SWPP Practice Session #5
Frontend Testing

2022 Oct 5

Announcement

- Github team and repository for each team are set up. Please check your emails and accept the invitation to get access to the team (*swpp2022-teamX*) and repo.
- Make an appointment with TA for Sprint Meeting #1. The sooner the better!
  - Team 1-7: Jaewoo Maeng (available time slot)
  - Team 8-14: Junyeol Ryu (available time slot)
  - Team 15-20: Jongsun Yun (available time slot)

Clone Repo

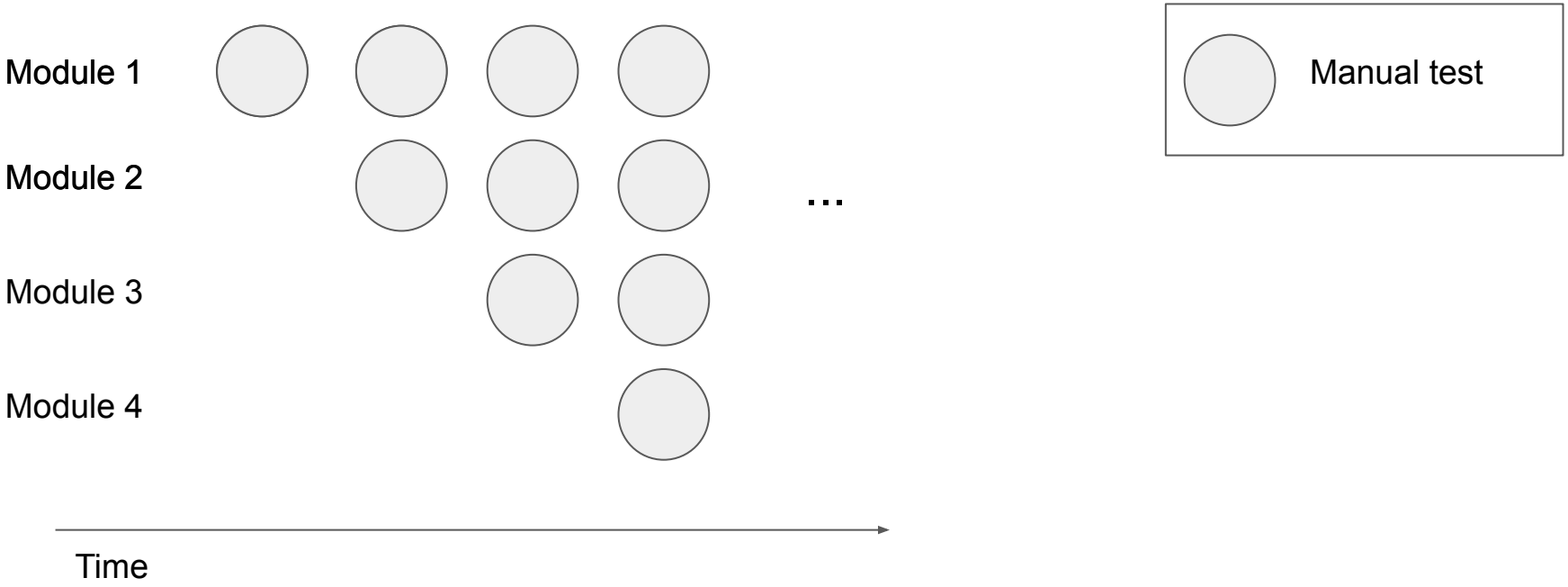
- **Please fork and clone this repository**
- https://github.com/swpp22fall-practice-sessions/swpp-p5-react-testing
- We have checkpoint branches ready. If you're in trouble and can't keep up, you can jump to the following branches with `$ git checkout {branch_name}`

Why testing?

Misconceptions about testing as a beginner

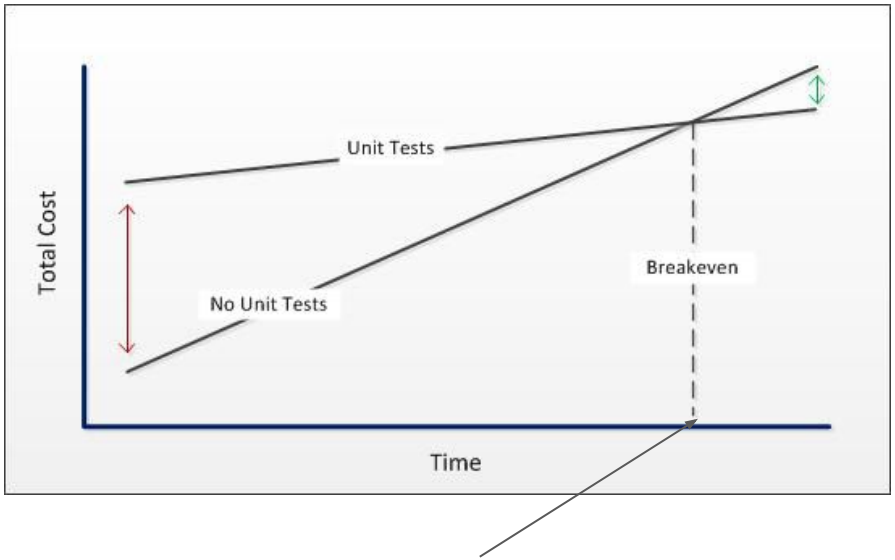
1. Manual testing is straightforward and handy. Then, why should I write tests?
2. We have type system (or maybe type hint with good IDE). I think unit test is unnecessary.
3. Managing a bunch of test cases looks hard. I’m not confident :(

Your hands can’t cover all as codebase grows



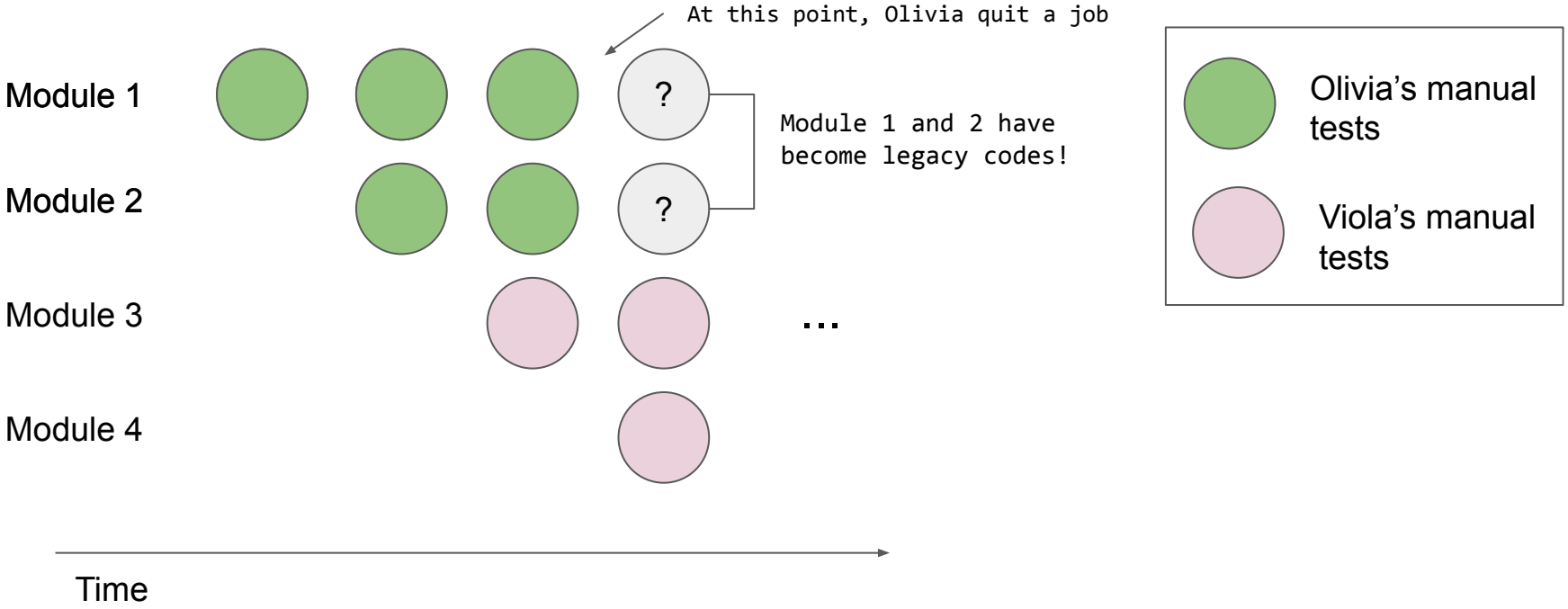
Your hands don’t scale:
you have to test O(N^2) cases as codebase grows.

Your hands can’t cover all as codebase grows



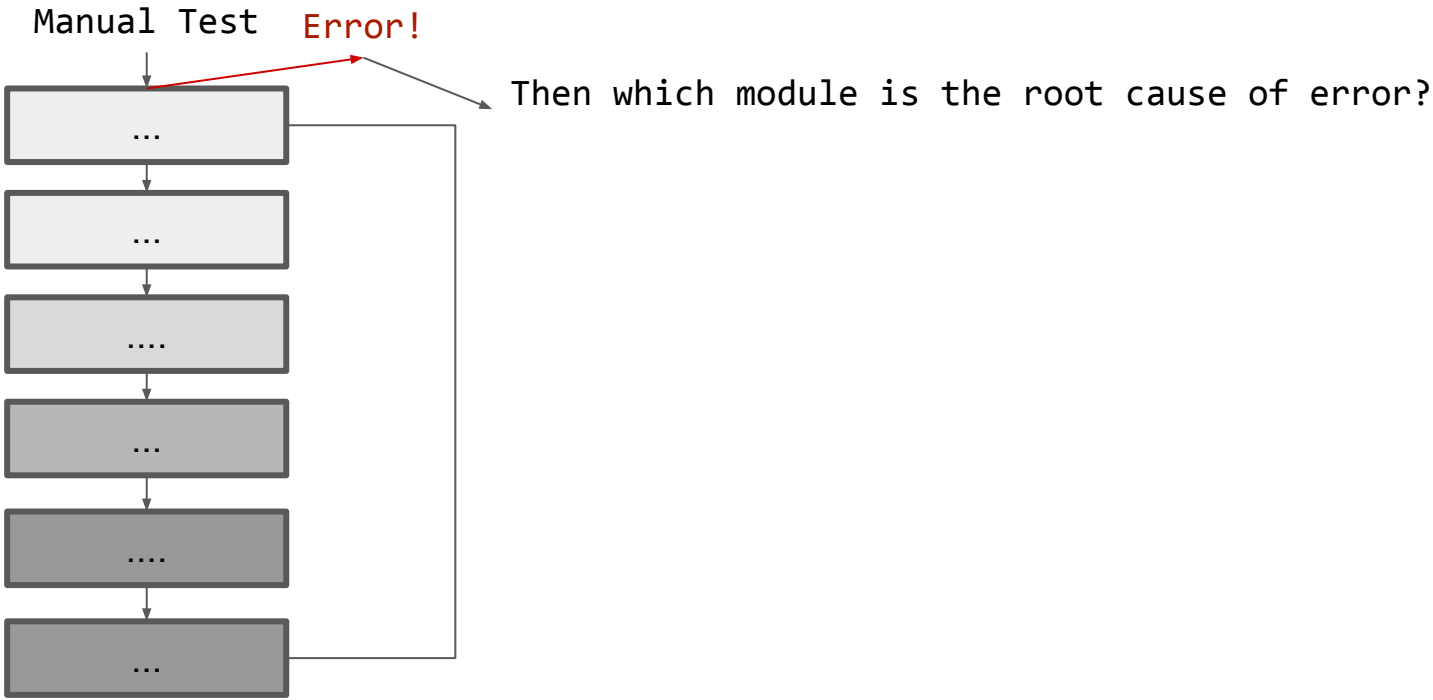
This breakeven comes much earlier than you’d expect, even in a simple toy application.

You can’t even touch other’s code without tests



Viola has no idea about how to test Olivia’s features.

You can’t spot the root cause easily without unit test



Type system will not save you

```
def add(x: int, y: int) -> int:
    return x + y

def g():
    add(1, "2")
```



```
def add(x: int, y: int) -> int:
    return 0
```

Managing test cases is easier than it looks

- Suppose you've just altered codebase:
- Then, run tests and see the results.
  - If all test passes, you're good to go.
  - If some test fails, examine both code and test cases and try to make it green.
    - If there are mistakes in your code, then revise your code.
    - If the failed test is not logically valid any more, revise the failing test case.
      This happens if your modification implies major spec change.
- Repeat the above process until all is **green**.
- Don't forget to cover your new modules with new tests.

Why Should We Do Testing?

1. Guard against changes that break existing code.
    a. You can safely modify other's code.
    b. You can safely modify *your* code.
2. Reveals mistakes in implementation.
    a. Tests shine a harsh light on the code from many angles.
3. Reveals mistakes in *design*.
    a. When a part of the application seems hard to test, the root cause is often a design flaw.
    b. You can cure the design flaw now rather than later when it becomes expensive to fix.

Basic concepts of testing

Typical structure of testings

- Test Suite: “I’m going to describe tests on `add(a, b)`”
- Test Cases(It should ~)
  - “**It** should add up two numbers”
  - “**It** should concatenate two strings”
- Assertions/Expectations
  - I expect `add(2, 2)` to equal to `4` and I expect `add(2, -2)` to equal to `0`
  - I expect `add(“pine”, “apple”)` to equal to “`pineapple`”

Basic structure of testings (Cont.)

Describe-It based

```
describe("add(x, y)", function() {
  it("should add up two numbers", function () {
    expect(add(2, 2)).toEqual(2);
    expect(add(2, -2)).toEqual(0);
  });
  it("should concatenate two strings", function () {
    expect(add("pine",
"apple")).toEqual("pineapple");
  });
});
```

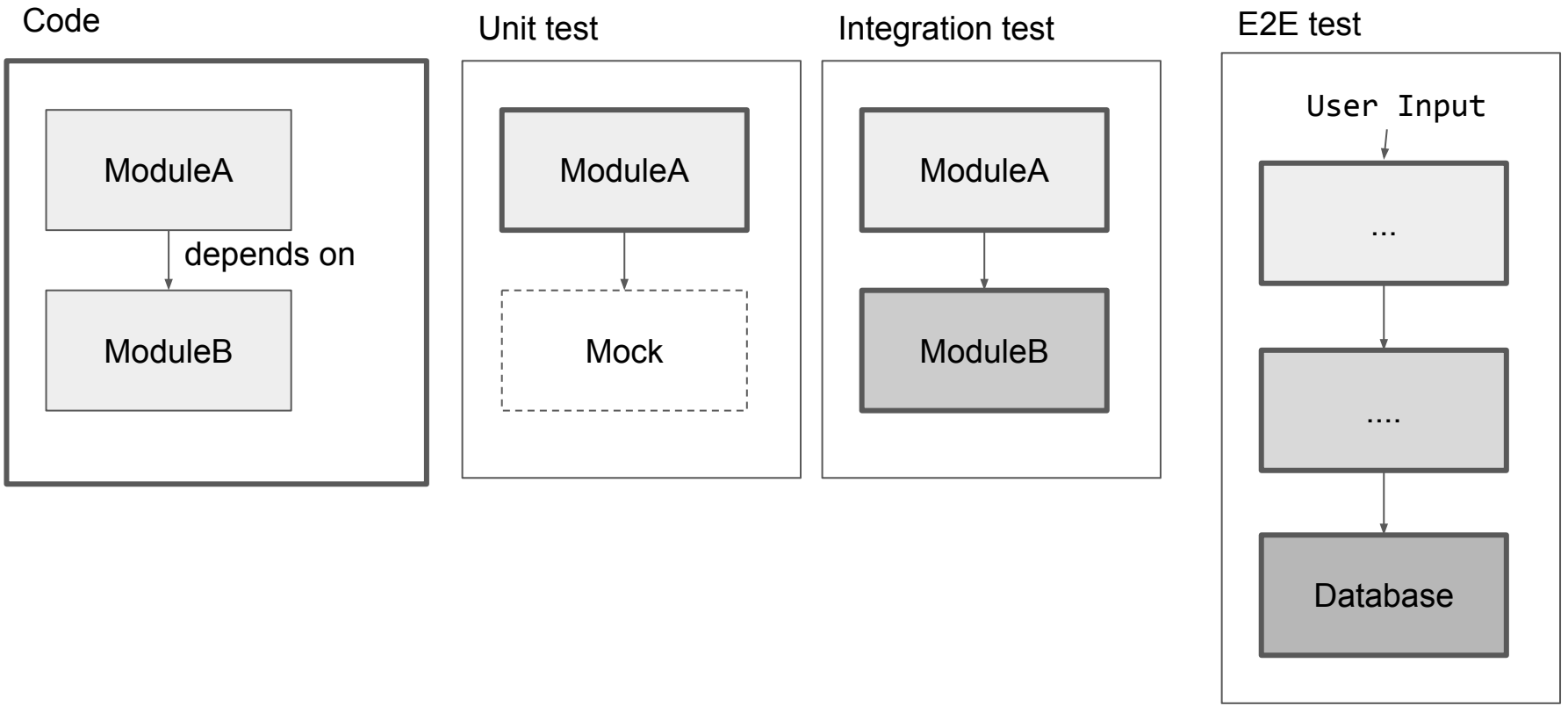
Class based

```
class TestAdd(unittest.TestCase):
    def test_two_numbers(self):
        self.assertEqual(add(2, 2), 4)
        self.assertEqual(add(2, -2), 0)

    def test_two_strings(self):
        self.assertEqual(add("pine", "apple"),
"pineapple")
```

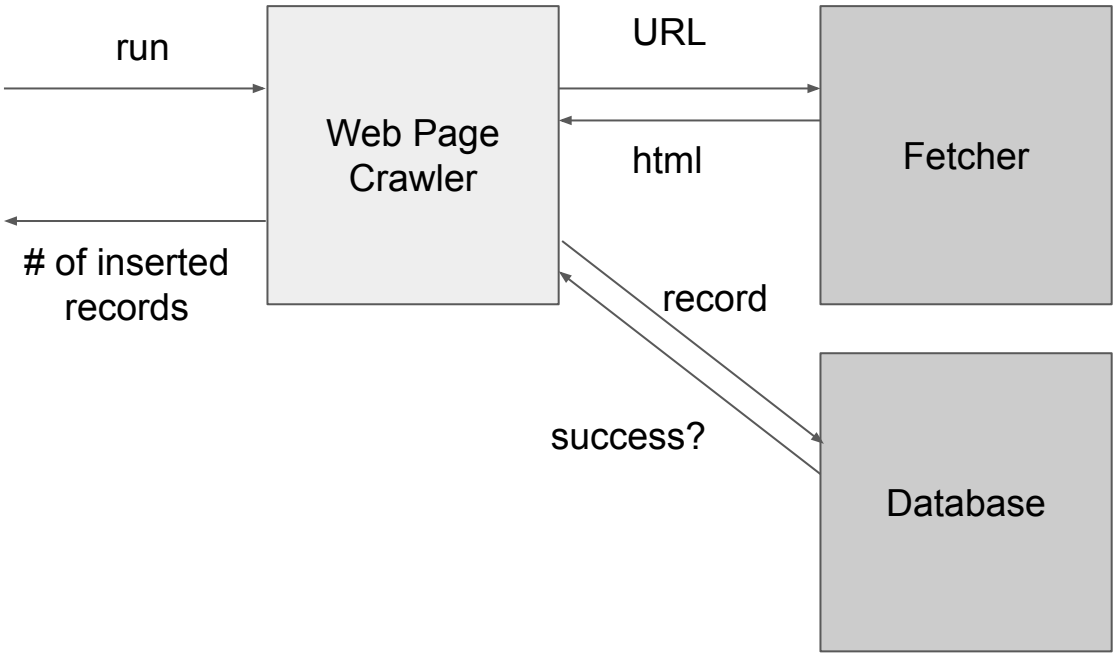
Several kinds of testing

- **Unit test**: a test on unit(e.g. class, function) without integrating with external dependencies
  - Fastest form of testing. Easy to scope down errors.
  - Dependencies are replaced by *mocks*.
- **Integration test**: a test on integration of several units
  - Combine multiple modules and examine them at once.
  - Focus on correctness of connection between modules.
- **E2E test**: an integration test where top end is a user input and bottom end is the deepest piece (e.g. database, network, file IO) of software.
  - Test that runs in the most similar environment to real one.
  - Slowest.
  - Easy to write but hard to spot the root cause of error when test cases fail.
  - Hard to isolate tests.

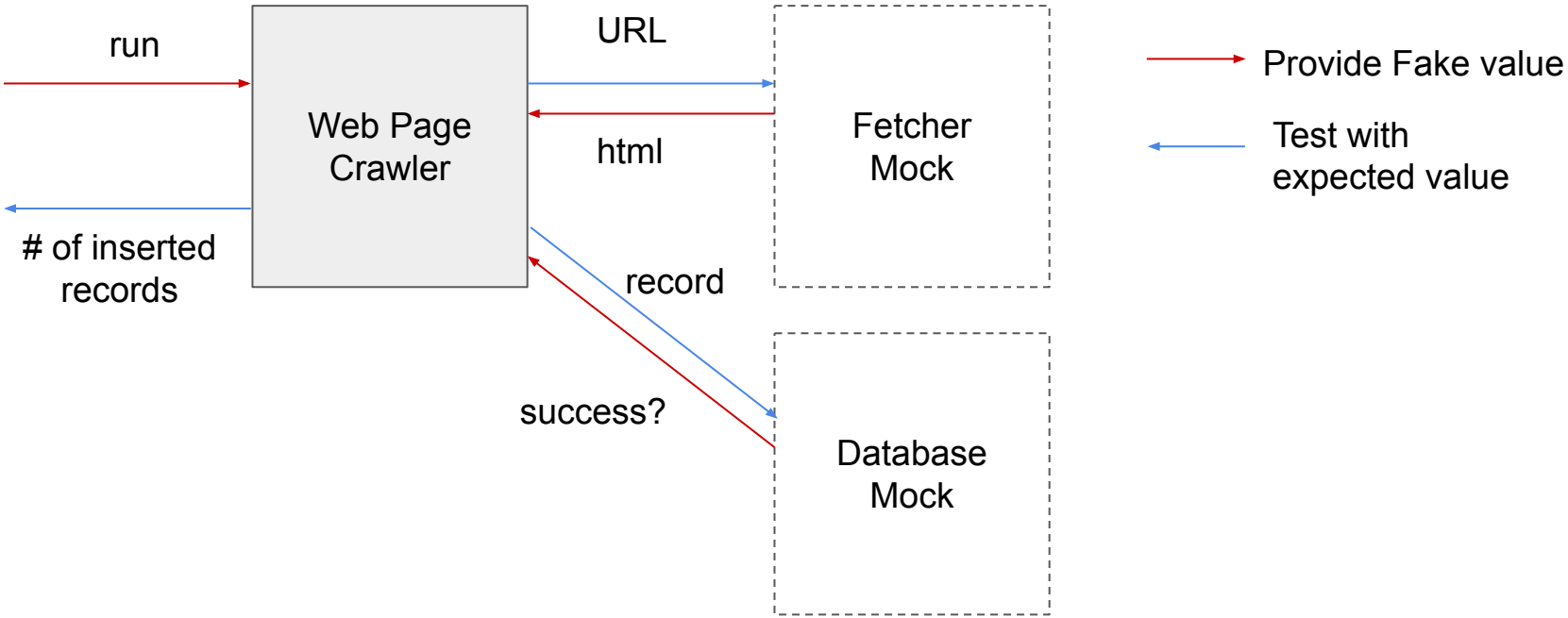
Several kinds of testing (Cont.)



Typical pattern of Mocking



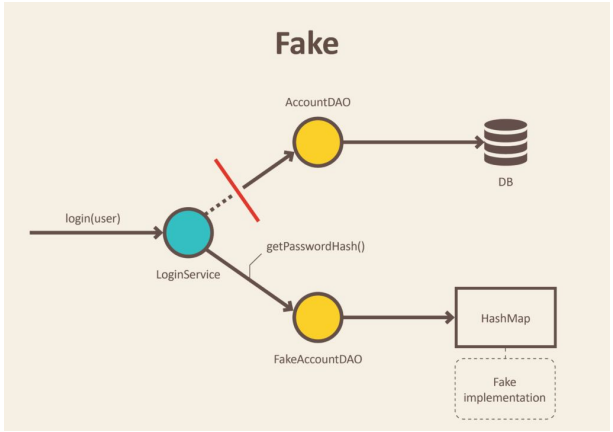
Typical pattern of Mocking

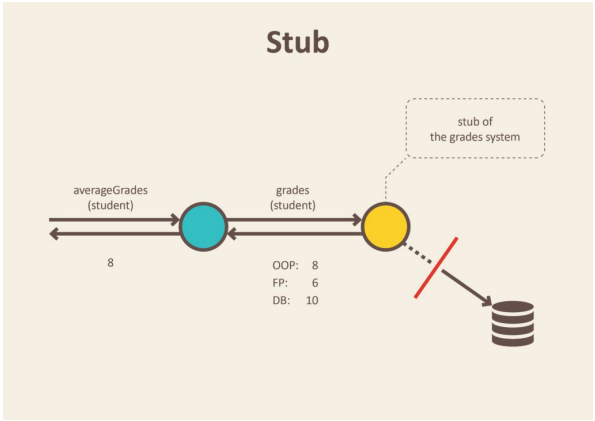


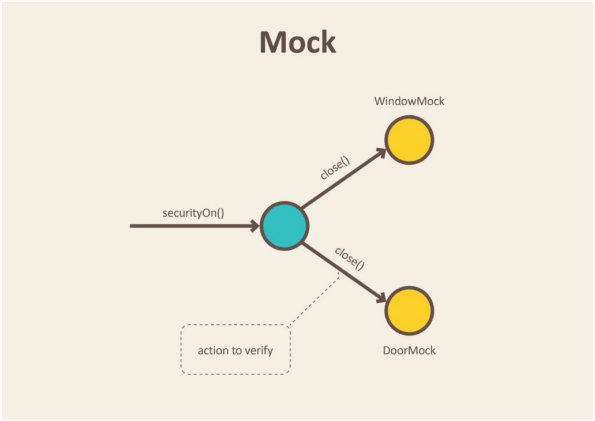
Test Doubles: Stub, Mock and Fake

- Test Double is an object that replace an original dependency.
- Three kinds
  - Stub
  - Mock / Spy
  - Fake
- *Mock* is the commonly (mis)used term to refer to the above concepts in a casual manner
  - In case of python, you can get `Mock` class from `unittest.mock` package.
- Why do we have to use {stub, mock, spy, fake}?
  - To avoid libraries causing side effects
    - e.g.. HTTP request to backend (like axios)
  - To isolate irrelevant non-library components from being tested
    - e.g., You shouldn't test Todo component when testing TodoList component
  - To make a test case much faster.

Test Doubles: Stub, Mock and Fake (Cont.)







Why integration test?

- The faucet works
- The sink works
- The combination of them might not work

Why integration test?

- The lock works
- The door works
- The combination of them might not work

Tips

1. Don’t neglect failing unit tests. *All green or nothing*. If you *have to* leave them failing, you can temporarily mark the test cases to skip.
2. Even though high coverage rate does not mean high quality of test suites, it is still meaningful for project management, because it acts as a nudge to encourage writing tests by giving feeling of achievement.
3. Don’t freak out when a test case fails. Rather, you should feel happy about the failing test because it could’ve sneaked into codebase and could’ve become a unnoticeable bug.
4. Counterintuitively, test execution speed is quite important.

Frontend Testing

Our final goal

- Now in your terminal, run

```
$ yarn && yarn test --coverage --watchAll=false
```

- And you should see something like...

File	% Stmts	% Branch	% Funcs	% Lines	Uncovered Line #s
All files	100	100	100	100	
src	100	100	100	100	
App.tsx	100	100	100	100	
src/components/Todo	100	100	100	100	
Todo.tsx	100	100	100	100	
src/components/TodoDetail	100	100	100	100	
TodoDetail.tsx	100	100	100	100	
src/containers/TodoList	100	100	100	100	
TodoList.tsx	100	100	100	100	
src/containers/TodoList/NewTodo	100	100	100	100	
NewTodo.tsx	100	100	100	100	
src/store	100	100	100	100	
index.ts	100	100	100	100	
src/store/slices	100	100	100	100	
todo.ts	100	100	100	100	

Frontend Testing Framework - Jest

- Jest
  - Popular Javascript Unit testing Framework
  - Describe-It style test suites
  - Easy-to-use Mock & Spy API

```
describe("add(x, y)", function() {
  it("should add up two numbers", function () {
    expect(add(2, 2)).toEqual(2);
    expect(add(2, -2)).toEqual(0);
  });
  it("should concatenate two strings", function () {
    expect(add("pine", "apple")).toEqual("pineapple");
  });
});
```

Jest

- Before-After hook

```
describe("the name of test suite", () => {
  beforeAll(function() {}); // called before suite
  afterAll(function() {});    // called after suite
  beforeEach(function() {});    // called before each spec
  afterEach(function() {});    // called after each spec
  it("should/contains ... spec1 ", () => {
      // expect(value).matchers
      expect(true).toBe(true);
      expect(true).not.toBe(false);
  });
  it("should/contains ... spec2 ", () => {
    expect(true).toEqual(true);
  })
  // ...
});
```

This is called ONCE before all tests

This is called ONCE after all tests

Jest

- Before-After hook

```
describe("the name of test suite", () => {
  beforeAll(function() {}); // called before suite
  afterAll(function() {});    // called after suite
  beforeEach(function() {});    // called before each spec
  afterEach(function() {});    // called after each spec
  it("should/contains ... spec1 ", () => {
      // expect(value).matchers
      expect(true).toBe(true);
      expect(true).not.toBe(false);
  });
  it("should/contains ... spec2 ", () => {
    expect(true).toEqual(true);
  })
  // ...
});
```

This is called EVERY TIME before each tests

This is called EVERY TIME after each tests

Jest

- Simple Testcase

```
describe("the name of test suite", () => {
  beforeAll(function() {}); // called before suite
  afterAll(function() {});    // called after suite
  beforeEach(function() {});    // called before each spec
  afterEach(function() {});    // called after each spec
  it("should/contains ... spec1 ", () => {
      // expect(value).matchers
      expect(true).toBe(true);
      expect(true).not.toBe(false);
  });
  it("should/contains ... spec2 ", () => {
    expect(true).toEqual(true);
  })
  // ...
});
```

The real part of test. It should perform scenarios and check its results.

Jest

- Test suite is nestable

```
describe("the name of test suite", () => {
  beforeEach(function() {});    // called before each of each spec

  describe("the name of inner", () => {
    beforeEach(function() {})    // called before each spec below
    it("should/contains ... spec1 ", () => {
        // except(value).matchers
        expect(true).toBe(true);
        expect(true).not.toBe(false);
    });
});
```

It allows nested syntax, meaning that beforeEach of the parent gets called before each of the inner tests as well.

Jest

- Special “it”s

```
describe("the name of test suite", () => {
  xit("should/contains ... spec1 ", () =>
{
      // expect(value).matchers
      expect(true).toBe(true);
      expect(true).not.toBe(false);
  });
});
```

This spec does not run

```
describe("the name of test suite", () => {
  fit("should/contains ... spec1 ", () =>
{
      // expect(value).matchers
      expect(true).toBe(true);
      expect(true).not.toBe(false);
  });
});
```

Only this spec runs

Jest Matcher APIs for general objects

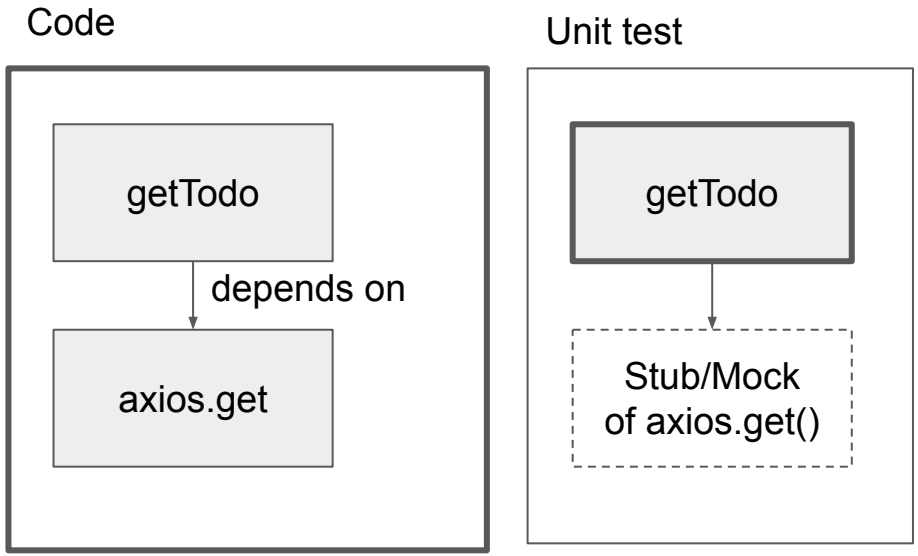
- Equality
  - `.toBe(value)`
  - `.toEqual(value)`
  - `.toBeCloseTo(number, numDigits?)`
- Inequality
  - `.toBeLessThan(number)`
  - `.toBeGreaterThan(number)`
  - `.toBeGreaterThanOrEqual(number)`
  -
- True/False
  - `.toBeDefined()`
  - `.toBeTruthy()`
  - `.toBeFalsy()`
- Object/List
  - .toHaveProperty(keyPath, value?)
  - .toHaveLength(number)

Why use a matcher like `expect(x).toBeGreaterThan(0)` if you can use `expect(x > 0).toBe(true)`?

-> Jest shows more comprehensive logs about failed assertion in the former case.

… See Jest official document (https://jestjs.io/docs/en/expect)

Mocking/Stubbing External Dependencies



jest.fn

```
axios.get = jest.fn(
  // Your stub/mock function...
);
```

jest.spyOn

```
const spy = jest.spyOn(axios, 'get')
  .mockImplementation(
    // Your stub/mock function...
  );
```

Jest Mock/Spy Matcher APIs

- `.toHaveBeenCalled()`
- `.toHaveBeenCalledTimes(number)`
- `.toHaveBeenCalledWith(arg1, arg2, ...)`
- `.toHaveBeenLastCalledWith(arg1, arg2, ...)`
- `.toHaveBeenNthCalledWith(nthCall, arg1, arg2, ....)`
- `.toHaveReturned()`
- `.toHaveReturnedTimes(number)`
- `.toHaveReturnedWith(value)`
- `.toHaveLastReturnedWith(value)`
- `.toHaveNthReturnedWith(nthCall, value)`

You can test a mock function after you have called its root method. You can inspect whether it has been called, what arguments it has been called with, or how many times it has been called.

Remember names of Jest mock matchers are *perfect infinitive* (to have p.p.)

… See Jest official document (https://jestjs.io/docs/en/expect)

React Testing Framework - React Testing Library

- React Testing Library
  - testing library for React
  - On top of `react-dom/test-utils` which is built in testing for react
  - Examine, render, access and manipulate React elements
  - `yarn test` to start

// Part of package.json

```
"scripts": {
  "start": "react-scripts start",
  "build": "react-scripts build",
  "test": "react-scripts test",
  "eject": "react-scripts eject"
},
```

```
No tests found related to files changed since last commit.
Press `a` to run all tests, or run Jest with `--watchAll`.

Watch Usage
 › Press a to run all tests.
 › Press f to run only failed tests.
 › Press q to quit watch mode.
 › Press p to filter by a filename regex pattern.
 › Press t to filter by a test name regex pattern.
 › Press Enter to trigger a test run.


✨  Done in 5.07s.
```

Play with RTL

- Make our first test code and run `yarn test --watchAll` in shell

```
// Modify src/App.test.tsx
import { render, screen } from "@testing-library/react";
import { Provider } from "react-redux";

import App from "./App";
import { store } from "./store";

test("renders App.tsx", () => {
 render(
   <Provider store={store}><App /></Provider>
 );
 screen.debug();
 expect(true).toBe(false)  // This make failing test case
});
```

We need redux provider to start App.

This will show rendered html code

Testing Independent Components

- Test `Todo.tsx`.
- We expect ‘Todo’ have given title and not done state.
- If todo is not done, we expected to be button with “Done” text.

```
// src/components/Todo/Todo.test.tsx
import { render, screen } from "@testing-library/react";
import Todo from "./Todo";

describe("<Todo />", () => {
 it("should render without errors", () => {
   render(<Todo title={"TODO_TITLE"} done={false} />);
   screen.getByText("TODO_TITLE"); // Implicit assertion
   const doneButton = screen.getByText("Done"); // Implicit assertion
   expect(doneButton).toBeInTheDocument(); // Explicit assertion
 });
});
```

Testing Independent Components

```tsx
// src/components/Todo/Todo.test.tsx
import { render, screen } from "@testing-library/react";
import Todo from "./Todo";

describe("<Todo />", () => {
 it("should render without errors", () => {
   render(<Todo title={"TODO_TITLE"} done={false} />);
   screen.getByText("TODO_TITLE"); // Implicit assertion
   const doneButton = screen.getByText("Done"); // Implicit assertion
   expect(doneButton).toBeInTheDocument(); // Explicit assertion
 });
});
```

`describe(string, function)` : define **Test Suite**

[Jest] *a collection of individual Test Specs*

(The string parameter can be any text)

Testing Independent Components

```tsx
// src/components/Todo/Todo.test.tsx
import { render, screen } from "@testing-library/react";
import Todo from "./Todo";

describe("<Todo />", () => {
 it("should render without errors", () => {
   render(<Todo title={"TODO_TITLE"} done={false} />);
   screen.getByText("TODO_TITLE"); // Implicit assertion
   const doneButton = screen.getByText("Done"); // Implicit assertion
   expect(doneButton).toBeInTheDocument(); // Explicit assertion
 });
});
```

`it(string, function)`: define **Test Spec**
[Jest]*contain one or more Test Expectations*
(The string parameter can be any text)

Testing Independent Components

```tsx
// src/components/Todo/Todo.test.tsx
import { render, screen } from "@testing-library/react";
import Todo from "./Todo";

describe("<Todo />", () => {
 it("should render without errors", () => {
  render(<Todo title={"TODO_TITLE"} done={false} />);
   screen.getByText("TODO_TITLE"); // Implicit assertion
   const doneButton = screen.getByText("Done"); // Implicit assertion
   expect(doneButton).toBeInTheDocument(); // Explicit assertion
 });
});
```

`render(JSX, [options])`: render JSX component. [RTL]

Testing Independent Components

```tsx
// src/components/Todo/Todo.test.tsx
import { render, screen } from "@testing-library/react";
import Todo from "./Todo";

describe("<Todo />", () => {
 it("should render without errors", () => {
   render(<Todo title={"TODO_TITLE"} done={false} />);
   screen.getByText("TODO_TITLE"); // Implicit assertion
   const doneButton = screen.getByText("Done"); // Implicit assertion
   expect(doneButton).toBeInTheDocument(); // Explicit assertion
 });
});
```

`getByText(text)`: Find element with text.**[RTL]**
Throw Error when cannot find any element.

Testing Independent Components

```tsx
// src/components/Todo/Todo.test.tsx
import { render, screen } from "@testing-l
import Todo from "./Todo";

describe("<Todo />", () => {
 it("should render without errors", () => {
   render(<Todo title={"TODO_TITLE"} done={false} />);
   screen.getByText("TODO_TITLE"); // Implicit assertion
   const doneButton = screen.getByText("Done"); // Implicit assertion
   expect(doneButton).toBeInTheDocument(); // Explicit assertion
 });
});
```

`expect(actual)`: **Expectation**
[Jest] *used with Matcher, describes an expected piece of behavior*

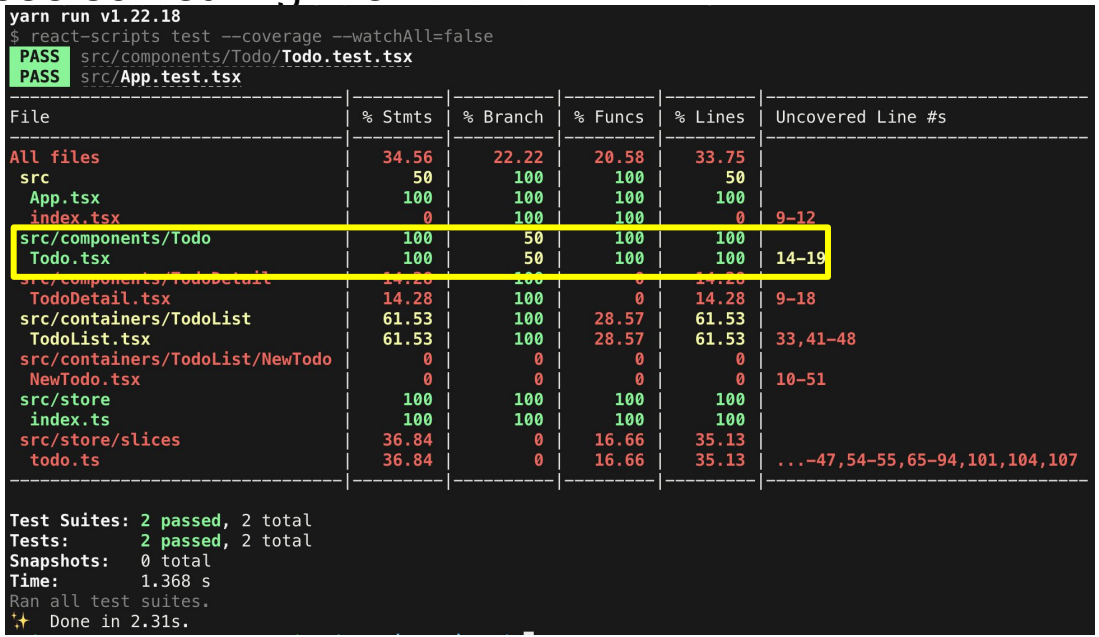
`matcher(expected)`: **Matcher**
[Jest] *do comparison between **actual** and **expected** value*

Testing Independent Components

- Now in your terminal, run

```
$ yarn test --coverage --watchAll=false
```

- And you'll see something like...



```
yarn run v1.22.18
$ react-scripts test --coverage --watchAll=false
 PASS  src/components/Todo/Todo.test.tsx
 PASS  src/App.test.tsx
-------------------------------|---------|----------|---------|---------|-------------------------------
File                           | % Stmts | % Branch | % Funcs | % Lines | Uncovered Line #s
-------------------------------|---------|----------|---------|---------|-------------------------------
All files                      |   34.56 |    22.22 |   20.58 |   33.75 |
 src                           |      50 |      100 |     100 |      50 |
  App.tsx                      |     100 |      100 |     100 |     100 |
  index.tsx                    |       0 |      100 |     100 |       0 | 9-12
 src/components/Todo           |     100 |       50 |     100 |     100 |
  Todo.tsx                     |     100 |       50 |     100 |     100 | 14-19
 src/components/TodoDetail     |   14.28 |      100 |       0 |   14.28 |
  TodoDetail.tsx               |   14.28 |      100 |       0 |   14.28 | 9-18
 src/containers/TodoList       |   61.53 |      100 |   28.57 |   61.53 |
  TodoList.tsx                 |   61.53 |      100 |   28.57 |   61.53 | 33,41-48
 src/containers/TodoList/NewTodo |       0 |        0 |       0 |       0 |
  NewTodo.tsx                  |       0 |        0 |       0 |       0 | 10-51
 src/store                     |     100 |      100 |     100 |     100 |
  index.ts                     |     100 |      100 |     100 |     100 |
 src/store/slices              |   36.84 |        0 |   16.66 |   35.13 |
  todo.ts                      |   36.84 |        0 |   16.66 |   35.13 | ...-47,54-55,65-94,101,104,107
-------------------------------|---------|----------|---------|---------|-------------------------------

Test Suites: 2 passed, 2 total
Tests:       2 passed, 2 total
Snapshots:   0 total
Time:        1.368 s
Ran all test suites.
✨  Done in 2.31s.
```

Why?

In component/Todo/Todo.js

```
11  const Todo = (props: IProps) => {
12    return (
13      <div className="Todo">
14        <div className={`text ${props.done && "done"}`} onClick={props.clickDetail}>
15          {props.title}
16        </div>
17        {props.done && <div className="done-mark">&#x2713;</div>}
18        <button className={props.done ? "Undone" : "done"} onClick={props.clickDone}>
19          {props.done ? "Undone" : "Done"}
20        </button>
21        <button onClick={props.clickDelete}>Delete</button>
22      </div>
23    );
24  };
25  export default Todo;
26
```

Here we have 2 branches that we didn't test!

Testing Independent Components

- Add some more tests

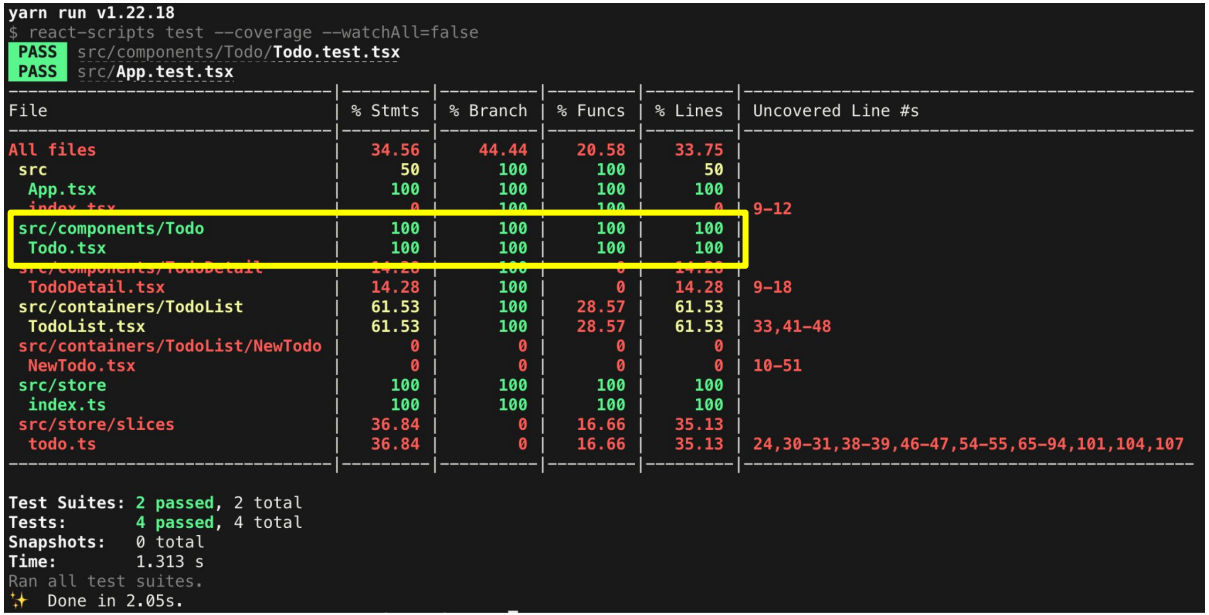
```
// src/components/Todo/Todo.test.tsx
...
describe("<Todo />", () => {
 ...
 it("should render done mark when done is true", () => {
   render(<Todo title={"TODO_TITLE"} done={true} />);
   const title = screen.getByText("TODO_TITLE");
   expect(title.classList.contains("done")).toBe(true);
   screen.getByText("Undone");
 });
 it("should render undone mark when done is false", () => {
   render(<Todo title={"TODO_TITLE"} done={false} />);
   const title = screen.getByText("TODO_TITLE");
   expect(title.classList.contains("done")).toBe(false);
   screen.getByText("Done");
 });
});
```

Testing Independent Components

- Again in your terminal, run

```
$ yarn test --coverage --watchAll=false
```

- And you'll see something like...



```
yarn run v1.22.18
$ react-scripts test --coverage --watchAll=false
 PASS  src/components/Todo/Todo.test.tsx
 PASS  src/App.test.tsx
-----------------------------------|---------|----------|---------|---------|-------------------
File                               | % Stmts | % Branch | % Funcs | % Lines | Uncovered Line #s
-----------------------------------|---------|----------|---------|---------|-------------------
All files                          |   34.56 |    44.44 |   20.58 |   33.75 |
 src                               |      50 |      100 |      50 |         |
  App.tsx                          |     100 |      100 |     100 |     100 |
  index.tsx                        |       0 |      100 |     100 |       0 | 9-12
 src/components/Todo               |     100 |      100 |     100 |     100 |
  Todo.tsx                         |     100 |      100 |     100 |     100 |
 src/components/TodoDetail         |   14.28 |      100 |       0 |   14.28 |
  TodoDetail.tsx                   |   14.28 |      100 |       0 |   14.28 | 9-18
 src/containers/TodoList           |   61.53 |      100 |   28.57 |   61.53 |
  TodoList.tsx                     |   61.53 |      100 |   28.57 |   61.53 | 33,41-48
 src/containers/TodoList/NewTodo   |       0 |        0 |       0 |       0 |
  NewTodo.tsx                      |       0 |        0 |       0 |       0 | 10-51
 src/store                         |     100 |      100 |     100 |     100 |
  index.ts                         |     100 |      100 |     100 |     100 |
 src/store/slices                  |   36.84 |        0 |   16.66 |   35.13 |
  todo.ts                          |   36.84 |        0 |   16.66 |   35.13 | 24,30-31,38-39,46-47,54-55,65-94,101,104,107
-----------------------------------|---------|----------|---------|---------|-------------------

Test Suites: 2 passed, 2 total
Tests:       4 passed, 4 total
Snapshots:   0 total
Time:        1.313 s
Ran all test suites.
✨  Done in 2.05s.
```

To Exclude Unnecessary Files

- You can exclude files from test coverage measurement by:
- *index.tsx* files will be excluded from HW3 test coverage measurement, too.

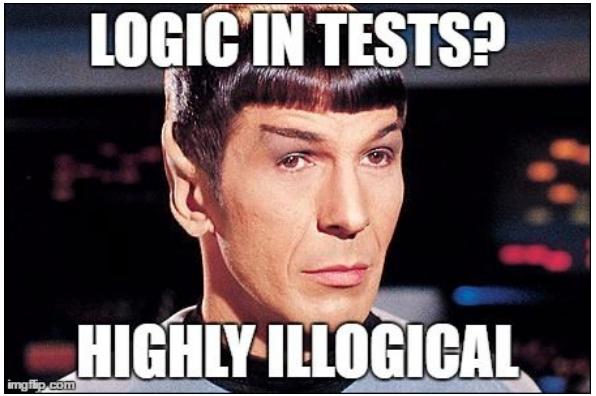
```
// add this to package.json

//...
 "jest": {
   "collectCoverageFrom": [
     "src/**/*.{js,jsx,ts,tsx}",
     "!src/index.tsx",
     "!src/test-utils/*"
   ]
//...
```

Positive

Negative

Tips

- Remember to be FIRST:
  - **F**AST
  - **I**SOLATED/INDEPENDENT
  - **R**EPEATABLE
  - **S**ELF-VALIDATING
  - **T**HOROUGH/TIMELY
- In test code, stay away from complicated logic!
  - You DO NOT want your test code to have any bug. (What tests test codes with bugs?)
- Keep your app code testable
  - Single Responsibility Principle: each function/class/module should have responsibility over a single, small, modular part of the whole functionality



Tips

- toBe(...) vs. toEqual(...)
  - toBe(...) checks equality as if they were compared using `===`. (Strictly speaking, `Object.is` is used instead of `===`, which has a subtle difference.)
  - toEqual(...) checks whether the two objects are **equivalent.**

```
const person1 = {name: 'Kyle', age: 20};
const person2 = {name: 'Kyle', age: 20};

expect(person1).toEqual(person2); // This test succeeds.
expect(person1).toBe(person2);    // This one doesn't.
```

Testing Library queries

Query By

- **Role:** getByRole(“img”):
  - HTML tag to Role map: site
- **LabelText:** getByLabelText: <label for="search" />
- **PlaceholderText:** getByPlaceholderText: <input placeholder="Search" />
- **AltText:** getByAltText:
- **DisplayValue:** getByDisplayValue: <input value="JavaScript" />

… See RTL official document (https://testing-library.com/docs/queries/about)

Testing Library queries

Type of Query	0 Matches	1 Match	>1 Matches	Retry (Async/Await)
Single Element				
`getBy...`	Throw error	Return element	Throw error	No
`queryBy...`	Return `null`	Return element	Throw error	No
`findBy...`	Throw error	Return element	Throw error	Yes
Multiple Elements				
`getAllBy...`	Throw error	Return array	Return array	No
`queryAllBy...`	Return `[]`	Return array	Return array	No
`findAllBy...`	Throw error	Return array	Return array	Yes

… See RTL official document (https://testing-library.com/docs/queries/about)

Fire User Behavior

```
// <button>Submit</button>
fireEvent(
  getByText(container, "Submit"),
  new MouseEvent("click", {
    bubbles: true,
    cancelable: true,
  })
  fireEvent.click(screen.getByText(/click me/i))
);
```

Testing TodoSlice

- TodoSlice is part of redux store.
- Think what we expect for todoSlice.
  - It should have initState.
  - It should handle CURD todo data from server.
    - `fetchTodos`, `fetchTodo`, `postTodo`, `deleteTodo`, `toggleDone`
    - All should work async
    - All should save data to redux store

Testing TodoSlice

```
// src/store/slices/todo.test.ts
import {
  AnyAction,
  configureStore,
  EnhancedStore
} from "@reduxjs/toolkit";
import axios from "axios";
import { ThunkMiddleware } from "redux-thunk";
import reducer, { TodoState } from "./todo";
import {
  fetchTodos,
  fetchTodo,
  postTodo,
  deleteTodo,
  toggleDone
} from "./todo";
```

```
describe("todo reducer", () => {
 let store: EnhancedStore<
   { todo: TodoState },
   AnyAction,
   [ThunkMiddleware<{ todo: TodoState }, AnyAction,
undefined>]
 >;
 const fakeTodo = {
  id: 1, title: "test", content: "test", done: false
 };

 beforeAll(() => {
   store = configureStore(
     { reducer: { todo: reducer } }
   );
 }); // end beforeAll
}); // end describe
```

Testing TodoSlice

- Check initState
- Mock axios.get

```
// src/store/slices/todo.test.ts
// Inside describe, after beforeAll()
 it("should handle initial state", () => {
   expect(reducer(undefined, { type: "unknown" })).toEqual({
     todos: [],
     selectedTodo: null,
   });
 });
 it("should handle fetchTodos", async () => {
   axios.get = jest.fn().mockResolvedValue({ data: [fakeTodo] });
   await store.dispatch(fetchTodos());
   expect(store.getState().todo.todos).toEqual([fakeTodo]);
 });
```

Mock axios.get

Testing TodoSlice

- Do same thing for other actions

```
// src/store/slices/todo.test.ts
 it("should handle fetchTodo", async () => {
   axios.get = jest
     .fn()
     .mockResolvedValue({ data: fakeTodo });
   await store.dispatch(fetchTodo(1));
   expect(store.getState().todo.selectedTodo).toEqual(
     fakeTodo
   );
 });
 it("should handle deleteTodo", async () => {
   axios.delete = jest
     .fn()
     .mockResolvedValue({ data: null });
   await store.dispatch(deleteTodo(1));
   expect(store.getState().todo.todos).toEqual([]);
 });
```

```
 it("should handle postTodo", async () => {
   jest.spyOn(axios, "post").mockResolvedValue({
       data: fakeTodo,
   });
   await store.dispatch(
     postTodo({ title: "test", content: "test" })
   );
   expect(store.getState().todo.todos).toEqual([fakeTodo]);
 });
 it("should handle toggleDone", async () => {
   jest.spyOn(axios, "put").mockResolvedValue({
       data: fakeTodo,
   });
   await store.dispatch(toggleDone(fakeTodo.id));
   expect(
     store
       .getState()
       .todo.todos.find((v) => v.id === fakeTodo.id)?.done
   ).toEqual(true);
 });
```

Tips

- jest.fn(...) vs. jest.spyOn(...)
  - They basically serve **the same purpose**; mocking.
  - jest.spyOn() can be used only to an existing function in a module
- Implementation using spyOn() is in Checkpoint 5: `$ git checkout checkpoint5`

jest.fn

```
axios.get = jest.fn(
  // Your mock function...
);
```

jest.spyOn

```
const spy = jest.spyOn(axios, 'get')
  .mockImplementation(
    // Your mock function...
  );
```

Tips

- It is often a good idea to clear all mocks in afterEach().
- Otherwise, your spies will accumulate call counts.

```
// This test passes.
it('test 1', () => {
  //…
  axios.get = jest.fn();
  //…
  expect(axios.get).toHaveBeenCalledTimes(1);
};

// But this one fails; 2 !== 1
it('test 2', () => {
  //…
  axios.get = jest.fn();
  //…
  expect(axios.get).toHaveBeenCalledTimes(1);
};
```

```
afterEach(() => { jest.clearAllMocks() });
it('test 1', () => {
  //…
  axios.get = jest.fn();
  //…
  expect(axios.get).toHaveBeenCalledTimes(1);
};

// Now this one passes too.
it('test 2', () => {
  //…
  axios.get = jest.fn();
  //…
  expect(axios.get).toHaveBeenCalledTimes(1);
};
```

Not Enough

src/store/slices	89.47	33.33	77.77	91.42	
todo.ts	89.47	33.33	77.77	91.42	65-66,100

```
62  reducers: {
63    getAll: (state, action: PayloadAction<{ todos: TodoType[] }>) => {},
64    getTodo: (state, action: PayloadAction<{ targetId: number }>) => {
65      const target = state.todos.find((td) => td.id === action.payload.targetId);
66      state.selectedTodo = target ?? null;
67    },
68    toggleDone: (state, action: PayloadAction<{ targetId: number }>) => {
69      const todo = state.todos.find((value) => value.id === action.payload.targetId);
70      if (todo) {
71        todo.done = !todo.done;
72      }
73    },
```

Remove not using codes.

Not Enough

```
// src/store/slices/todo.test.ts
 it("should handle error on postTodo", async () => {
   const mockConsoleError = jest.fn();
   console.error = mockConsoleError;
   jest.spyOn(axios, "post").mockRejectedValue({
     response: { data: { title: ["error"] } },
   });
   await store.dispatch(
     postTodo({ title: "test", content: "test" })
   );
   expect(mockConsoleError).toBeCalled();
 });
 it("should handle null on fetchTodo", async () => {
   axios.get = jest.fn().mockResolvedValue({ data: null
   await store.dispatch(fetchTodo(1));
   expect(store.getState().todo.selectedTodo).toEqual(
     null
   );
 });
```

```
 it("should handle not existing todo toggle", async () => {
   const beforeState = store.getState().todo
   jest.spyOn(axios, "put").mockResolvedValue({
     data: { ...fakeTodo, id: 10 },
   });
   await store.dispatch(toggleDone(2));
   expect(store.getState().todo).toEqual(beforeState)
 });
```

Enough

File	% Stmts	% Branch	% Funcs	% Lines	Uncovered Line #s
All files	64.47	75	61.29	63.51	
src	100	100	100	100	
App.tsx	100	100	100	100	
src/components/Todo	100	100	100	100	
Todo.tsx	100	100	100	100	
src/components/TodoDetail	14.28	100	0	14.28	
TodoDetail.tsx	14.28	100	0	14.28	9-18
src/containers/TodoList	61.53	100	28.57	61.53	
TodoList.tsx	61.53	100	28.57	61.53	33,41-48
src/containers/TodoList/NewTodo	0	0	0	0	
NewTodo.tsx	0	0	0	0	10-51
src/store	100	100	100	100	
index.ts	100	100	100	100	
src/store/slices	100	100	100	100	
todo.ts	100	100	100	100	

Testing redux-connected Component

- A redux store needs to be stubbed so that we can set its initial state to a fake value.
  - You need to wrap a component with `<Provider store={mockedStore}></Provider>`
- We also need to mock child components. We don’t want to render all components in subtree.
- We need to wait for all component is ready.

Mocking Child Components

- You can use `jest.mock()` to automatically mock a module.

```
// containers/TodoList/TodoList.test.tsx
import { IProps as TodoProps } from "../../components/Todo/Todo";

jest.mock("../../components/Todo/Todo", () => (props: TodoProps) => (
 <div data-testid="spyTodo">
   <div className="title" onClick={props.clickDetail}>
     {props.title}
   </div>
   <button className="doneButton" onClick={props.clickDone}>
     done
   </button>
   <button className="deleteButton" onClick={props.clickDelete}>
     delete
   </button>
 </div>
));
```

Add mock for redux store

- You should make a mock store with fake initial value

```
// src/test-utils/mocks.ts

import { configureStore, PreloadedState } from "@reduxjs/toolkit";
import { RootState } from "../store";
import todoReducer from "../store/slices/todo";

export const getMockStore = (preloadedState?: PreloadedState<RootState>) => {
 return configureStore({
   reducer: { todo: todoReducer },
   preloadedState,
 });
};
```

Mocking redux store

- We need mock store

```
// containers/TodoList/TodoList.test.tsx
import { fireEvent, render, screen } from "@testing-library/react";
import { Provider } from "react-redux";
import { MemoryRouter, Route, Routes } from "react-router";
import { TodoState } from "../../store/slices/todo";
import { getMockStore } from "../../test-utils/mock";
import TodoList from "./TodoList";
...
const stubInitialState: TodoState = {
 todos: [
   { id: 1, title: "TODO_TEST_TITLE_1", content: "TODO_TEST_CONTENT_1", done: false },
   { id: 2, title: "TODO_TEST_TITLE_2", content: "TODO_TEST_CONTENT_2", done: false },
   { id: 3, title: "TODO_TEST_TITLE_3", content: "TODO_TEST_CONTENT_3", done: false },
 ],
 selectedTodo: null,
};
const mockStore = getMockStore({ todo: stubInitialState });
```

Mocking navigate and dispatch

- We need to mock navigate and dispatch for monitoring react

```
// containers/TodoList/TodoList.test.tsx
...
const mockNavigate = jest.fn();
jest.mock("react-router", () => ({
 ...jest.requireActual("react-router"),
 useNavigate: () => mockNavigate,
}));
const mockDispatch = jest.fn();
jest.mock("react-redux", () => ({
 ...jest.requireActual("react-redux"),
 useDispatch: () => mockDispatch,
}));
```

Wrap with mock redux store and Router

- We need to wrap a component with `<Provider store={mockStore}/>` and `<Router/>`

```
// containers/TodoList/TodoList.test.tsx
...
describe("<TodoList />", () => {
 let todoList: JSX.Element;
 beforeEach(() => {
   jest.clearAllMocks();
   todoList = (
     <Provider store={mockStore}>
       <MemoryRouter>
         <Routes>
           <Route path="/" element={<TodoList title="TODOLIST_TEST_TITLE" />} />
         </Routes>
       </MemoryRouter>
     </Provider>
   );
 });
});
```

Test render

- It should render.

```
// containers/TodoList/TodoList.test.tsx
// Inside describe, after beforeEach()
...
 it("should render TodoList", () => {
   const { container } = render(todoList);
   expect(container).toBeTruthy();
 });
 it("should render todos", () => {
   render(todoList);
   const todos = screen.getAllByTestId("spyTodo");
   expect(todos).toHaveLength(3);
 });
```

Test onClick

- It should handle clicks.

```
// containers/TodoList/TodoList.test.tsx
...
 it("should handle clickDetail", () => {
   render(todoList);
   const todos = screen.getAllByTestId("spyTodo");
   const todo = todos[0];
   // eslint-disable-next-line testing-library/no-node-access
   const title = todo.querySelector(".title");
   fireEvent.click(title!);
   expect(mockNavigate).toHaveBeenCalledTimes(1);
 });
```

```
it("should handle clickDone", () => {
  render(todoList);
  const todos = screen.getAllByTestId("spyTodo");
  const todo = todos[0];
  // eslint-disable-next-line testing-library/no-node-access
  const doneButton = todo.querySelector(".doneButton");
  fireEvent.click(doneButton!);
  expect(mockDispatch).toHaveBeenCalled();
});
it("should handle clickDelete", () => {
  render(todoList);
  const todos = screen.getAllByTestId("spyTodo");
  const todo = todos[0];
  // eslint-disable-next-line testing-library/no-node-access
  const deleteButton = todo.querySelector(".deleteButton");
  fireEvent.click(deleteButton!);
  expect(mockDispatch).toHaveBeenCalled();
});
```

Challenge to achieve 100% coverage

- We have about 70% test coverage now.
- From now on, adding test cases until 100% coverage is repetitive work.
- You can challenge to achieve 100% test coverage. It’s optional.
- After trying, you can jump to the final branch and compare with its tests.

Wrap up

- *Testing matters.*
- Describe a test suite with `describe(..)` and define a test case with `it(...)`
- Expect
  - `expect(1 + 1).toBe(2)`
  - `expect({...obj, name: “Alchan”}).toEqual({age: 20, name: “Alchan”})`
  - `expect(mockFunction).toHaveBeenCalled()`
- You can replace a dependency of an object.
  - `jest.fn((arg) => mockedValue)`
  - `jest.spyOn(obj, “property”).mockImplementation((arg) => mockedValue)`
  - `jest.mock(“modulePath”, () => mockedModule)`
- `Enzyme`
  - `shallow(<Component />) vs. mount(<Component />)`

Today’s Task

- Some test case broke when a calendar feature added. Make it **green** again.
- Achieve **100%** coverage for NewTodo.js and todo.js
- There could be intended bugs in source code somewhere.
  You can freely fix anything in source code.
- When you’re stuck, try `screen.debug()` to print the debug information
- Please make a PR along with screenshot to TA.
  - into `<repo>:task` from `{yours}:task`
  - until tomorrow 9PM for your attendance check