Qualitätssicherungsplan
PWS17

Frauke Beccard
Sebastian Ossinger
Nicolas Schäfer
Jonathan Eberle

16. Januar 2017

Inhaltsverzeichnis

1 Dokumentationskonzept

Für das ganze Projekt gilt, dass es auf Englisch verfasst wird, das umfasst sowohl Variablen- wie Funktions- und Modulnamen als auch die komplette Dokumentation, Projektwebsite und Commit-Messages.

1.1 Coding Standard

Der Coding Standard setzt fest, wie Quellcode innerhalb des Projektes zu strukturieren ist. Wir verwenden den Angular Style Guide [1], da dieser spezifisch fuer Angular Projekte bereits sehr genaue und durchdachte Regeln aufstellt.

Dabei ist besonders wichtig, dass Funktionalitäten möglichst atomar isoliert in Module aufgeteilt sind und die dementsprechenden Regeln zur Organisation der Dateien im Projekt eingehalten werden.

Zusätzlich legen wir folgende Richtlinien fest:

- Wir verwenden anstelle von Tabs für Einrückungen 4 Leerzeichen.
- Die Zeilenlänge sollte auf 80 Zeichen beschränkt sein
- Jede Datei sollte am Ende ein Newline enthalten

1.2 Quelltextdokumentation

Die Funktionen und Komponenten des Quelltextes müssen soweit möglich kommentiert werden, damit eine umfassende und leicht zu navigierende Dokumentation daraus generiert werden kann. Diese Dokumentation ist wichtig, da der Code unter Umständen später von anderen Leuten weiterverwendet und entwickelt werden soll und auch innerhalb des Teams schon zur notwendigen Verständigkeit beiträgt.

Für die Quelltextdokumentation wird TypeDoc verwendet, welches einfacher zu verwenden ist als die Alternative JSDoc. TypeDoc kann eine Web-Dokumentation automatisch erstellen.

Eine dokumentierte Funktion würde beispielsweise so aussehen:

```
/**
 * Kommentar zur Methode "doSomething".
 * @param target  Kommentar zum Parameter "target".
 * @returns       Kommentar zum return Wert der Funktion.
 */
function doSomething(target:any, arg:any):number {
    return 0;
}
```

Wir folgen dabei den TypeDoc-Richtlinien[2].

[1] https://angular.io/styleguide
[2] http://typedoc.org/guides/doccomments/

1.3 Quelltextinterne Kommentare

Quelltextinterne Kommentare sind generell nur dann zu verwenden, wenn eine Zeile oder ein Abschnitt ansonsten unverständlich wäre, oder um vor unerwarteten Stolperfallen zu warnen. Generell sollte in solchen Fällen aber wenn möglich der Code so refaktorisiert oder restrukturiert werden um auf solche Kommentare verzichten zu können. Falls Sie trotzdem unerlässlich sind sollten sie sich immer direkt auf der Zeile über dem betreffenden Abschnitt befinden.

1.4 Standards für die Entwurfsbeschreibung

Die Entwurfsbeschreibung ist als Textdokument mit folgender Gliederung zu führen:

1. Allgemeines (Kontext der Aufgabenstellung)
2. Produktübersicht (äußerliche Funktionsmerkmale des Systems)
3. Grundsätzliche Struktur- und Entwurfsprinzipien (Erläuterung zentraler Entscheidungen bezüglich spezieller Architekturen oder Frameworks, übergreifende Aspekte der Modellierung und Strukturierung der Software)
4. Struktur- und Entwurfsprinzipien einzelner Pakete (funktionale Aspekte)
5. Datenmodell
6. Glossar

Sie soll kompakt gehalten werden, Abbildungen und Texte sind möglichst nah an der referenzierten Textstelle einzufügen. Die Entwurfsbeschreibung legt einen begrifflichen Rahmen und eine Strukturierung der Modellierung fest und führt auf, welche funktionalen, sowie organisatorischen softwaretechnischen Prinzipien und Standards angewendet wurden.

1.5 Standards für die Projektwebsite

1.5.1 Wiki-Einträge

Da die Wiki Einträge direkt in der Projektwebsite eingebunden werden sollte auch hier ein gewisser Standard eingehalten werden. Einträge im Wiki werden (soweit möglich) im MarkDown-Format verfasst. Sie sind übersichtlich, kurz und gut strukturiert, da sie einen rein informativen Nutzen haben. Der html-Code zu Beginn jeder Seite (siehe Abb. 1) soll stets für neue Seiten übernommen werden und nicht gelöscht werden, da er wichtige Einstellungen für die Strukturierung beinhaltet. Des weiteren muss jede Wiki-Seite folgendes enthalten: eindeutiger Titel Informationsteil, indem kurz zusammengefasst wird, welches Thema die Seite beinhaltet

1.5.2 Issues

Issues müssen einen eindeutigen Titel tragen und in diesem nur eine kurze Zusammenfassung der Aufgabe enthalten. Weitere Informationen sind als Beschreibung zu formulieren. Sie können nur einen Verantwortlichen haben und sind (wenn möglich) einem angemessenen Milestone zuzuordnen. Bei sog. „Bug Issues“ muss in der Beschreibung das Problem erklärt und eine Möglichkeit es zu reproduzieren erwähnt werden.

Die Entwicklung des in einem Issue beschriebenen Features findet auf einem dem Issue eindeutig zugeordneten atomaren feature-Branch statt.

Zum schließen eines Issues muss im Kommentar der schließende Commit und eine kurze Beschreibung, warum der Issue abgearbeitet ist, anzugeben.

2 Testkonzept

2.1 Jasmine Testing Suite

Die Jasmine Testing Suite stellt verschiedene Methoden und Schemata zum Testen von Javascript Code, speziell Unit Tests zur Verfügung. Dafür wird kein weiteres Framework oder ähnliches benötigt, aber in Zusammenhang mit einem Typescript Transpiler lassen sich die Tests auch in Typescript verfassen. Die Tests in der Jasmine Testing Suite verlaufen nach dem Ässert-Prinzip", man gibt also einen Funktion an, und danach den von ihr erwarteten Ausgabewert. Für Jasmine werden Tests in der Form:

```
describe("SuiteName", () => {
    it("SpecName", () => {
        expect(true).toBe(true);
    });
});
```

Der Name der Suite sollte dabei repräsentativ dafür stehen auf welche Funktionalität getestet werden soll und der Name jedes Specs dafür, was genau überprüft wird. Eine Suite wird genau dann als "passedäusgegeben, wenn alle ihre Specs ihre jeweiligen Erwartungen erfüllen. Es gibt noch weitere Funktionen als expect() dazu Referenz bei der Jasmine 2 introduction.[3]

In Jasmine können auch die Angular Testing Utilities verwendet werden.

[3] https://jasmine.github.io/2.4/introduction.html

2.2 Angular Testing Utilities

Angular bietet über Unit Tests hinaus verschiedene Möglichkeiten, existierenden Code in seinem Interaktion mit Nutzern, dem eigenen Template oder mit anderen Komponenten zu simulieren und zu prüfen. Im Vergleich zum reinen Jasmine bieten die Angular Testing Utilities noch weitere Funktionen an um die direkte Interaktion mit Angular und seinen Komponenten zu testen. Funktionsweise: Mithilfe der TestBed-Klasse und ihrer Methode configureTestingModule wird eine Modul-Umgebung zum Testen erzeugt. Die zu testende Komponente wird deklariert und sobald die Konfiguration der Test-Umgebung abgeschlossen ist wird eine Instanz dieser Komponente erzeugt. Hierbei gibt TestBed.createComponent ein ComponentFixture zurück, dass den Zugriff auf die Instanz der Komponente sowie aus das DebugElement ermöglicht. Mit dem DebugElement kann der DOM-Baum des ComponentFixture nach Elementen durchsucht werden, die bestimmte Prädikatfunktionen erfüllen (z.B. „CSS selector predicate“).

2.3 Karma

Karma[4] ist ein Test Runner, das bedeutet er führt alle Tests aus gibt dann dem User die jeweiligen Ergebnisse aus. Karma selbst macht diese normalerweise im Hintergrund bei jeder Codeänderung und gibt sie auf einen selbst erstellten lokalen Webserver aus. Für die Implementation in der Gitlab CI besteht aber auch die Möglichkeit die Ausgabe auf die Command Line zu beschränken.

Unit Tests werden im Ordner src/app/ abgelegt und haben die Endung .spec.ts

2.4 Protractor

Protractor[5] ist ein Test Runner für End to End Tests. Bei einem End to End Test wird versucht eine User Experience nachzumodellieren und so zu testen, ob alle Teile der Application so zusammenspielen wie von ihnen erwartet wird. Dabei werden API Calls auch direkt laufen gelassen, um einer echten User Experience so nah wie möglich zu kommen. Dies steht im Gegensatz zu Unit Tests, wo die Calls eventuell durch Mock Calls ersetzt werden.

End to End Tests werden im Ordner e2e/ des Projekts abgelegt.

2.5 Code Coverage

Die Code Coverage ist ein sehr umstrittenes Thema, da Sie nicht unbedingt die Qualität an Tests wiederspiegelt und somit eigentlich keine aussagekräftige Variable ist. Deshalb haben wir uns entschieden die erwartete Coverage bei 70% für Unit Tests anzusetzen und 20% für End to End Tests. Die niedrige erwartete Coverage für End to End Tests stammt daher, dass diese potentiell viel schwieriger zu schreiben beziehungsweise überhaupt zu erstellen sind.

[4] https://karma-runner.github.io/1.0/index.html
[5] http://www.protractortest.org/

3 Organisatorisches Konzept

3.1 Termine

Jede Woche findet am Donnerstag um 13:15 ein Treffen mit Tutor und Betreuer statt. Der Ort wird jeweils intern (Slack) bekanntgegeben.

Issues für Abgaben müssen jeweils am Tag vorher bis 16:00 Uhr abgearbeitet werden, damit der Teamleiter und der Arbeitsplanverantwortliche die Abschlusskontrolle durchführen und die Abgabe entsprechend vorbereiten kann.

Am selben Abend sollten alle Mitglieder im Slack-Channel überprüfen, ob noch Fehler aufgefunden wurden, damit diese möglichst noch vor dem Abgabetermin behoben werden können.

3.2 Git

Für die Nutzung des Git-Repository verwenden wir folgenden Workflow[6]:

Auf den master-Branch werden Merges nur zu Abgabeterminen und nur bei erfolgreichen Tests und Builds durchgeführt. Darunter liegt der develop Branch, von dem für Issues eigene Branches abgezweigt und nach Fertigstellung zurückgemerged werden.

Git Commits sollten in der Beschreibung eine kurze Erklärung im Präsens haben, die beschreibt, was der Commit ändert. Zum Beispiel: add logger module for logging purposes.

3.3 Continuous Integration

In der Continuous Integration, welche über Gitlab zu Verfügung gestellt wird, soll sichergestellt werden, dass alle Builds korrekt laufen und die angestrebte Code Coverage eingehalten wird. Unit Tests sollen direkt von der CI ausgeführt werden, während End to End Tests nur auf ihre Coverage geprüft werden, weil End to End Tests leicht sehr viel Zeit in Anspruch nehmen können und so den Workflow hindern.

3.4 Sonstiges

Bei Abstimmungen gilt das einfache Mehrheitsrecht. So können Abstimmungen auch in Abwesenheit mancher Mitglieder durchgeführt werden.

[6] http://nvie.com/posts/a-successful-git-branching-model/