Front-end Unit Testing - Jimmy Liao

Questions

- Are you free to refactor code ?
- Have you confidence to release your untested code ?
- How to measure the quality of code ?

Front-end Unit Testing

- Testing Framework
- Basic Unit Testing
- Snapshot Testing
- Testing Library
- Redux Unit Testing
- More Testing

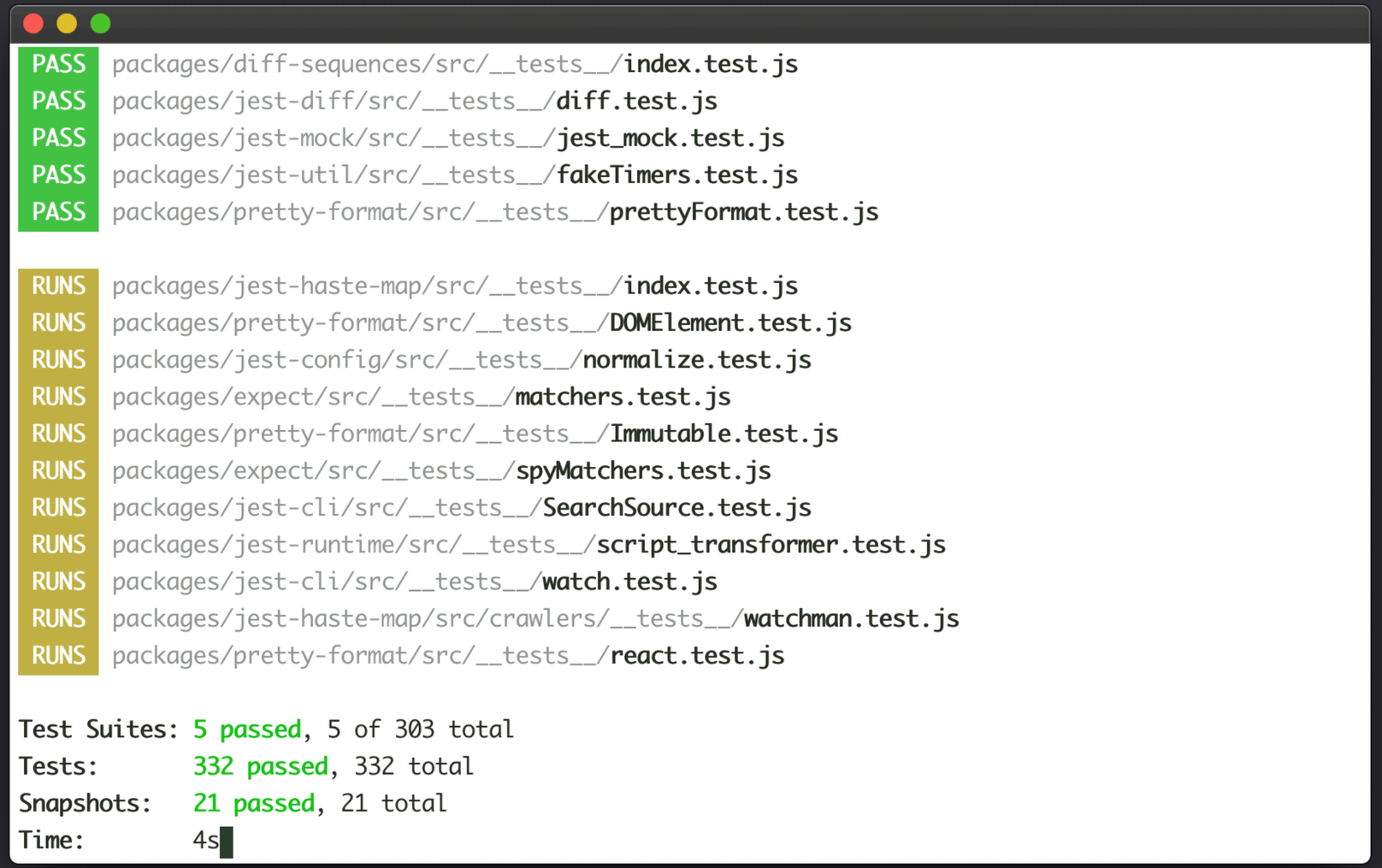
01

Testing Framework

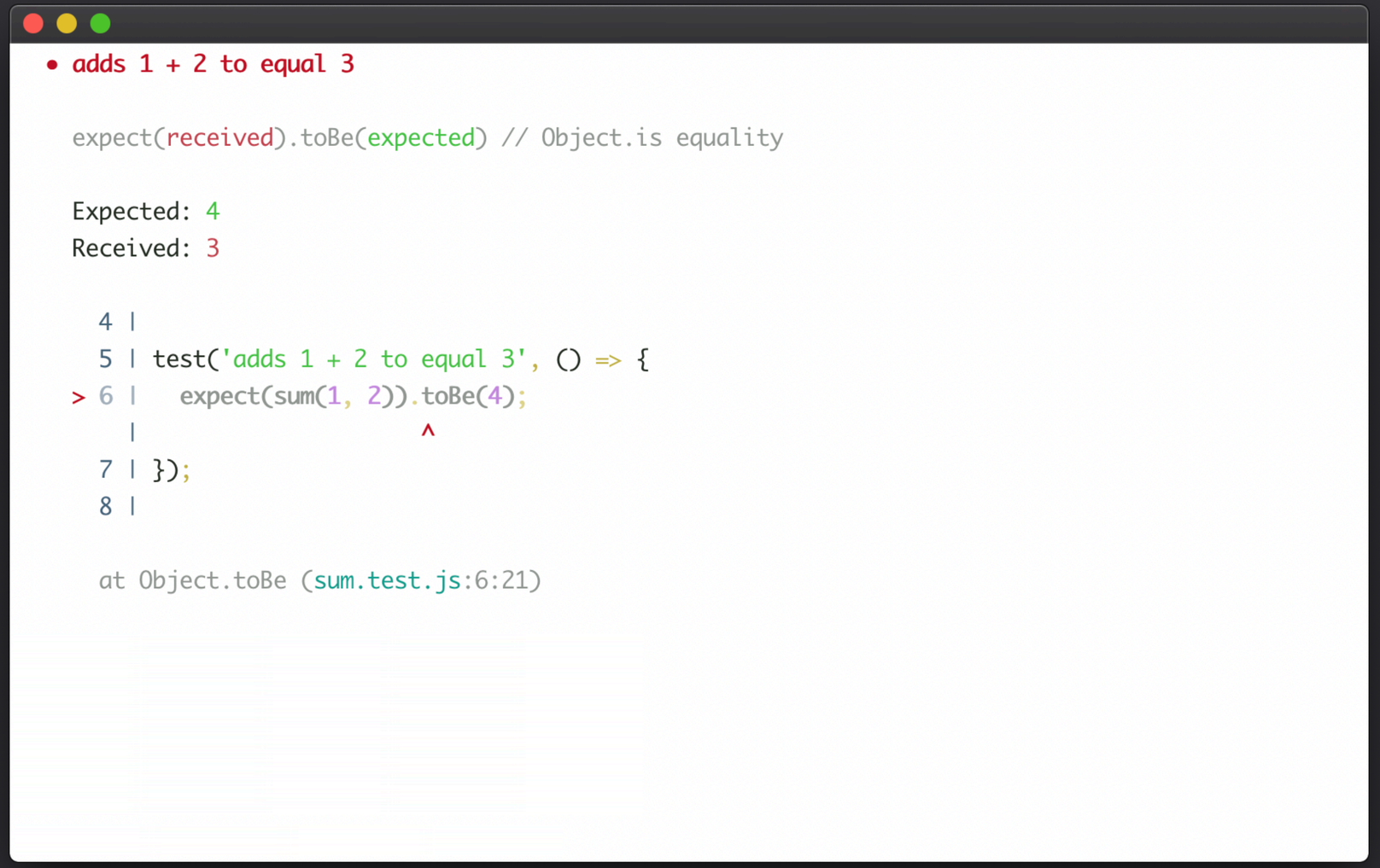
Jest

Jest is a delightful JavaScript Testing Framework with a focus on simplicity.

FAST AND SAFE



GREAT EXCEPTIONS

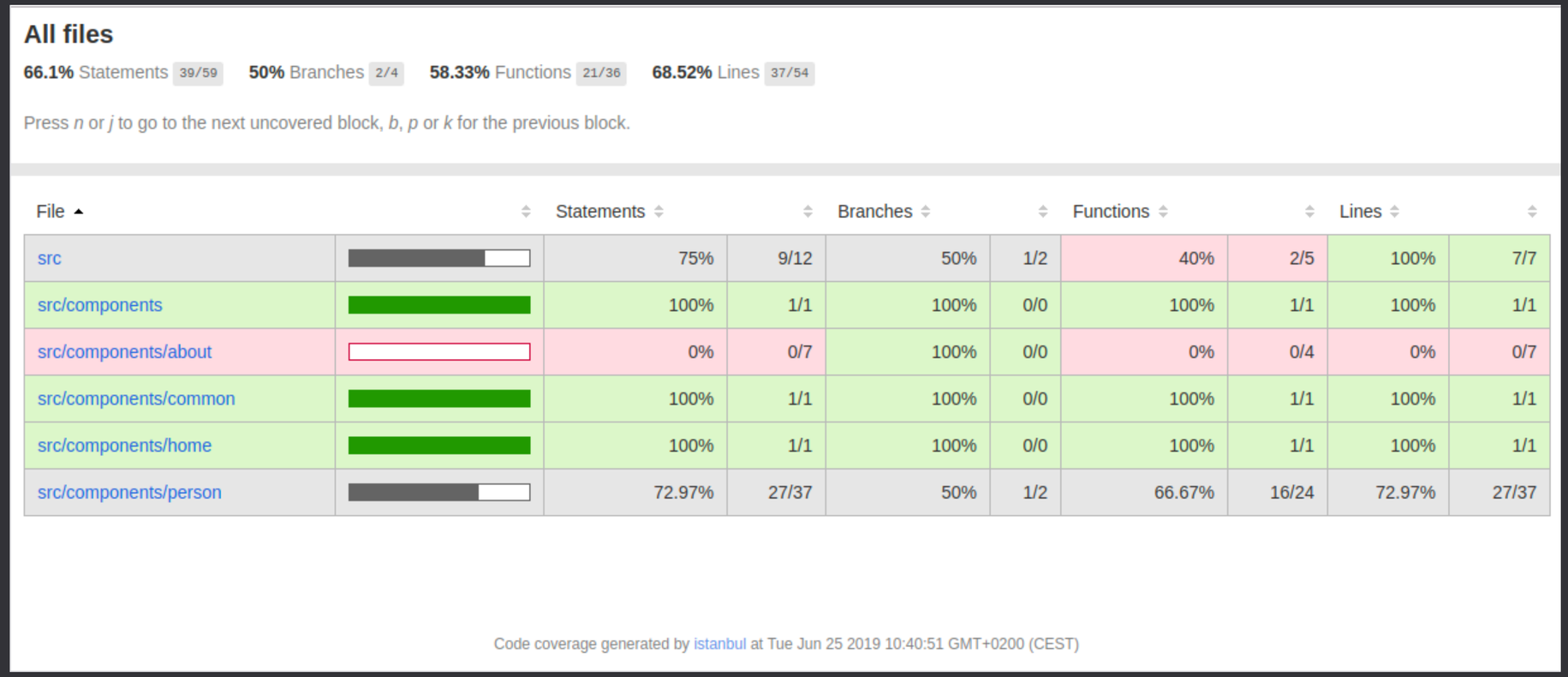


```
• adds 1 + 2 to equal 3

  expect(received).toBe(expected) // Object.is equality

  Expected: 4
  Received: 3

    4 |
    5 | test('adds 1 + 2 to equal 3', () => {
  > 6 |   expect(sum(1, 2)).toBe(4);
      |                     ^
    7 | });
    8 |

    at Object.toBe (sum.test.js:6:21)
```

EASY MOCKING

```
• should bail after `n` failures

  expect(jest.fn()).toBeCalledWith(expected)

  Expected mock function to have been called with:
    {"interrupted": false}
  as argument 1, but it was called with
    {"interrupted": true}.

  Difference:

  - Expected
  + Received

    Object {
  -   "interrupted": false,
  +   "interrupted": true,
    }

    at Object.toBeCalledWith (packages/jest-cli/src/__tests__/TestScheduler.test.js:156:20)
```

Istanbul, Test Coverage Tool



All files

66.1% Statements `39/59` **50%** Branches `2/4` **58.33%** Functions `21/36` **68.52%** Lines `37/54`

Press *n* or *j* to go to the next uncovered block, *b*, *p* or *k* for the previous block.

File ▲		Statements		Branches		Functions		Lines	
src		75%	9/12	50%	1/2	40%	2/5	100%	7/7
src/components		100%	1/1	100%	0/0	100%	1/1	100%	1/1
src/components/about		0%	0/7	100%	0/0	0%	0/4	0%	0/7
src/components/common		100%	1/1	100%	0/0	100%	1/1	100%	1/1
src/components/home		100%	1/1	100%	0/0	100%	1/1	100%	1/1
src/components/person		72.97%	27/37	50%	1/2	66.67%	16/24	72.97%	27/37

Code coverage generated by istanbul at Tue Jun 25 2019 10:40:51 GMT+0200 (CEST)

All files / src/components/person person_list_page.jsx

57.14% Statements `8/14` **100%** Branches `0/0` **54.55%** Functions `6/11` **57.14%** Lines `8/14`

Press *n* or *j* to go to the next uncovered block, *b*, *p* or *k* for the previous block.

```
1   import toastr from 'toastr';
2   import React from 'react';
3   import { Link } from 'react-router';
4   import { getPersons, deletePerson } from '../../api_calls';
5   import PersonListRow from './person_list_row';
6
7   export default class PersonList extends React.Component {
8     constructor(props) {
9  1x   super(props);
10 1x   this.state = { persons: [] };
11    }
12
13    loadPersons() {
14 1x   getPersons()
15      .then(result => {
16 1x     this.setState({ persons: result.data });
17      })
18      .catch(error => toastr.error(error));
19    }
20
21    componentDidMount() {
22 1x   this.loadPersons();
23    }
24
25    deletePerson() {
26      deletePerson()
27        .then(response => {
28          toastr.success(response.data);
29          this.loadPersons();
30        })
31        .catch(error => toastr.error(error));
32    }
33
34    render() {
35 2x   const persons = this.state.persons;
36
37 2x   return (
38      <div>
39        <table className="table">
40          <thead>
41            <tr>
42              <th>ID</th>
43              <th>First Name</th>
44              <th>Last Name</th>
45              <th>Email</th>
46              <th> </th>
47            </tr>
48          </thead>
49          <tbody>
50            {persons.map(person => (
51 15x         <PersonListRow key={person.id} person={person} />
52            ))}
53          </tbody>
54        </table>
55        <div>
56          <Link to="/persons" onClick={() => this.deletePerson()}>
57            Delete last person
58          </Link>
59        </div>
60      </div>
61    );
62  }
63  }
64
```

Code coverage generated by istanbul at Tue Jun 25 2019 10:40:51 GMT+0200 (CEST)

Getting Started

- npm install --save-dev jest
- npm install --save-dev @types/jest
- {script: { “test”: “jest” }} // package.json
- npm test

- npx create-react-app react-project-name // babel-jest and webpack
- npm test

Example

```
function sum(a, b) {
  return a + b;
}

module.exports = sum;

```

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

describe('sum', () => {
  it('adds 1 + 2 to equal 3', () => {
    expect(sum(1, 2)).toBe(3);
  });

  it('adds 1 + -2 to equal -1', () => {
    expect(sum(1, -2)).toBe(-1);
  });
});

```

02

Basic Unit Testing

Arrange Act Assert (AAA)

- **Arrange** describes whatever setup is needed
- **Act** describes the subject's behavior that's under test (and typically only describes a single line needed to invoke that behavior)
- **Assert** describes the verification that the subject's behavior had the desired effect by evaluating its return value or measuring a side-effect (with a spy or mock)

Case 1: Time Formatter

- Give a positive second number S,
  please format it with a proper unit by the following rule
- If seconds S < 60, return { unit: “seconds”, time: S }
- If seconds S >= 60 and S < 3600
  return { unit: “minutes”, time: parseInt(S / 60)}
- If seconds S > 3600
  return { unit: ”hours”, time: parseInt(S / 3600)}

```
1  // time-format.test.js
2  import { toProperUnit } from './time-format';
3
4  describe('time-format', () => {
5    describe('toProperUnit()', () => {
6      it('should return seconds if seconds < 1 min', () => {
7        const seconds = 30;
8
9        const result = toProperUnit(seconds);
10
11       expect(result).toEqual({
12         time: 30,
13         unit: 'seconds',
14       });
15     });
16   });
17 });
18
```

```
1  // time-format.js
2  const MINUTE = 60;
3
4  export function toProperUnit(seconds) {
5    if (seconds < MINUTE) {
6      return {
7        time: seconds,
8        unit: 'seconds',
9      };
10   }
11   return null;
12 }
13
14 export default {
15   toProperUnit,
16 };
17
```

```
1  // time-format.test.js
2  import { toProperUnit } from './time-format';
3
4  describe('time-format', () => {
5    describe('toProperUnit()', () => {
6      it('should return seconds if seconds < 1 min', () => {
7        const seconds = 30;
8
9        const result = toProperUnit(seconds);
10
11       expect(result).toEqual({
12         time: 30,
13         unit: 'seconds',
14       });
15     });
16   });
17 });
18
```

```
// time-format.test.js
import { toProperUnit } from './time-format';

describe('time-format', () => {
  describe('toProperUnit()', () => {
    it('should return seconds if seconds < 1 min', () => {
      const seconds = 30;

      const result = toProperUnit(seconds);

      expect(result).toEqual({
        time: 30,
        unit: 'seconds',
      });
    });

    it('should return minutes if seconds >= 1 min and seconds < 1 hour', () => {
      const seconds = 300;

      const result = toProperUnit(seconds);

      expect(result).toEqual({
        time: 5,
        unit: 'minutes',
      });
    });
  });
});

```

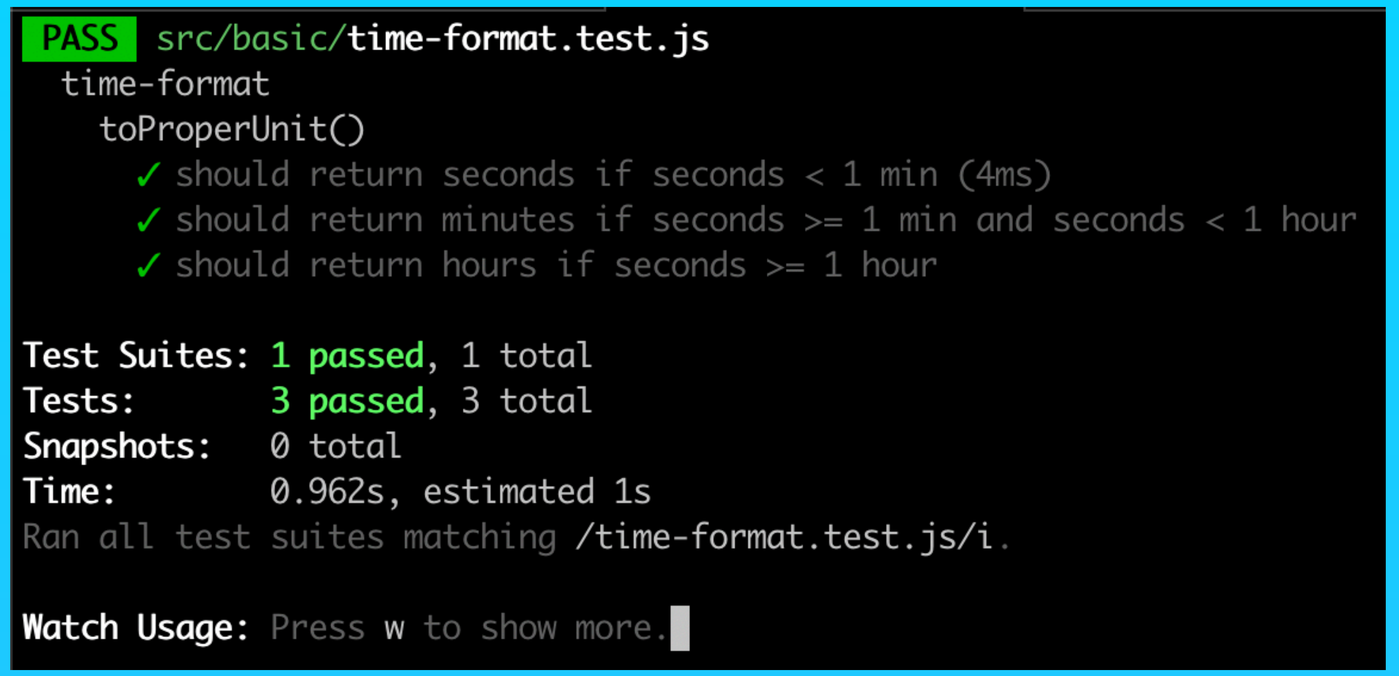
```js
// time-format.js
const MINUTE = 60;
const HOUR = 3600;

export function toProperUnit(seconds) {
  if (seconds < MINUTE) {
    return {
      time: seconds,
      unit: 'seconds',
    };
  }
  if (MINUTE <= seconds && seconds < HOUR) {
    return {
      time: parseInt(seconds / MINUTE, 10),
      unit: 'minutes',
    };
  }
}

export default {
  toProperUnit,
};

```

```js
// time-format.test.js
import { toProperUnit } from './time-format';

describe('time-format', () => {
  describe('toProperUnit()', () => {
    it('should return seconds if seconds < 1 min', () => {
      const seconds = 30;

      const result = toProperUnit(seconds);

      expect(result).toEqual({
        time: 30,
        unit: 'seconds',
      });
    });

    it('should return minutes if seconds >= 1 min and seconds < 1 hour', () => {
      const seconds = 300;

      const result = toProperUnit(seconds);

      expect(result).toEqual({
        time: 5,
        unit: 'minutes',
      });
    });
  });
});

```

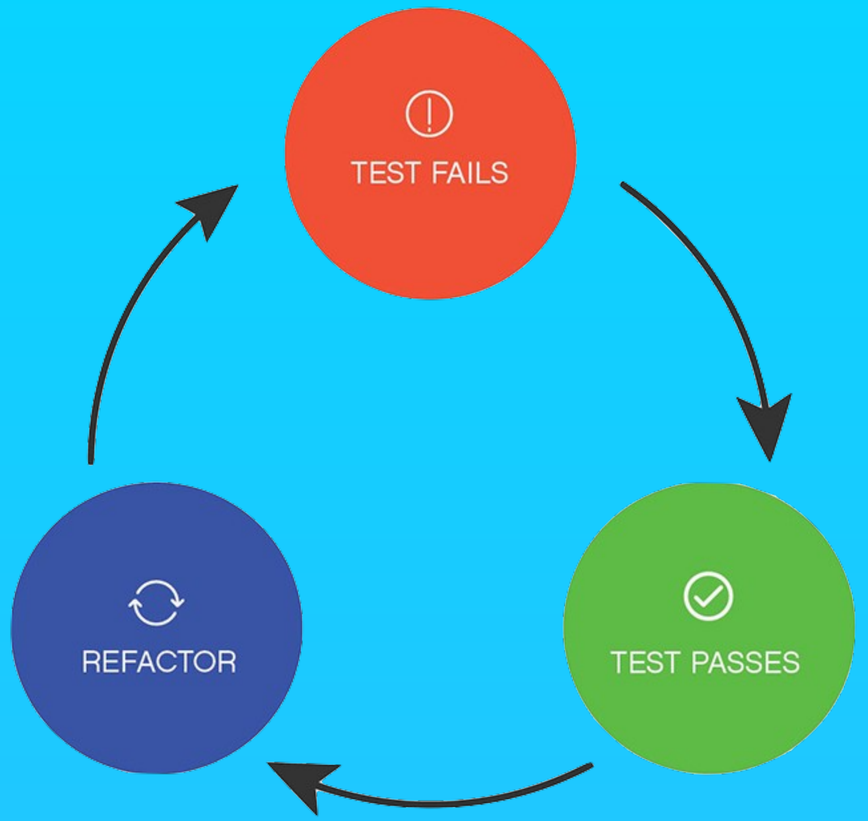
```js
// time-format.test.js
import { toProperUnit } from './time-format';

describe('time-format', () => {
  describe('toProperUnit()', () => {
    it('should return seconds if seconds < 1 min', () => {
      const seconds = 30;

      const result = toProperUnit(seconds);

      expect(result).toEqual({
        time: 30,
        unit: 'seconds',
      });
    });

    it('should return minutes if seconds >= 1 min and seconds < 1 hour', () => {
      const seconds = 300;

      const result = toProperUnit(seconds);

      expect(result).toEqual({
        time: 5,
        unit: 'minutes',
      });
    });

    it('should return hours if seconds >= 1 hour', () => {
      const seconds = 36000;

      const result = toProperUnit(seconds);

      expect(result).toEqual({
        time: 10,
        unit: 'hours',
      });
    });
  });
});

```

```js
// time-format.js
const MINUTE = 60;
const HOUR = 3600;

export function toProperUnit(seconds) {
  if (seconds < MINUTE) {
    return {
      time: seconds,
      unit: 'seconds',
    };
  }
  if (MINUTE <= seconds && seconds < HOUR) {
    return {
      time: parseInt(seconds / MINUTE, 10),
      unit: 'minutes',
    };
  }
  return {
    time: parseInt(seconds / HOUR, 10),
    unit: 'hours',
  };
}

export default {
  toProperUnit,
};

```

```js
// time-format.test.js
import { toProperUnit } from './time-format';

describe('time-format', () => {
  describe('toProperUnit()', () => {
    it('should return seconds if seconds < 1 min', () => {
      const seconds = 30;

      const result = toProperUnit(seconds);

      expect(result).toEqual({
        time: 30,
        unit: 'seconds',
      });
    });

    it('should return minutes if seconds >= 1 min and seconds < 1 hour', () => {
      const seconds = 300;

      const result = toProperUnit(seconds);

      expect(result).toEqual({
        time: 5,
        unit: 'minutes',
      });
    });

    it('should return hours if seconds >= 1 hour', () => {
      const seconds = 36000;

      const result = toProperUnit(seconds);

      expect(result).toEqual({
        time: 10,
        unit: 'hours',
      });
    });
  });
});

```


PASS src/basic/time-format.test.js
time-format
toProperUnit()
✓ should return seconds if seconds < 1 min (4ms)
✓ should return minutes if seconds >= 1 min and seconds < 1 hour
✓ should return hours if seconds >= 1 hour
Test Suites: 1 passed, 1 total
Tests: 3 passed, 3 total
Snapshots: 0 total
Time: 0.962s, estimated 1s
Ran all test suites matching /time-format.test.js/i.
Watch Usage: Press w to show more.

TDD Cycle

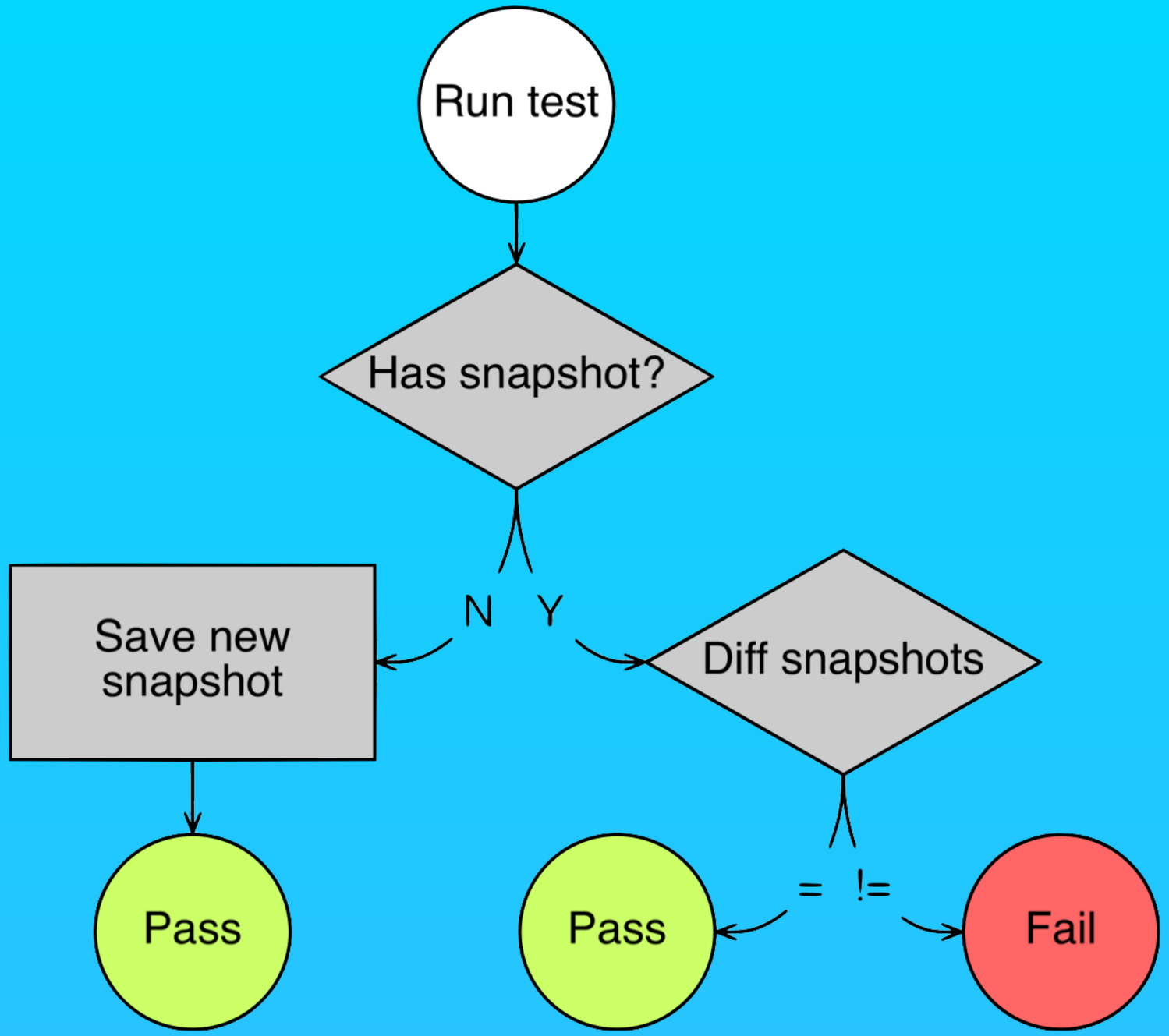


03

Snapshot Testing

Snapshot Testing

Test to make sure your UI does not change unexpectedly


Run test
Has snapshot?
N
Y
Save new snapshot
Diff snapshots
Pass
=
!=
Pass
Fail

Why Snapshot Testing

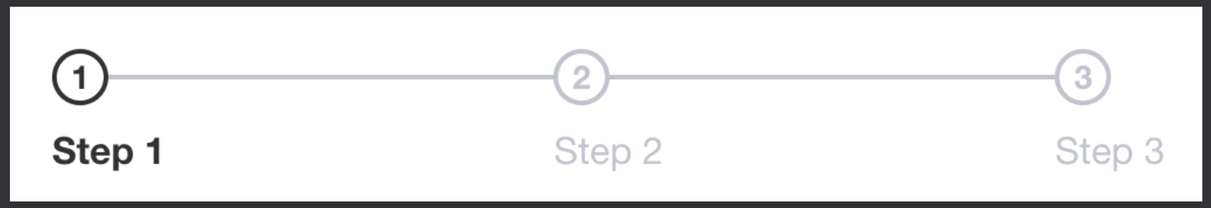
- DOM Tree, CSS Styles or JSON are hard to assert
- Make sure UI components are not changed unexpectedly
- Set up steps and running speed are faster than visual snapshots tool

react-test-renderer

- npm install --save-dev react-test-renderer
- expect(tree).toMatchSnapshot();
- jest --updateSnapshot

Case 2: JobSteps

- **JobSteps** is a React component which guides the user to complete tasks in accordance with the process
- **Total** is a prop which means how much steps user have to complete
- **Step** is a prop which means user is doing the job in which step



```jsx
const JobSteps = ({ step, total }) => {
  if (step === null) {
    return null;
  }

  const stepItems = [];
  for (let i = 1; i <= total; i += 1) {
    const circleClass = classnames('circle', {
      'circle-active': step === i,
    });
    stepItems.push(
      <span className="step" key={i}>
        {i !== total ? <span className="line" /> : null}
        <span className={circleClass}>
          {i}
        </span>
      </span>,
    );
  }

  return <span className="container">{stepItems}</span>;
};
```

```
1 import React from 'react';
2 import { create } from 'react-test-renderer';
3
4 import JobSteps from './JobSteps';
5
6 describe('<JobSteps/>', () => {
7   it('should render nothing if step is null', () => {
8     const tree = create(<JobSteps step={null} total={3} />).toJSON();
9
10    expect(tree).toMatchSnapshot();
11  });
12
13  it('should highlight step 1 if job is in step 1', () => {
14    const tree = create(<JobSteps step={1} total={3} />).toJSON();
15
16    expect(tree).toMatchSnapshot();
17  });
18 });
19
```

```
// Jest Snapshot v1, https://goo.gl/fbAQLP

exports[`<JobSteps/> should highlight step 1 if job is in step 1 1`] = `
<span
  className="container"
>
  <span
    className="step"
  >
    <span
      className="line"
    />
    <span
      className="circle circle-active"
    >
      1
    </span>
  </span>
  <span
    className="step"
  >
    <span
      className="line"
    />
    <span
      className="circle"
    >
      2
    </span>
  </span>
  <span
    className="step"
  >
    <span
      className="circle"
    >
      3
    </span>
  </span>
</span>
`;

exports[`<JobSteps/> should render nothing if step is null 1`] = `null`;

```

Best Practices

- Treat snapshots as code (snapshot files should be committed)
- Use descriptive snapshot names
- Test component should not be updated often
- Disallow large snapshots
- styled-components/jest-styled-components

04

Testing Library

Testing Library

Simple and complete testing utilities
that encourage good testing practices

@testing-library & enzyme

	@testing-library	enzyme
Support Framework	DOM, React.js, React Native, Vue.js, Angular…and more	React.js, preact
Support React.js Version	^16.8.0	^0.13.0
Test Concept	Resemble the way your software is used	More API and Feature Shallow Rendering
Assertions	@testing-library/jest-dom	jest-enzyme

Testing Library

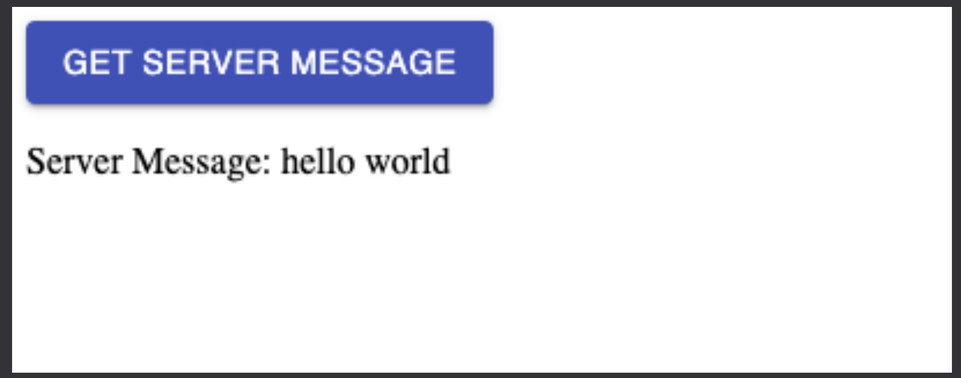
- Queries
- Firing Events
- Async Utilities
- jest-dom/expect

@testing-library setup

- npm install --save-dev @testing-library/react
- npm install --save-dev @testing-library/jest-dom
- import '@testing-library/jest-dom/extend-expect'

Case 3: Get Server Message

- **Given** a button which is named Get Server Message
- **When** user clicks the button
- **Then** front-end will request server message to back-end
- **And** front-end will display server message from back-end



Why we have to mock Response

- Because we are making unit test
- Network response slow down the test speed
- Your API may cost money
- Unit test should be deterministic

```jsx
import React, { useState } from 'react';
import { getServerMessage } from './api';

const GetServerMessage = () => {
  const [serverMessage, setServerMessage] = useState(null);

  const handleMessageClick = async () => {
    const { data } = await getServerMessage();
    const [key] = Object.keys(data);
    const message = data[key];
    setServerMessage(`${key} ${message}`);
  };

  return (
    <div>
      <button type="button" onClick={handleMessageClick}>
        Get Server Message
      </button>
      <div style={{ marginTop: '1rem' }}>
        <p>
          Server Message:
          {' '}
          {serverMessage}
        </p>
      </div>
    </div>
  );
};

export default GetServerMessage;

```

```
import React from 'react';
import { render, fireEvent, waitForElement } from '@testing-library/react';
import axios from 'axios';

import GetServerMessage from './GetServerMessage';

jest.mock('axios');

describe('<GetServerMessage />', () => {
  it('should display empty server message', async () => {
    const { getByText } = render(
      <GetServerMessage />,
    );

    const message = getByText('Server Message:');

    expect(message).toBeInTheDocument();
  });

  it('should get server message after user clicks the button', async () => {
    const { getByText } = render(
      <GetServerMessage />,
    );
    const button = getByText('Get Server Message');
    axios.get.mockResolvedValue({ data: { hello: 'testing' } });

    fireEvent.click(button);

    const message = await waitForElement(() => getByText('Server Message: hello testing'));

    expect(message).toBeInTheDocument();
  });
});

```

enzyme setup

- npm install --save-dev enzyme enzyme-adapter-react-16
- npm install --save-dev enzyme-adapter-react-16
- npm install --save-dev jest-enzyme

```
1 import Enzyme from 'enzyme';
2 import Adapter from 'enzyme-adapter-react-16';
3 import 'jest-enzyme/lib/index';
4
5 Enzyme.configure({ adapter: new Adapter() });
6
```

```jsx
import React from 'react';
import { act } from 'react-dom/test-utils';
import { mount } from 'enzyme';
import axios from 'axios';

import GetServerMessage from '../testing-library/GetServerMessage';

const wait = async (amount = 0) => {
  await act(async () => {
    await new Promise((resolve) => setTimeout(resolve, amount));
  });
};

jest.mock('axios');

describe('<GetServerMessage />', () => {
  it('should display empty server message', async () => {
    const wrapper = mount(<GetServerMessage />);

    const p = wrapper.find('p');

    expect(p).toHaveText('Server Message: ');
  });

  it('should get server message after user clicks the button', async () => {
    const wrapper = mount(<GetServerMessage />);
    const button = wrapper.find('button');
    axios.get.mockResolvedValue({ data: { hello: 'testing' } });

    button.simulate('click');
    await wait();

    const p = wrapper.find('p');
    expect(p).toHaveText('Server Message: hello testing');
  });
});

```

Enzyme Shallow Rendering

- https://airbnb.io/enzyme/docs/api/shallow.html

05

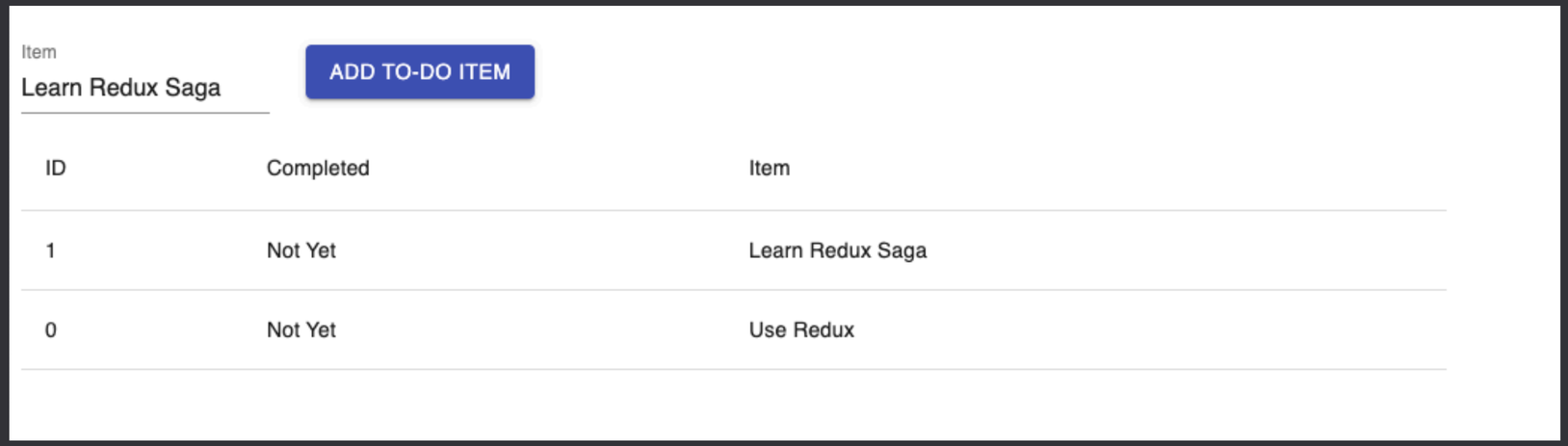
Redux Unit Testing

Redux Unit Testing

- Action
- Reducer
- Redux Container

Case 4: To-Do List

- **Action**: ADD_TODO
- **Reducer**: ADD_TODO, initState
- **Redux Container**: Click add button and add 1 item



```
1 import { addTodo } from './actions';
2 import { ADD_TODO } from './actionTypes';
3
4 describe('actions', () => {
5   it('should create an action to add a todo', () => {
6     const text = 'Finish docs';
7
8     const action = addTodo(text);
9
10    expect(action).toEqual({
11      type: ADD_TODO,
12      text: 'Finish docs',
13    });
14  });
15 });
16
```

```
export const addTodo = (text) => ({
  type: 'ADD_TODO',
  text,
});

export default undefined;

```

```
import { addTodo } from './actions';
import { ADD_TODO } from './actionTypes';

describe('actions', () => {
  it('should create an action to add a todo', () => {
    const text = 'Finish docs';

    const action = addTodo(text);

    expect(action).toEqual({
      type: ADD_TODO,
      text: 'Finish docs',
    });
  });
});

```

```
import reducers from './reducers';
import { addTodo } from './actions';

describe('todos reducers', () => {
  it('should return the initial state', () => {
    const state = undefined;
    const action = {};

    const newState = reducers(state, action);

    expect(newState).toEqual([
      {
        text: 'Use Redux',
        completed: false,
        id: 0,
      },
    ]);
  });

  it('should handle ADD_TODO', () => {
    const state = [
      {
        text: 'Use Redux',
        completed: false,
        id: 0,
      },
    ];
    const action = addTodo('Run the tests');

    const newState = reducers(state, action);

    expect(
      newState,
    ).toEqual([
      {
        text: 'Run the tests',
        completed: false,
        id: 1,
      },
      {
        text: 'Use Redux',
        completed: false,
        id: 0,
      },
    ]);
  });
});

```

```
import { ADD_TODO } from './actionTypes';

const initialState = [
  {
    text: 'Use Redux',
    completed: false,
    id: 0,
  },
];

export default (state = initialState, action) => {
  switch (action.type) {
    case ADD_TODO:
      return [
        {
          id: state.reduce((maxId, todo) => Math.max(todo.id, maxId), -1) + 1,
          completed: false,
          text: action.text,
        },
        ...state,
      ];
    default:
      return state;
  }
};

```

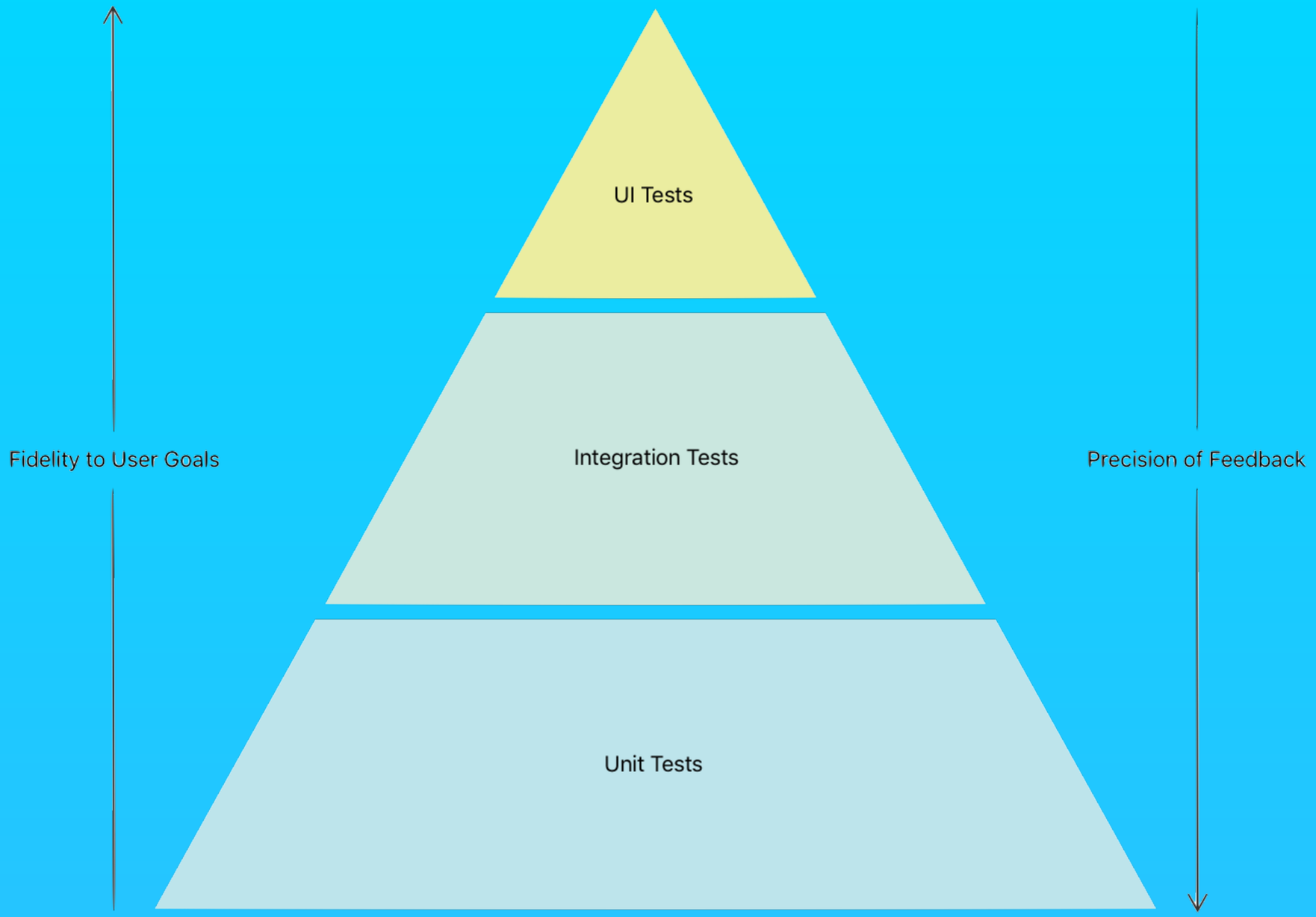
```
import reducers from './reducers';
import { addTodo } from './actions';

describe('todos reducers', () => {
  it('should return the initial state', () => {
    const state = undefined;
    const action = {};

    const newState = reducers(state, action);

    expect(newState).toEqual([
      {
        text: 'Use Redux',
        completed: false,
        id: 0,
      },
    ]);
  });

  it('should handle ADD_TODO', () => {
    const state = [
      {
        text: 'Use Redux',
        completed: false,
        id: 0,
      },
    ];
    const action = addTodo('Run the tests');

    const newState = reducers(state, action);

    expect(
      newState,
    ).toEqual([
      {
        text: 'Run the tests',
        completed: false,
        id: 1,
      },
      {
        text: 'Use Redux',
        completed: false,
        id: 0,
      },
    ]);
  });
});

```

```jsx
import React, { useState } from 'react';
import { useSelector, useDispatch } from 'react-redux';
import {
  Box, Button, TextField, Grid, Table, TableHead, TableBody, TableRow, TableCell,
} from '@material-ui/core';

import { addTodo } from './actions';

export default () => {
  const todo = useSelector(((state) => state));
  const dispatch = useDispatch();
  const addTodoAction = (text) => dispatch(addTodo(text));

  const [pending, setPending] = useState('');

  const handlePendingChange = ({ currentTarget: { value = '' } }) => {
    setPending(value);
  };

  const handleAddClick = () => {
    addTodoAction(pending);
  };

  return (
    <Grid container>
      <Grid
        container
        justify="flex-start"
        alignItems="center"
      >
        <Box mr={3}>
          <TextField
            id="standard-basic"
            label="Item"
            margin="normal"
            value={pending}
            onChange={handlePendingChange}
          />
        </Box>
        <Button variant="contained" color="primary" onClick={handleAddClick}>
          Add To-Do Item
        </Button>
      </Grid>
      <Grid item sm={6}>
        <Table aria-label="To-Do Table">
          <TableHead>
            <TableRow>
              <TableCell>ID</TableCell>
              <TableCell>Completed</TableCell>
              <TableCell>Item</TableCell>
            </TableRow>
          </TableHead>
          <TableBody>
            {todo.map((item) => (
              <TableRow key={item.id}>
                <TableCell>{item.id}</TableCell>
                <TableCell>{item.completed ? 'Done' : 'Not Yet'}</TableCell>
                <TableCell>{item.text}</TableCell>
              </TableRow>
            ))}
          </TableBody>
        </Table>
      </Grid>
    </Grid>
  );
};

```

```jsx
import React from 'react';
import { render, fireEvent } from '@testing-library/react';
import { Provider } from 'react-redux';
import { createStore } from 'redux';

import ToDoList from './ToDoList';
import reducers from './reducers';

const store = createStore(reducers);
const renderComponent = () => render(
  <Provider store={store}>
    <ToDoList />
  </Provider>,
);

describe('<ToDoList />', () => {
  it('should render init state: Use Redux', () => {
    const { getByText } = renderComponent();

    const cell = getByText('Use Redux');

    expect(cell).toBeInTheDocument();
  });

  it('should render added item after user adds 1 item', () => {
    const { getByText, getByLabelText } = renderComponent();
    const button = getByText('Add To-Do Item');
    const input = getByLabelText('Item');
    fireEvent.change(input, { target: { value: 'Add Testing' } });

    fireEvent.click(button);

    const cell = getByText('Add Testing');
    expect(cell).toBeInTheDocument();
  });
});

```

Test redux without <Provider />

- https://redux.js.org/recipes/writing-tests


UI Tests
Integration Tests
Unit Tests
Fidelity to User Goals
Precision of Feedback

06

More Testing

More and More

- reselect
- redux-saga
- redux-saga-request
- i18n-next
- React Hooks
- Storybook Testing
- Chromatic - Visual Regression Testing

End to End Testing

- **Cypress:** https://www.cypress.io/
- **Robot Framework:** https://robotframework.org/
- **Playwright:** https://playwright.dev/

Timeline of Auto Testing

潛伏期

- 沒人想維護
- 只能手動測試
- 大量 hotfix
- 推卸責任
- 相信重寫就會改善只是沒時間重寫

撞牆期

- 有人開始寫
- 各寫各的
- 壞了沒人修
- 「寫自動測試讓開發變得更慢」

成長期

- Hotfix／bug減少
- Tests 量衝起來
- 相互討教
- 需要修別人的test
- 討論規範
- 人緣變好

穩定期

- 追求綠燈長亮
- 追求可讀性
- 勇敢 refactor
- 相信撰寫測試其實是開發的一部份
- 內建品質

Questions

- Are you free to refactor code ?
- Have you confidence to release your untested code ?
- How to measure the quality of code ?

Example Code

- **GitHub Repo:** https://github.com/jimmy-liao-gogoro/react-unit-testing
- **GitHub Action:** https://github.com/jimmy-liao-gogoro/react-unit-testing/actions

Reference

- **Unit Testing Questions:** https://eggjs.org/en/core/unittest.html
- **AAA:** https://github.com/testdouble/contributing-tests/wiki/Arrange-Act-Assert
- **TDD:** https://medium.com/swlh/learning-to-love-tdd-f8eb60739a69
- **Snapshot Testing:** https://medium.com/@luisvieira_gmr/snapshot-testing-react-components-with-jest-3455d73932a4
- **Testing Pyramid:** https://developer.apple.com/documentation/xcode/testing_your_xcode_project

Q & A

Thank you for listening