Wybrane elementy praktyki projektowania oprogramowania

Wykład 15/15
Testy jednostkowe, React

Wiktor Zychla 2021/2022

1 Spis treści

2 Testy jednostkowe....2
3 Testy aplikacji webowych....4
3.1 Selenium....4
3.2 Puppeteer....4
4 React....6

2 Testy jednostkowe

Testowanie kodu pozwala wykrywać błędy implementacji jeszcze przed wdrożeniem. Szczególnymi testami są testy jednostkowe czyli testy kodu, które są wyrażone w kodzie. Przygotowanie testów jednostkowych to próba odtworzenia takich przepływów kontroli w testowanym kodzie, które mogą wystąpić w rzeczywistości. Dobre testy jednostkowe charakteryzują się dużym tzw. *pokryciem* kodu, czyli uwzględnieniem możliwe dużej liczby możliwych danych testowych, na których kod wykonuje się różnymi ścieżkami wykonania.

W przypadku Node.js jest bardzo wiele frameworków wspierających testowanie. Jednym z nich jest mocha.

Framework najwygodniej zainstalować globalnie

```
npm install mocha -g
```

a następnie w podfolderze **test** dodać jeden lub wiele plików, które mocha wykona jako testy.

Przykład

```
var assert = require('assert');

function add() {
  return Array.prototype.slice.call(arguments).reduce(function(prev, curr) {
    return prev + curr;
  }, 0);
}

describe('Top level', function() {

  before(function() {
    // runs before all tests in this block
  });

  after(function() {
    // runs after all tests in this block
  });

  beforeEach(function() {
    // runs before each test in this block
  });

  afterEach(function() {
    // runs after each test in this block
  });

    describe('add()', function() {
        var tests = [
            {args: [1, 2],       expected: 3},
            {args: [1, 2, 3],    expected: 6},
```

```
            {args: [1, 2, 3, 4], expected: 10}
        ];

        tests.forEach(function(test) {
            it('correctly adds ' + test.args.length + ' args', function() {
            var res = add.apply(null, test.args);
            assert.equal(res, test.expected);
            });
        });
    });


    describe('Sublevel', function() {
        it( 'unit test function definition', function() {
            assert.equal( -1, [1,2,3].indexOf(4) );
        });
    });
});
```

Dla wygodny można w **package.json** w sekcji scripts zmapować klucz **test** na polecenie **mocha:**

```
  "scripts": {
    "test": "mocha"
  }
```

Wynik działania testu (czyli wykonania **npm test**)

```
Top level

  add()
    √ correctly adds 2 args
    √ correctly adds 3 args
    √ correctly adds 4 args
  Sublevel
    √ unit test function definition


  4 passing (29ms)
```

3 Testy aplikacji webowych

3.1 Selenium

Okazuje się, że testowanie aplikacji internetowych nie jest takie łatwe – stosunkowo łatwo przetestować to co dzieje się na serwerze ale jak testować silnik przeglądarki, to co widzi i z czym pracuje użytkownik?

Na te pytania długo nie było odpowiedzi.

Przez długi czas jednym z rozwiązań było korzystanie z interfejsów automatyzujących aplikacje okienkowe. W Windows był to interfejs UI Automation API. W taki sposób daje się automatyzować proste aplikacje, ale już nie – złożoną zawartość w przeglądarce.

Krokiem naprzód była pierwsza wersja platformy Selenium (2005). Pomysł był karkołomny – wykorzystać niskopoziomową wiedzę na temat tego jak działają popularne przeglądarki i przygotować API do sterowania nimi. Pomysł był zawodny, nowa wersja przeglądarki mogła powodować problemy w działaniu Selenium.

W 2012 roku podjęto udaną próbę ustandaryzowania sposobu komunikacji narzędzia z przeglądarkami – zamiast je „hackować" zaproponowano protokół komunikacyjny, WebDriver, Specjalna wersja przeglądarki (dostępne na stronie Selenium) implementują ten protokół i pozwalają na sterowanie nimi z poziomu kodu (napisanego w zasadzie w dowolnym języku).

Implementacji protokołu dla Node.js jest wiele, w tym np. Web Driver IO.

```
var webdriverio = require('webdriverio');
var options = { desiredCapabilities: { browserName: 'chrome' } };
var client = webdriverio.remote(options);
client
    .init()
    .url('https://duckduckgo.com/')
    .setValue('#search_form_input_homepage', 'WebdriverIO')
    .click('#search_button_homepage')
    .getTitle().then(function(title) {
        console.log('Title is: ' + title);
        // outputs: "Title is: WebdriverIO (Software) at DuckDuckGo"
    })
    .end();
```

3.2 Puppeteer

Puppeteer reprezentuje inne podejście – ogranicza się do jednej przeglądarki (Chromium) ale za to:

- Udostępnia całe jej API

- Pozwala automatyzować przeglądarkę bez tworzenia jej okna (w tym: bez potrzeby posiadania pulpitu w interaktywnej sesji użytkownika!)

```
const puppeteer = require('puppeteer');

(async () => {
  const browser = await puppeteer.launch();
  const page = await browser.newPage();
  await page.goto('https://example.com');
  await page.screenshot({path: 'example.png'});

  await browser.close();
})();
```

4 React

Tematem programowania aplikacji po stronie przeglądarki nie zajmowaliśmy się na wykładzie, również z tego powodu że jest to temat bardzo szeroki. Frameworki takie jak Angular, React, Vue.js czy Ext.js pozwalają na budowanie bardzo złożonych aplikacji.

Frameworkiem wartym polecenia, łatwym do nauki jest **React**.

Na poniższym przykładzie można zademonstrować kluczowe podstawy architektury:

```
<!-- react demo.html -->
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <meta http-equiv="X-UA-Compatible" content="ie=edge">
    <script crossorigin src="https://unpkg.com/react@17/umd/react.development.js"></script>
    <script crossorigin src="https://unpkg.com/react-dom@17/umd/react-
dom.development.js"></script>
    <script src=" https://unpkg.com/@babel/standalone/babel.min.js"></script>
    <title>Document</title>
    <script type="text/babel">

        class CustomList extends React.Component {
            constructor(props) {
                super(props);
            }
            render() {
                return <ol>
                {
                    this.props.items.map( item => <li key={item}>{item}</li> )
                }
                </ol>
            }
        }

        class Hello extends React.Component {

            constructor(props) {
                super(props);

                var me = this;
                this.state = { happy: true, now: new Date(), items: ['a', 'b', 'c'] };

                setInterval( function() {
                    me.setState( {now: new Date()});
                }, 1000);
            }
```

```
            render() {
                return <div>
                    <div>
                        Hello {this.props.message}
                    </div>
                    <div>
                        {this.state.happy ? (<div>I am happy</div>) : (<div>I am sad</div>)}
                    </div>
                    <div>
                        {this.state.now.toString()}
                    </div>
                    <div>
                        <CustomList items={this.state.items} />
                    </div>
                    <div>
                        <button onClick={() => this.setState({happy: !this.state.happy})}>Happy or s
ad?</button>
                    </div>
                </div>
            }
        }

        window.addEventListener('load', function() {
            const element = <Hello message=' world' />
            ReactDOM.render(element, document.getElementById('root'));
        })
    </script>
</head>
<body>
    <div id="root"></div>
</body>
</html>
```

Wśród elementów na które należy zwrócić uwagę występują tu:

- Komponenty
- Właściwości (**props**)
- Stan (**state**)
- Modyfikacja stanu (**setState**)
- Wirtualny DOM, rekoncyliacja
- JSX

Kluczowe dla okiełznania architektury dużej aplikacji jest:

- Użycie wsparcia dla zarządzania dużymi danymi – na przykład komponent Redux
- Użycie biblioteki komponentów wizualnych – na przykład Material UI

W powyższym przykładzie użyty jest komponent babel do transpilacji JSX do Javascript. Aby uniknąć takiej niedogodności (niewłaściwej dla produkcyjnych aplikacji), należy zestawić cały potok. Jednym z łatwych sposobów jest użycie pakietu Create React App.