Inhaltsverzeichnis

Abbildungsverzeichnis

1 Einführung

1.1 Unternehmensvorstellung

Eine Vorstellung des Unternehmens Unitedprint.

1.2 Der Freedesign-Editor

Eine Vorstellung des Freedesign-Editors und seines Funktionsumfanges.

1.3 Problemstellung

Als Anfang 2018 die Entwicklung der aktuellen FreeDesign-Anwendung begonnen wurde, hatte das Team nur wenig Erfahrung im Entwickeln von ReactJS-Anwendungen. Des Weiteren wurde die Anwendung unter einem hohen zeitlichen Druck entwickelt. Dadurch sind eine Reihe von technischen Schulden entstanden. Eine der Hauptschulden ist eine fehlende Definition der Quelltext-Architektur. Durch die Verwendung von ReactJS und Redux wird zwar bereits eine gewisse Architektur vorgeben, diese bezieht sie jedoch auf die Strukturierung der grafischen Oberfläche. Für die Domain-Logik wurde jedoch keine spezifische Architektur festgelegt, was die Pflege und Weiterentwicklung der Anwendung, erschweren kann. Die aktuelle Architektur weist derzeit folgende offensichtliche Schwächen auf:

- Die Architektur ist nicht dokumentiert.
- Der Quelltext für die grafische Oberfläche und für die Domain-Logik sind mitunter viel zu eng gekoppelt, was den Austausch und die Aktualisierung von JavaScript- Bibliotheken

erschwert.

- Durch die zuvor genannte enge Kopplung ist es für einige Teile des Quelltextes schwer Unit-Tests zu erstellen bzw. zu pflegen.
- Einige Teile des Quelltextes weisen Muster von Anti-Patterns auf.

Da die Anwendung einer permanenten Weiterentwicklung unterliegt, ist es wichtig die Software in eine geeignetere Architektur zur überführen. Weiterhin entwickelt sich die Webtechnologie mit großer Geschwindigkeit weiter. An dieser Stelle ist eine Architektur notwendig, die eine effiziente Pflege ermöglicht.

1.4 Ziel der Diplomarbeit

Das Ziel der Diplomarbeit ist die Ausarbeitung eines Vorgehens zur Überführung einer Ist-Architektur einer, in TypeScript implementierten, ReactJS-Anwendung in eine Soll-Architektur. Um eine hohe Akzeptanz einer solcher Maßnahme zu erreichen, ist eine Rahmenbedingung, dass die Überführung schrittweise geschieht und die Weiterentwicklungsarbeit der ReactJS-Anwendung begleitet.

2 Grundlagen

Zusammenfassung der technischen Grundlagen, die für das Verständnis des Freedesign-Editors wichtig sind.

2.1 Unit-Testing

2.1.1 Definition

Anlehnend an Osherove (2015, S. 34) ist ein Unit-Test ein Stück Quelltext, welcher automatisch einen isolierten Baustein, das Testobjekt, einer Software ausführt. Dabei kann das Testobjekt ein Skript, eine Funktion, ein Modul oder bei objektorientierten Sprachen eine Klasse sein. Wichtig ist, dass das Testobjekt abgeschlossen ist. Nach der Ausführung prüft der Unit-Test das Ergebnis, welches vom Testobjekt erzeugt wurde. Entspricht das Ergebnis den vom Test definierten Erwartungen, wird das Testergebnis als erfolgreich gekennzeichnet, andernfalls wird es als negativ gekennzeichnet. Das Testergebnis ist konsistent, solange die Logik des Testobjektes nicht geändert wird.

2.1.2 Anforderungen an Unit-Tests

Unabhängigkeit der Unit-Tests

Die Entwicklung einer Software sollte von regelmäßigen Ausführungen und Erweiterungen der Unit-Tests durch die Entwickler begleitet sein. Damit die Tests regelmäßig von den Entwicklern ausgeführt werden, sollte ihre Ausführung Millisekunden bis maximal wenige Sekunden lang dauern. Andernfalls würde die Akzeptanz der Tests bei den Entwicklern gesenkt. Die Aus-

führung der Tests sollte auch die Entwicklungsarbeit nicht unterbrechen oder gar blockieren. Um dies zu gewährleisten, ist es wichtig, dass die Unit-Tests unabhängig von ihrer Umgebung ausführbar sind und nicht aufeinander aufbauen (vgl. Springer 2015, S. 19). Das bedeutet, dass keine Datenbankverbindung oder Verbindung zu anderen Systemen bestehen darf. Diese Verbindungen würden es notwendig machen, die Testumgebung zu konfigurieren und würden die Testsausführung verlangsamen oder auch stören. Das Testen mit Datenbankenverbindungen und weiteren externen Abhängigkeiten wird den Integrationstests und den Systemtests überlassen. Durch die schnelle Ausführbarkeit muss der Entwickler auch nicht jedesmal die gesamte Applikation starten, um seine Arbeit zu prüfen.

Leichte Verständlichkeit

Eine weitere wichtige Anforderung an Unit-Tests ist, dass sie leicht verständlich programmiert sind und sich somit jedem Entwickler, der im Projekt involviert ist, innerhalb weniger Minuten erschließen (vgl. Osherove 2015, S. 224). Somit können eventuelle Fehlfunktionen, die zu fehlerhaften Unit-Tests führen, schnell vom Entwickler erfasst werden. Das hilft den involvierten Entwicklern den Quelltext der Software leichter zu verstehen. Unit-Tests können somit auch als Teil der Softwaredokumentation für die Entwickler gesehen werden (vgl. Springer 2015, S. 19).

Da Software aus einer Vielzahl von Funktionen und Funktionalitäten besteht, ist ebenfalls mit einer Vielzahl von Unit-Tests zu rechnen. Aus diesem Grund sollten die Tests von Anfang an in einer sinnvollen Struktur angelegt werden und mit Techniken ausgestattet sein, welche die Wartung der Tests möglichst gering hält.

Vermeidung von Logik in den Unit-Tests

Innerhalb der Unit-Tests sollte auf Logik verzichtet werden (vgl. Osherove 2015, S. 197). Diese führt nicht nur zur schweren Verständlichkeit der Tests, sondern macht die Unit-Tests selber anfällig für Fehler.

2.1.3 Mehraufwand durch das Erstellen von Unit-Tests

Das Erstellen von Unit-Tests bedeutet einen nicht unerheblichen Mehraufwand für die Programmierarbeit, was dazu führen kann, dass Entwickler auf den Einsatz von Unit-Tests verzichten. Im Entwickler-Team des Unternehmens Unitedprint wird bereits ein Code-Review-Prozess durchgeführt, bei dem Quelltextänderungen vor dem Einsatz im Produktivsystem von weiteren Entwicklern auf Fehler überprüft wird. Dieser Prozess sollte um die Überprüfung der Unit-Tests erweitert werden. Es muss jedoch auch jedem Entwickler bewusst sein, dass der Einsatz von Unit-Tests sinnvoll ist.

2.2 Refactoring

2.3 Verwendete Technologien

Beschreibung der Technologien die zum Einsatz kommen.

2.3.1 TypeScript

Kurze Beschreibung der Programmiersprache TypeScript.

2.3.2 Webpack

Beschreibung des Bundling-Prozess durch Webpack.

2.3.3 ReactJS

Beschreibung der UI-Bibliothek ReactJS.

2.3.4 Redux

Beschreibung des Zustandsmanager Redux.

2.3.5 Jest und Enzyme

Beschreibung der Bibliotheken, die für Unit-Tests zum Einsatz kommen.

2.4 Anwendungsumgebung

Beschreibung wie der Freedesign-Editor in die Webseite eingebunden ist.

2.5 Bereitstellungsprozess

Beschreibung, wie der Freedesign-Editor veröffentlicht wird.

2.5.1 Bereitstellungszyklus

Beschreibung des Release-Zyklus.

2.5.2 Bereitstellungsablauf

Beschreibung der Build-Pipeline.

3 Ergebnisse

3.1 Ist-Architektur

3.1.1 Analyse Ist-Architektur

Analyse der Ist-Architektur

- Hierarchisierung
- Paketstruktur und Zugriffe
- Komponenten-Strukturierung
- etc.

3.1.2 Bewertung Ist-Architektur

Kritische Bewertung der Ist-Architektur auf Schwächen.

- Einhaltung von SOLID
- Design-Smells
- Zyklen
- etc.

3.2 Soll-Architektur

3.2.1 Entwurf einer optimierten Soll-Architektur

Entwurf einer Soll-Architektur, die die Schwächen der Ist-Architektur ausmerzt.

3.2.2 Prototyp

Implementation der Soll-Architektur in einem Prototypen zur Bestätigung der Machbarkeit.

3.3 Migrationsstrategie

3.3.1 Migrationsschritte

Ausarbeitung von Migrationsschritten die zum Erreichen der Soll-Architektur notwendig sind.

3.3.2 Priorisierung

Priorisierung der Migrationsschritte.

3.3.3 Migrationsplan

Erstellung eines Ablaufplans zu Migration.

3.3.4 Vorgehensweise für das Refactoring

Beschreibung einer Vorgehensweise wie das Refactoring für die einzelnen Migrationsschritte vorgenommen werden sollte. => Test-Abdeckung, Separieren von Quelltext, etc.

4 Diskussion

5 Zusammenfassung

6 Ausblick

Quellenverzeichnis

Osherove, Roy (2015). *The Art of Unit-Testing - 2. Auflage, deutsch*. Heidelberg: MITP-Verlags GmbH & Co. KG. ISBN: 978-3-826-68721-1.

Springer, Sebastian (2015). *Testgetriebene Entwicklung mit JavaScript*. Heidelberg: Dpunkt. ISBN: 978-3-86490-207-9.