(profHacker);
    expect(func()).toEqual("Alyssa P Hacker");
});

test("Changing name changes full name", ()=>{
    profHacker.firstName = "Dr. Alyssa";
    expect(profHacker.fullName()).toEqual("Dr. Alyssa P Hacker");
})
```

# Nesting Specs

- “When its current price is higher than the paid price:
  - It should have a positive return of investment
  - It should be a good investment”
- How do we describe that?

```
describe("when its current price is higher than the paid price", function() {  
  beforeEach(function() {  
    stock.sharePrice = 40;  
  });  
  test("should have a positive return of investment", function() {  
    expect(investment.roi()).toBeGreaterThan(0);  
  });  
  test("should be a good investment", function() {  
    expect(investment.isGood()).toBeTruthy();  
  });  
});
```

# Matchers

- How does Jest determine that something is what we expect?

```
expect(investment.roi()).toBeGreaterThan(0);  
expect(investment).isGood().toBeTruthy();  
expect(investment.shares).toEqual(100);  
expect(investment.stock).toBe(stock);
```

- These are “matchers” for Jest - that compare a given value to some criteria
- Basic matchers are built in:
  - toBe, toEqual, toContain, toBeNaN, toBeNull, toBeUndefined, >, <, >=, <=, !=, regular expressions
- Can also define your own matcher



# Matchers

```
test('null', () => {  
  const n = null;  
  expect(n).toBeNull();  
  expect(n).toBeDefined();  
  expect(n).not.toBeUndefined();  
});
```

```
const shoppingList = [  
  'diapers',  
  'kleenex',  
  'trash bags',  
  'paper towels',  
  'beer',  
];
```

```
test('the shopping list has beer on it', () => {  
  expect(shoppingList).toContain('beer');  
  expect(new Set(shoppingList)).toContain('beer');  
});
```

Demo: Jest

# Exercise: JEST

- Write a factorial program that computes the factorial of  $x$ .
- Write a JEST test suite that ensure that this function works correctly.