UNIVERSITÄT LEIPZIG

SOFTWARETECHNIK-PRAKTIKUM

Qualitätssicherungskonzept

SPE16: Gebäude-Navigator für Leipzig

18. Juli 2016

Inhaltsverzeichnis:

0. Qualitätssicherungskonzept

Die Qualitätssicherung dieses Projektes hält in diesem Dokument fest, wie die Dokumentation sowie die Tests zum erfolgreichen Erstellen und zur nachhaltigen Erweiterbarkeit der Software gewährleistet werden.

1. Dokumentationskonzept

Die Dokumentation des Quelltexts ist eine der wichtigsten Aufgaben beim Erstellen der Tests sowie der Wartung und Erweiterbarkeit eines Softwareprojekts. Um diese Ziele zu erreichen, ist eine genaue Beschreibung der Software nötig. Dies beinhaltet die Dokumentation im Quelltext sowie einer genauen quelltextnahe strukturierte Dokumentation mit Hilfe eines geeigneten Werkzeugs.

1.1. Interne Dokumentation

Die im Projekt SPE16 verwendete Skriptsprache ist JavaScript. Die Dokumentation des Quelltexts wird durch die in der Skriptsprache festgelegte Zeichenfolge ermöglicht. Diese Zeichenfolge wird im folgendem dargestellt:

1. Zum Kommentieren einer Zeile werden zwei Slashs (/) verwendet. Diese können eine Zeile auskommentieren oder aber am Ende der Zeile stehen um einen erklärenden Kommentar zu hinterlassen.

```
//Dies ist ein einzeiliger Kommentar.
```

2. Weiterhin können ganze Blöcke im Quellcode eingeklammert werden. Ein Slash, gefolgt von einem Stern (*), öffnet einen Block. Zum Schließen eines Blocks wird der Stern vor dem Slash platziert.

```
/*
Dies ist ein
mehrzeiliger Kommentar.
*/
```

Diese Zeichenfolge soll bei der Implementierung verwendet werden im den Quelltext intern zu dokumentieren. Vom Programmierer, der die Implementierung durchführt, ist darauf zu achten, dass der Quellcode, der nicht selbsterklärend ist, durch die oben genannte Zeichenfolge verständlich zu beschreiben ist.

1.2. Quelltextnahe strukturierte Dokumentation

Zu der internen Dokumentation des Quelltexts wird im Rahmen des Projekts eine quelltextnahe strukturierte Dokumentation angelegt. Diese sorgfältig angelegte Dokumentation im Quelltext wird durch ein weiteres, Skriptsprachen unabhängiges, Werkzeug erstellt. Dieses Werkzeug, verwendet mit der im Projekt angewendeten Skriptsprache JavaScript, ist JSDOC. Mit diesem Hilfsmittel wird anhand einer festgelegten Syntax und Konventionen in den Kommentaren des Quelltexts eine leicht navigierbare HTML-Seite erstellt. Diese HTML-Seite beschreibt die Daten und Funktionen mit Typeninformation für Parameter oder Rückgabewerte. Die folgende Tabelle stellt einige der wichtigen Angaben in den Kommentaren dar, damit JSDOC eine quelltextnahe, strukturierte Dokumentation erstellen kann.

JSDOC-Angabe	Beschreibung
@param {Typ} Beschreibung des Parameters	Beschreibung eines Parameters mit Datentyp und Name.
@return {Typ} Beschreibung der Rückgabe	Beschreibt die Semantik der Rückgabe und den Type einer Funktion.
@fires #[event:]	Dokumentiert die Events, die eine Funktion emittieren kann.
@throws {Typ} Beschreibung	Beschreibt eventuell auftretende Ausnahmen (Exceptions) bei der Ausführung der Funktion.
@author []	Hinterlegt Angaben zum Autor.
@deprecated []	Markiert die Funktion als veraltet mit einer optionalen Beschreibung.

Diese und weitere JSDOC Angaben werden vom Programmierer in den Kommentaren eingepflegt mit dem Ziel, eine quelltextnahe, aber außerhalb des Quelltexts verfügbare Dokumentation zu erstellen. Diese dient, wie auch die interne Dokumentation, einer einfacheren Pflege und Wartung des Softwareprojekts.

2. Coding Standard

Beim Coding Standard wird der Quelltext nach festgelegten Regeln erstellt. Durch diese Regeln wird die Softwarequalität verbessert und die Wartung und Pflege werden erleichtert. Da externe Werkzeuge nur schlecht oder äußerst eingeschränkt einen guten Programmierstil überprüfen können, trägt der Programmierer die Verantwortung, diesen Programmierstil zu erstellen. Der Coding Standard in diesem Projekt orientiert sich an dem Coding Standard von airbnb für react (*https://github.com/airbnb/javascript/tree/master/react*). Auf Basis dieses Coding Standards gelten z.B. folgende Regeln:

1. class extends React.Component wird React.createClass bevorzugt:

```
class Listing extends React.Component {
  // ...
  render() {
    return <div>{this.state.hello}</div>;
  }
}
```

statt

```
const Listing = React.createClass({
  // ...
  render() {
    return <div>{this.state.hello}</div>;
  }
});
```

2. Verwendung camelCase bei Props und Variablen

```
<Foo
  userName="hello"
  phoneNumber={12345678}
/>
```

3. JSX Tags werden in Klammern geschrieben, sofern sie mehr als eine Zeile umfassen

```
render() {
  return (
    <MyComponent className="long body" foo="bar">
      <MyChild />
    </MyComponent>
  );
}
```

statt:

```
render() {
  return <MyComponent className="long body" foo="bar">
           <MyChild />
         </MyComponent>;
}
```

4. Tags, die keine Kindknoten besitzen, sind immer selbstschließend. Es wird stets ein Leerzeichen vor dem schließenden Tag gesetzt:

```
<Foo className="stuff" />
```

statt

```
<Foo className="stuff"></Foo>
```

und bei mehrzeiligen Attributen:

```
<Foo
  bar="bar"
  baz="baz"
/>
```

5. render-Methoden geben immer einen Wert zurück

```
render() {
  return (<div />);
}
```

statt

```
render() {
  (<div />);
}
```

Desweiteren gilt für den normalen JS-Code der Coding Standard von Google für die Skriptsprache JavaScript (*https://google.github.io/styleguide/javascriptguide.xml*).
Der Programmierer soll sich an diese Regeln halten, damit die Lesbarkeit von Quelltext innerhalb des Erstellungsprozesses der Software sowie weitere Arbeiten garantiert bleiben.

3. Testkonzept

3.1. Vorbemerkung zum Test in React

Da die Anwendung in react geschrieben wird bietet es sich an ein Test Framework, welches im Zusammenhang mit react entwickelt wurde, zu verwenden. React bietet von Hause aus Test Utilitys an, welche ausgewählte Ereignisse simulieren. Mit Hilfe von jest, einem Unit Testing Framework von Facebook, wird in Kombination mit dem Assertion System Jasmine 2 ein vollständiges Test Framework zur Verfügung gestellt.

3.2. Jest

Jest verbindet die Test Utilities von react und das Assertion System von Jasmine 2 um Tests auf den Komponenten durchzuführen. Um jest zu verwenden müssen bestimmte Voraussetzungen erfüllt werden. Zum einen muss ein Ordner „__tests__“ im Root-Verzeichnis angelegt werden und zum anderen eine Datei „.babelrc“ mit dem Inhalt „{"presets": ["es2015", "react"]}“ erstellen. Jest wird in Node.js mit dem Befehl „npm install --save-dev jest-cli“ installiert.

3.3. Jasmine 2

Jasmine 2 ist das Assertion System welches standardmäßig in jest verwendet wird. Jasmine 2 ist ein verhalten orientiertes Softwareentwicklungsframework bei dem anhand von bestimmten Szenarios die Module getestet werden. Der Aufbau von Jasmine 2 beginnt mit einem „describe“ welche die Test Suite beschreibt. In dem folgenden „it“ werden nach einander „expect“ ausgeführt die nacheinander Erwartungen der zu testenden Komponente mit „fail“ durchgefallen und „pass“ bestanden bewerten.

3.4. Testvorgehen

Soll eine Komponente getestet werden, so wird eine JavaScript Datei im Ordner „__tests__“ erstellt. Durch „npm test“ wird jest gestartet. Jest durchsucht den Ordner „__tests__“ nach den Tests und startet diese. In der Konsole erscheinen nach einiger Zeit die Ergebnisse der Tests.

3.5. Beispiel

Die Testdatei ist wie folgt aufgebaut:

```
jest.unmock('../src/js/components/Filter');


import React from 'react';
import ReactDOM from 'react-dom';
import TestUtils from 'react-addons-test-utils';
```

```
import Filter from '../src/js/components/Filter';


//Test Suite Body
describe("Filter", function() {
  it('should exists', function() {
    // Render into document
    var filter = TestUtils.renderIntoDocument( <Filter /> );
    //Jasmine 'toBeTruthy' matcher
    expect(TestUtils.isCompositeComponent(filter)).toBeTruthy();
  });
});
```

Dieser Test macht nichts weiter, als die Komponente *Filter* zu rendern und anschließend wird erwartet dass diese Komponente gerendert ist.