Friedrich-Alexander-Universität Erlangen-Nürnberg
Technische Fakultät, Department Informatik

ROBERT OBERMEIER
BACHELOR-THESIS

WEB APP INTERNATIONALIZATION

Eingereicht am 6. März 2018

Betreuer: M. Sc. Andreas Kaufmann, Prof. Dr. Dirk Riehle
Professur für Open-Source-Software
Department Informatik, Technische Fakultät
Friedrich-Alexander-Universität Erlangen-Nürnberg

Versicherung

Ich versichere, dass ich die Arbeit ohne fremde Hilfe und ohne Benutzung anderer als der angegebenen Quellen angefertigt habe und dass die Arbeit in gleicher oder ähnlicher Form noch keiner anderen Prüfungsbehörde vorgelegen hat und von dieser als Teil einer Prüfungsleistung angenommen wurde. Alle Ausführungen, die wörtlich oder sinngemäß übernommen wurden, sind als solche gekennzeichnet.

Erlangen, 6. März 2018

License

Erlangen, 6. März 2018

Abstract

To be successfull with a web application it's vital to reach users in a variety of languages and region specific formating. This is due to the world wide globalization of markets and especially the global design of the web in general. Internationalizing an application is a difficult task, especially in an environment with short release cycles and continous changes in user interfaces, like in web applications. There are a bunch of libraries and tools to support web developers with internationalization of new solutions. Integrating those in an existing project however is everything else than a trivial task. This can be due to limitations in the existing system or in libraries and tooling that are available to the developer. This thesis describes these problems and integrates a possible solution to these into an existing web application for qualitative data analysis called QDAcity. Additonally it expands on common tooling like Webpack and Gulp to create a solution to localize a modern web application.

Zusammenfassung

Um im Web erfolgreich zu sein ist es unverzichtbar Rücksicht auf Benutzer von einer Reihe von Sprachen und Regionen zu nehmen. Dies liegt vor Allem an der durch Globalisierung erschließbaren Kundschaft außerhalb des Ursprungsmarktes und dem allgmein globalen Charakter des Internets. Die Internationalisierung einer Anwendung ist eine umfangreiche Aufgabe, besonders in Umgebungen, die kurze Entwicklungszyklen und stetige Änderungen an der Nutzerschnittstelle besitzen, wie z.B Webapplikationen. Es gibt bereits eine umfangreiche Anzahl an Bibliotheken und Werkzeugen, welche bei der Entwicklung neuer Lösungen diese im Bereich Internationalisierung unterstützen. Die Integration jedoch solcher Lösungen in bereits bestehende Systeme erweist sich meist schwieriger als gedacht. Dies kann an Einschränkungen des existenten Systems, der Werkzeuge oder Biblotheken, die dem Entwickler zur Verfügung stehen liegen. Diese Arbeit beschreibt diese Probleme und integriert eine Lösung dieser in die bereits vorhandene Webanwendung für qualitative Datenanalyse QDAcity. Zusätzlich werden Erweiterungen bereits vorhandener Werkzeuge wie Webpack und Gulp entwickelt um die Lokalisierung einer modernen Webanwendung zu unterstützen.

Inhaltsverzeichnis

1 Einleitung

1.1 Lokalisierung

Im Bereich der Anwendungsentwicklung besonders bei Benutzerschnittstellen ist eine Interaktion zwischen Anwendung und Benutzer des Systems notwendig. Der Erfolg solcher Anwendungen hängt stark von der Funktion der Nutzerschnitstelle für den Kunden, sowie der Ästhetik dieser aus der Sicht des Kunden, ab. Es ist daher wichtig Schnittstellen zwischen Benutzer und der Anwendung möglichst effizient zu entwerfen und für den Benutzer einfach verständlich zu gestalten. Dazu ist es notwendig die Zielgruppe für ein Produkt zu analysieren und die daraus gewonnenen Unterschiede für Nutzergruppen aus verschiedenen Kulturen und Nationalitäten im Design zu berücksichtigen.

1.1.1 Terminologie

Einer dieser Unterschiede ist die Sprache und Format für Datum, Zeit, Währungs und Zahlenangaben. All diese Eigenschaften werden im Begriff Lokalisierung zusammengefasst. Dies folgt der Terminologie nach Fry und Lommel, 2003, welche hier kurz eingeführt wird. Lokalisierung macht also ein Produkt in einer Region / Kultur verfügbar und somit in dieser benutzbar.

Dazu sind jedoch Modifikationen an bereits vorhandenen System notwendig. Das Sicherstellen, dass ein Produkt auf technischen Wegen lokalisiert werden kann. Sowie die Anpassungen am Design der Anwendung werden unter dem Begriff Internationalisierung zusammengefasst.

Die regionsabhängigen Unterschiede, die selbst bei derselben Sprache auftreten können, wie z.b bei Großbritanien und den Vereinigten Staaten, werden unter dem Begriff Locale erfasst und die tatsächliche Übersetzung der Anwendung analog dazu Translation.

Der Sammelbegriff aller Prozesse eine Anwendung zu lokalisieren wird Globalisierung genannt. Sie umfasst alle notwendigen technischen, ökonomischen, personellen, Marketing und Unternehmensentscheidungen, welche für die Lokalisierung

getroffen und umgesetzt werden müssen. Sie erschließt also alle erforderlichen Probleme, die für den Erschluss des internationalen Marktes erforderlich sind.

1.1.2 Notwendigkeit der Lokalisierung

Betrachtet man das Angebot an Webanwendungen, so sind viele primär oder ausschließlich in englischer Sprache verfügbar. Dies folgt dem Trugschluss, dass Englisch, als globale Sprache, von vielen oder allen Nutzern gesprochen werden muss. Dies ist jedoch nicht der Fall. Nur 8–10% der Weltbevölkerung sprechen English als Muttersprache (Aykin, 2016, S. 4). Selbst unter den mehrsprachigen Nutzern spricht nur einer aus vier ausreichend gut Englisch und über 90% bevorzugen eine andere Sprache als Englisch als primäre Sprache im Privat- und Geschäftsleben (Fry & Lommel, 2003, S. 12). Für fremdsprachige Nutzer ist es also notwendig das Angebot in ihrer eigenen Muttersprache anzubieten und zusätzlich kann das Angebot der eigenen Muttersprache die Qualität der Nutzungserfahrung für multilinguale Nutzer steigern. Neben der Sprache ergeben sich auch weitere regionale Unterschiede, welche Nutzer unter Umständen verwirren können. Dazu zählen unter anderem unterschiedliche Datumsformate, wie z.b das in den Vereinigten Staaten übliche Format, welches in der Reihenfolge Monat, Tag und schließlich Jahr im Gegensatz zum typischen Format in Deutschland, welches zuerst Tag und anschließend Monat verwendet. Solche kleinen Unterschiede können dem Nutzer unbekannt sein und zu Problemen in der Verwendung der Anwendung führen.

1.1.3 Probleme und optimale Vorgehensweise

Nach Aykin, 2016, S. 15, Fry und Lommel, 2003 und Vora, 2009 lassen sich eine Reihe an best Practices und Anforderungen an die Internationalisierung einer Anwendung ableiten.

So ist es notwendig den Zeichensatz der zu lokalisierenden Sprache zu unterstützen. Dazu sind die Schriftart und der Zeichensatz auf Kompatibilität zu prüfen. Eine Lösung bei der Wahl des Zeichensatzes ist der Einsatz eines Unicode-Zeichensates. Diese sind per Design auf die Unterstützung möglichst vieler Zeichen verschiedener Sprachen optimiert. Jedoch ist zu berücksichtigen, dass beispielsweise bei UTF-8 nicht Lateinischen Schriftzeichen mehr Speicherbedarf benötigen und somit ein evtl. höherer Payload beim Ausliefern der Sprache berücksichtigt werden muss.

Zusätzlich wird empfohlen Sprache, wo möglich, von Konfiguration und Layout zu trennen und zu lokalisierende Resourcen zu isolieren. Dazu zählt auch die Bedeutung von Farben und Symbolen in den entsprechenden Regionen. Es müssen

diese auf regionsabhängige Unterschiede geprüft werden, wie z.b. die Farbe Rot in China für Glück steht, jedoch in allen anderen Teilen der Welt auf eine Gefahr hindeutet.

Um Resourcen sowie Kosten zu sparen empfiehlt es sich außerdem auf Text der in Grafiken eingebettet ist zu verzichten. Es ist nicht notwendig auf Text in Grafiken zu verzichten, jedoch sollte dieser nicht Teil der Grafik sondern beim Anzeigen auf diese erst aufgetragen werden. Dadurch verhindert man die Notwendigkeit Grafiken für jede Sprache neu zu entwerfen oder entwerfen zu lassen, was mit hohen Kosten verbunden sein kann.

Es ist beim Entwurf der Anwendung auch auf den sprachabhängigen Platzbedarf zu achten. So kann es bei einzelnen Sprachen zu erhöhtem Platzverbrauch der Textausschnitte kommen, zb. im Deutschen oder zu nur sehr geringem Platzbedarf wie bei den Kurzzeichen der chinesischen Schrift, welche wiederrum eine größere Darstellung benötigen können um lesbar zu sein.

Desweiteren ist das klare Kennzeichnen der zu übersetzenden Anteile einer Anwendung wichtig. Dies verhindert, dass Textabschnitte beim Übersetzen übersehen oder Teile, die nicht übersetzt werden sollen ausversehen übersetzt werden. Es eignet sich hier eine Abstraktion zu schaffen, welche es erlaubt die Texte aus dem Quellcode herauszulösen und somit einer Vermischung verschiedener Interessen zu verhindern. Es ist nicht nützlich für jede Sprache eine eigene Kopie des Anwendungcodes zu besitzen.

Es sollte verhindert werden Zeichenketten im Quelltext aufzuteilen und dort zu verbinden. Dadurch könnte eine Übersetzung gänzlich verhindert werden, da die Satzteile in unterschiedlichen Sprachen auch unterschiedlichen grammatikalischen Regeln folgen müssen.

Die Pluralisierung sollte von der Internationalisierungslösung übernommen werden.

Es ist Notwendig den Kontext in die Übersetzungsdateien zu transferieren. Um eine korrekte Übersetzung zu ermöglichen muss dem Übersetzer klar sein in welchem Umfeld ein Wort verwendet wird.

Neben diesen technischen Problemen, welche bei der Internationalisierung beachtet werden müssen, sind auch bei der Wahl der Sprache, hier Wortwahl und Satzbau, der Anwendung auf Übersetzbarkeit zu achten. Dabei ist nach GDSG Contributors, 2005 auf die folgenden Probleme zu achten:

- Vermeiden von Slang oder Jargon.
- Möglichst keine mehrdeutige Begriffe verwenden.
- Keine Synonyme verwenden und Begriffe konsistent verwenden.

- Dasselbe Wort nicht für verschiedene Bedeutungen verwenden.
- Auf korrekte Rechtschreibung achten.
- Auf konsistente Verwendung von Groß- und Kleinschreibung achten.
- Vermeiden von Ausdrücken, zb. Adjektiven, die nicht eindeutig sind.

In dieser Arbeit wird eine Lösungen für diese Probleme in der Übersetzung von Webanwendungen am Beispiel der Webapplikation QDAcity behandelt.

1.2 QDAcity

QDAcity ist eine Software zur computergestützten qualitativen Datenanalyse. Die qualitative Datenanalyse findet Anwendung in einer Reihe von wissenschaftlichen Disziplinen, wie der wissenschaftlichen Ethnologie, Bildungs- und Sozialwissenschaften, Linguistik, Marktforschung sowie Psychologie. Sie dient im Gegensatz zur quantitativen Datenanalyse als Analyse unstrukturierter Daten, wie Interviewtranskripte, die komplexe Sachverhalte unabhänig vom Betrachter erkennbar machen soll und dabei wesentlichen wissenschaftlichen Ansprüchen genügen muss. Die vielen verschiedenen Zweige qualitativer Forschung und deren Forschungsinteressen lassen sich nach Tesch, 1991 durch drei Gemeinsamkeiten beschreiben. Sie sind sprach-orientiert, beschreibend / interpretativ und theoriebildend. Sprachorientiert, weil sie sich mit der Bedeutung von Worten, wie Menschen kommunizieren, dem Nutzen der Sprache und dem verstehen dieser Interaktionen beschäftigt. Deskriptiv, weil sie soziale Phänomene beschreibt und interpretiert. Theoriebildend, da sie versucht Verbindungen zwischen diesen Phänomenen herzustellen, sprich wie Ereignisse durch Aktoren und deren Verständnis der Sache beinflusst und durch diese definiert wird.

Qualitative Datenanalyse beschäftigt sich also mit dem Was und Warum, den Bedeutungen der Daten, wohingegen quantitative Datenanalyse sich auf Zahlen mit Hilfe von statistischen und mathematischen Mitteln konzentriert.

Die Konzeptualisierung dieser Daten lässt sich unabhängig der verschiedenen Herangehensweisen nach Dey, 2003 in die Ausformulierung der Konzepte und der Analyse der Beziehungen dieser unterteilen. Das Ausformulieren der Konzepte wird durch Beschreibung und Klassifizierung der Daten ermöglicht. Zur unterstützung bei der Analyse von Texten bietet QDAcity Methoden, welche das Annotieren und Kategorisieren von Textabschnitten ermöglicht und somit eine Klassifizierung der Daten erlaubt.

Bei der Klassifizierung werden die Charakteristiken eines Textabschnittes bewertet und diesem Kategorien zugewiesen. Diese Kategorisierung ist bestimmt

durch das Forschungsziel. Jedoch handelt es sich dabei auch um einen iterativen Prozess, so kann die Klassifizierung neue Klarheit über konzeptuelle Zusammenhänge schaffen, welche wiederrum eine Erweiterung der verwendeten Kategorien erlaubt.

In QDAcity wird eine Kategorisierung durch das Codesystem ermöglicht. Ein Codesystem umfasst eine Menge an Codes und beschreibt deren Beziehungen zueinander. Der Begriff Code wird hier für die Kategorien verwendet. Es war in den Sozialwissenschaften üblich aus Effiziensgründen das händische Ausschreiben der Kategorien oder Bennenen dieser mit langen Zeichenketten zu vermeiden. Daher wurden kurze Zahlenkodes verwendet um Textabschnitte damit zu markieren. Dieser Vorgang wird Coding genannt. In einem digitalen System muss ein Code jedoch nicht für jeden Vorgang an die Textstelle geschrieben werden sondern kann wie in QDAcity einfach durch Auswahl auf einen Textabschnitt angebracht werden.

Neben dem Verwalten von Texten und dem Kodieren dieser erlaubt das System die Beschreibung von Zusammenhängen zwischen einzelnen Codes. So können wie bei UML-Diagrammen Beziehungen zwischen Codes abgebildet werden und ermöglichen somit das Darstellen komplexerer Zusammenhänge und die Erweiterung des bestehenden Codesystems.

Zur collaborativen Arbeit erlaubt QDAcity zum einen die Verifikation der Ergebnisse durch Recoding eines Projektes durch Mitarbeiter des Forschungsprojektes, welche über das Interface Zugang zu einem Projekt erhalten können und zum anderen können mehrere Benutzer verteilt an der Kodierung eines Projektes arbeiten.

Die erneute Kodierung durch dritte kann im Anschluss mit Hilfe von statistischer Analyse auf Zustimmung geprüft werden. Dazu werden die verschiedenen Kodierungen miteinander verglichen und Übereinstimmungen ermittelt. Dadurch können Probleme an der Beschreibung des Codesystems erkannt und verbessert werden.

Um den Fortschritt eines Projektes zu dokumentieren können Revisionen eines Projektes erstellt werden. Diese Schnappschüsse können dann auch untereinander verglichen werden um Änderungen nachvollziehbar zu machen.

Im Gegensatz zu vielen anderen Lösungen der qualitative Datenanalyse wird zur Verwendung ausschließlich ein moderner Browser benötigt. Um die Reichweite zu erweitern wurde mit dieser Arbeit grundlegende Anpassungen geschaffen, um QDAcity lokalisierbar und damit in anderen Sprachen verfügbar zu machen.

2 Ziel der Arbeit

2.1 Teilkomponenten

Für die Globalisierung werden mehrere Modifikationen an QDAcity und dessen Buildsystem vorgenommen.

- Internationalisierung der Software
- Erweiterung der Benutzerschnittstelle
- Erweiterung des Buildsystems

QDAcity wird um eine Benutzerschnittstelle zur manuellen Wahl der Region bzw. Sprache erweitert. Diese erlaubt den Benutzern anschließend eine präferierte Anzeigesprache und -format zu wählen. Die erforderlichen Daten für die deutsche Sprache werden angefertigt. Die Software wird um eine interne Schnittstelle zur Lokalisierung erweitert. Das Buildsystem wird um eine Komponente zur Verifikation der Sprachdateien und zum Erzeugung der primären Sprachdatei erweitert.

Diese Einzelkomponenten werden weiterhin als Übersetzungsunterstützung zusammengefasst.

2.2 Anforderungen

Im folgenden sollen die zu erfüllenden Anforderungen an die Übersetzungsunterstützung gestellt werden. Die Anforderungen werden in funktionale, welche konkreten Funktionen zu implementieren sind, bzw. das System bereitstellen soll, und nicht funktionale, wie und mit welchen Eigenschaften diese Funktionen zur Verfügung gestellt werden, unterteilt. Die Erfüllung einer Anforderung muss durch ein definiertes Verfahren verifizierbar sein. Zum Beispiel kann eine Anforderung an die Laufzeit durch messen dieser und festlegen eines Schwellwertes in der Anforderung nach Implementierung auf Einhaltung geprüft werden.

Die Qualitätsmerkmale werden anschließend durch Zuordnung von Attributen in Kategorien nach *System and software quality models*, 2011 klassifiziert.

2.2.1 Funktionale Anforderungen

Integration in vorhandene Webanwendung

Die Übersetzungsunterstützung muss in eine beliebige Anwendung integriert werden. Die Anpassung der Komponenten der Webapplikation sollen durchgeführt und dokumentiert werden. Der Dokumentation müssen mögliche Anforderungen oder Limitierungen an, bzw. bei der Entwicklung neuer Komponenten entnommen werden können.

Definierbarkeit der zu übersetzenden Zeichenketten

Zur Entwicklung einer mehrsprachigen Anwendung benötigt der Entwickler Methoden zur Deklaration von Bezeichnern oder Beschreibungen, welche von den Übersetzern in die jeweiligen unterstützten Sprachen übergeführt werden sollen. Diese Zeichenketten müssen in allen Teilen der Webanwendung verwendet werden können.

Extraktion der Standardsprache

Es muss in der Buildumgebung möglich sein automatisiert alle im Programmquelltext definierten Bezeichner und deren Werte zu extrahieren. Diese sollen zu einem Template zusammengeführt werden, welches den Übersetzern als Startpunkt für neue Sprachen und als Referenz der Standardsprache dient.

Entwicklung eines Key-Value-Dateiformates

Da die Voraussetzungen an Softwarelokalisierer einen notwendigen Programmierhintergrund nicht erzwingen sollen muss ein Dateiformat für die Übersetzungen definiert werden, welches die Wartung und Erstellung durch einen technischen Laien erlaubt. Die Anzahl und Komplexität der Regeln der Grammatik soll gering gehalten und das Format keine bis wenige Escaping-Sequenzen beinhalten.

Verifikation der Sprachdateien

Die Sprachdateien sollen in einem Verifikationsschritt auf Vollständigkeit und syntaktische Korrektheit überprüft werden. Vollständig ist eine Sprachdatei dann, wenn sie alle zu übersetzenden Zeichenketten beinhaltet.

Übersetzung der Anwendung

Die Anwendung soll aus dem Englischen in die deutsche Sprache übersetzt werden.

Automatische Erkennung der Benutzerregion

Einem Besucher der Webanwendung soll beim ersten Aufruf bereits eine vom Standard abweichende Sprache angezeigt werden können. Dazu soll die Region oder Sprache eines Benutzers automatisch ermittelt und ausgewählt werden.

Auswahl der Anzeigesprache

Dem Besucher sowie registrierten Benutzern der Webapplikation soll die Möglichkeit geboten werden selbständig eine Sprache auszuwählen. Dies bedeuted es muss möglich sein die automatische Erkennung der Benutzersprache zu überschreiben und diese manuelle Auswahl muss persistent für diesen Benutzer sein.

2.2.2 Nicht-funktionale Anforderungen

Für die Charakterisierung der Qualitätsmerkmale der nicht funktionalen Anforderungen wird das Produktqualitätsmodel nach ISO/IEC 25010, *System and software quality models* (2011) verwendet. Dieses Modell unterteilt Produktqualität in acht Kategorien:

1. Funktionale Angemessenheit
2. Effektivität des Leistungsverhaltens
3. Kompatibilität
4. Benutzbarkeit
5. Zuverlässigkeit
6. Sicherheit

7. Wartbarkeit
8. Portabilität

Eine Bewertung dieser ist im Standard nicht vorgesehen, da diese stark vom Umfeld und Art des Projektes abhängen. Die Sicherheit eines Produktes ist im Bereich des Bankings beispielsweise höher priorisiert als bei einem Framework zur Lokalisierung von Webanwendungen. Es wurden die Kategorien 3, 4, 5, 7, und 8 als wichtige Anforderungen für die Evaluation gewählt, die Kategorie Sicherheit für diese Umgebung als nicht Zutreffend eingestuft worden. Die Wahl einer Sprache ist jedem gestattet und bedarf keiner Einschränkungen. Außerdem liegt die Sicherheit und Integrität des Quellcodes und der Übersetzungen des Projektes ausserhalb des Bereiches dieser Arbeit. Die nicht-funktionalen Anforderungen werden in den jeweiligen Kategorien aufgelistet.

Funktionale Angemessenheit

- Zur Übersetzung sollen Vorlagen der Orginalsprache existieren und
- neue Zeichenketten müssen dem System einfach hinzugefügt werden können.

Effektivität d. Leistungsverhaltens

- Der Entwicklungsprozess im Watch-Modus soll durch die zusätzlichen Erweiterungen der Buildumgebung nicht merklich verzögert werden. Im Watch-Modus darf die Zunahme der Latenz von durchgeführter Änderung bis zur Ausgabe der Assets 200ms nicht überschreiten.
- Der Übersetzungsvorgang (Transpilation) soll nicht mehr als 10% verlangsamt werden und
- das Aufrufen der Webseite und der Ladevorgang darf nicht angehalten oder mehr als 1 Sekunde verzögert werden.

Kompatibilität

- Die Lösung muss sich in das vorhandene Buildenvironment einpflegen lassen.

Benutzbarkeit

- Die Auswahl der Sprache soll durch maximal 3-4 Klicks möglich sein.

- Das System soll Bezeichner auf das gültige Format hin überprüfen und somit den Entwickler vor typischen Anwendungsfehlern warnen.
- Das Erlernen der Verwendung der Übersetzungsunterstützung soll durch eine Anleitung unterstützt werden.

Zuverlässigkeit

- Im Fehlerfall oder bei ungültigen Eingaben soll der Entwickler aussagekräftige Fehlermeldung mit Ursprung des Fehlers erhalten.
- Das System soll teilweise unvollständige Übersetzungen erlauben, jedoch fehlende Übersetzungen der einzelnen Sprachdateien auszeichnen, bzw. erkenntlich machen.

Wartbarkeit

- Die Ordnungsgemäße Funktionalität des Systems soll überprüfbar sein (Tests).
- Modifikationen an einzelnen Modulen müssen möglich sein.
- Die Module sind wiederverwendbar

Portabilität

- Die Übersetzungsunterstützung muss einfach austauschbar sein und nur lose an das System gekoppelt sein.

3 Architektur und Design

3.1 Bestehendes System

3.1.1 QDAcity

QDAcity besteht aus zwei Bereichen, dem Front- und Backend. Zusammen bestehen diese Bereiche aus drei unabhängigen Komponenten. Zwei, welche das Frontend der Anwendung bilden, und dem Backend, welches auf einer Erweiterung der Servlets für Java-Webanwendungen basiert. Die Erweiterung wurde für die Cloudumgebung des Unternehmens Google von selbigem entworfen. QDAcity verwendet Google AppEngine als Hostingplatform, welches eines der Produkte der Google Cloud Platform (GCP) darstellt. Das Backend wird vor allem als Schnittstelle zum persistenten Speicher (Google Cloud Datastore) verwendet und dient der Sicherstellung der korrekten Authentifizierung und Authorisierung der Zugriffe auf den Datastore. Zusätzlich erfüllt das Content-delivery-network der GCP die typischen Aufgaben eines Backends einer Webanwendung, wie das Bereitstellen der statischen Ressourcen. Zu diesen Zählen der Code des Frontends, Grafiken, Schriftarten und weitere externe Resourcen. Die Funktionalität des Backends wird als REST-Schnittstelle auf der Google Cloud Platform veröffentlicht. Das Frontend ist eine auf HTML5 und Javascript basierende Webanwendung. Die Anwendung wird zum Sprachstandard ECMAScript 6 übersetzt um eine möglichst breite Abdeckung an unterstützten Browsern zu unterstützen.

Das Frontend verwendet eine Reihe von Technologien um die Entwicklung zu erleichtern. Der genauere Aufbau und Zweck einiger Komponenten wird in diesem Kapitel behandelt.

Die hier behandelten Komponenten sind ausschließlich entwicklungsspezifische Abhängigkeiten, welche im laufenden Betrieb nicht verwendet werden. Dazu zählen ausschließlich Komponenten der Transpiler-, bzw. Buildumgebung, wie Gulp, Webpack und Babel.

Weitere Abhänigkeiten, die zum Zeichnen von Diagrammen oder zur Berechnung

von Statistischen Funktionen verwendet werden, werden hier außer Acht gelassen, zumal sie keinen direkten Einfluss auf das Userinterfacedesign haben.

3.1.2 React

Die Anwendung basiert auf dem Webframework React, welches ein Komponenten gestütztes Framework ist. Einzelne Komponenten sind in sich abgeschlossene Einheiten, welche jeweils eine Funktionalität der Anwendung implementieren. Diese React-Komponenten können ineinander geschachtelt werden und eine Einstiegskomponente dient als Wurzel des daraus entstehenden Baumes. React erweitert Javascript um deklarativen Syntax, welcher XML, bzw. HTML ähnelt und erlaubt damit auch direkt HTML in den Komponenten zu verwenden. Dadurch wird es möglich alle Teile einer Komponente einfach zusammenzuführen und es ist nicht notwendig Logik und Darstellung streng voneinander zu trennen. Ausserdem vereinfacht es damit die Einbindung und Manipulation der Elemente des DOMs der Anwendung.

Neben den syntaktischen Erweiterung im Umgang mit dem DOM wird dieser auch um Properties und Kontext erweitert, welcher die Darstellung oder Zustand der Anwendung beinflussen können. Es wird durch einen Zustandsvergleich der inneren Darstellung des DOMs, meist unter dem Begriff VirtualDOM[1] referenziert, sichergestellt, dass nur in sich geänderten Komponenten des Browser-DOMs Aktualisierungen veranlasst werden.[2]

3.1.3 Gulp

Die Aufgaben der Buildumgebung für QDAcity werden durch Gulp realisiert. Gulp ist ein task-driven (aufgaben-basiertes) Toolkit, welches die Definition der Buildschritte durch einzelne Aufgaben gliedert, welche Abhängigkeiten zu anderen Aufgaben besitzen können. Dabei werden zunächst erst alle Abhängigkeiten parallel ausgeführt und nach deren Abschluss der eigentliche Task ausgelöst. Es bietet also die typischen Funktionalitäten einer Buildumgebung.

Innerhalb der Tasks ist der Workflow typischerweise in drei Teile eingeteilt. Sammeln der Source Artifakte, verarbeiten dieser und erzeugen der Build Artifakte. Dabei werden diese einzelnen Abschnitte in einer Pipeline hintereinander geschal-

[1] VirtualDOM ist hier als eine zusammenfassende Beschreibung der verwendeten Technologie zu verstehen, sprich die innere Darstellungsform des Anwendungszustandes.

[2] Auf eine genauere Beschreibung der inneren Architektur wird hier explizit verzichtet, da sich diese mit dem kürzlichen Release von Version 16 vollständig verändert hat und damit bereits nicht mehr zutreffend wäre.

tet und die einzelnen Artifakte wahlweise als zusammenhängender Strom oder als einzelne Dateien schrittweise durch diese Pipeline gereicht.

Im Verarbeitungsschritt können mehrere Einzelschritte hintereinander gereiht werden und dadurch ist es möglich diese in verschiedenen Aufgaben wiederzuverwenden.

3.1.4 Webpack

Für das zusammenfassen aller Quelldatein und das erstellen eines vom Browser ausführbaren ECMAScript Artifakts wird der Bundler Webpack verwendet. Die Aufgaben eines Bundlers bestehen u.a. darin viele Einzelkomponenten einer Webanwendung in Chunks zusammenzufassen um die Anzahl der notwendigen Anfragen und damit Latenz- und Ladezeiten beim Laden einer Webanwendung zu reduzieren. Viele Bundler bieten darüber hinaus eine Reihe an weiteren Funktionen und Erweiterungen.

Webpacks Architektur erlaubt zwei Eintrittsstellen für Modifikationen. Eine Erweiterung ist durch sogenannte Loader möglich. Loader werden in Webpack verwendet um beliebige Assets in Javascriptmodule umzuwandeln. Loader können in Reihe geschalten werden, jedoch muss das Resultat am Ende ein Javscriptmodule sein. Dies kann zum Beispiel verwendet werden um beliebige Dateien mit in das Bundle einzubinden. Es muss dafür eben ein Loader angefertigt werden, welcher diese Datei in gültigen Javascript Code umwandelt. Eine dieser Erweiterungen ist die verwendung eines Transpilers zur Transformation von Quelldateien für eine Zielumgebung. Eine weitere Schnittstelle sind Events, welche durch Plugins genutzt werden können. Dazu werden für vordefinierte Einhängepunkte Funktionen registriert, welche Modifikationen am Zustand und der zu erzeugenden Ausgabeassets vornehmen können.

3.1.5 Babel

Als Transpiler für QDAcity dient Babel. Babel ist ein modularer Compiler für Javascript. Babel bietet eine Programmierschnittstelle, die es erlaubt das Verhalten des Compilers zu erweitern oder zu verändern. Primärer Einsatzzweck ist Erweiterungen der Sprache Javascript in Kode überzuführen, der von Interpretern älterer ECMAScript Versionen ausgeführt werden kann. Auch nicht standardisierte Erweiterungen wie der von React verwendete JSX Syntax ist möglich.

Babel durchläuft beim Übersetzen die für Kompiler typischen Phasen: Parsen, Transformation und Erzeugung. Beim Parsen werden die Quellen in zunächst in

Tokens zerlegt. Tokens sind die Bausteine einer Grammatik. Gemäß der Grammatik Regeln werden diese Token dann in der syntaktischen Analyse zu einem Syntaxbaum (AST) verbunden. In der Transformationsphase werden, je nach aktivierten Plugins, Modifikationen am AST vorgenommen. Der daraus Resultierende Syntaxbaum wird dann in der Generationsphase wieder in Kode übergeführt.

3.2 Design

Im folgenden Abschnitt wird der finale Entwurf für die Übersetzungsunterstützung dargestellt. Für die Lokalisierung wird React-Intl in die Anwendung integriert und eine Komponente zur Auswahl der Sprache/Region erstellt. Desweiteren wird eine Sprach-Dateiformat entworfen und die Buildumgebung erweitert, so dass die Primärsprache automatisch extrahiert werden kann.

3.2.1 Evaluation möglicher Lokalisierungslibraries

Bibliothek	Lizenz	Entwickler
Fluent.js	Apache 2.0	Project Fluent
L20n	Apache 2.0	Mozilla
Format.js	BSD-3	Yahoo
Polyglot	BSD-2	Airbnb
Globalize	MIT	jQuery
jQuery.i18n	GPL2	Wikimedia

Tabelle 3.1: Liste bekannter Globalisierungsbibliotheken

Es wurden aus einer Liste, die der Tabelle 3.1 zu entnehmen ist, von Globalisierungslösungen drei mögliche Kandidaten ausgewählt und für die Verwendung in QDAcity näher evaluiert. Neben den in Tabelle 3.2 bewerteten Bibliotheken wurden in einer Vorauswahl alle Libraries ausgeschlossen, welche keine direkte oder indirekte Unterstützung für React liefern. Eine eigene Implementierung wurde Aufgrund vorhandener Lösungen nicht in Betracht gezogen. Für die Verwendung der Bibliotheken mussten einige Kriterien erfüllt werden. Harte Kriterien, welche unbedingt erfüllt werden mussten, waren hier das Lizenzmodell der Libraries und Support für den deklarativen Syntax der React-Komponenten. Diese wurden von allen in Tabelle 3.2 aufgelisteten Bibliotheken erfüllt. Polyglot jedoch würde eine weitere externe Lösung benötigen und besitzt selbst keine direkte Anbindung an das React Framework. Weitere Kriterien zur Auswahl eines möglichen Kandidaten sind der Umfang der Library, Funktionsumfang, sowie das Format der

Kriterium	Bibliothek		
	Fluent.js	Format.js	Polyglot
Lizenzmodell	Apache 2.0	BSD 3-Clause	BSD 2-Clause
Open-Source	ja	ja	ja
Syntax	imperativ, deklarativ	imperativ, deklarativ	imperativ
Geschätzter Speicherbedarf	ca. 80kb	ca. 40kb	ca. 3kb
React support	ja	ja	nein, separate Lösung
Dokumentation	vorhanden, exemplarisch	vorhanden, exemplarisch	vorhanden, exemplarisch
Messageformat	eigenes Format (Project Fluent)	ICU-Messages	eigenes Format
Pluralisierung	ja	ja	ja
Platzhalter	ja	ja	ja
Extraktion	nein, geplant	ja, mit Einschränkungen	ja, extern

Tabelle 3.2: Vergleich von Globalisierungsbibliotheken

Zeichenketten. Um eine möglichst hohe Wartbarkeit der Lösung zu erreichen wurde das ICU-Message Format zu eigenen Lösungen der Libraries bevorzugt. Das ICU-Messageformat findet seit Jahren in vielen Anwendung Verwendung und ist umfangreich dokumentiert und getestet. Eigene Formate verhindern zudem einen späteren Austausch der Lokaliserungslösung oder erschweren diesen, da die Übersetzungen und Verwendung dieser unter Umständen an ein neues Format angepasst werden muss. Die Verwendung eines verbreiteten Formates verringert diese Gefahr.

Neben den erforderlichen Funktionen, wie der Unterstützung mehrsprachiger Pluralisierung oder dem Verwenden von Platzhaltern, welche mit Werten zur Laufzeit gefüllt werden, zählt die programmatische Extraktion von zu übersetzenden Zeichenketten eine wichtige Rolle. Diese wird nur direkt von FormatJS unterstützt.

Ein weiteres Kriterium an die Bibliothek ist ein möglichst kleiner Payload, sprich der Grad der Zunahme an Speicherbedarf an das Bundle, welches zum Laden der Anwendung benötigt wird. Da die Lokalisierung ein grundlegender Bestandteil der initialen Darstellung der Anwendung ist, muss dieser Möglichst klein Gehalten werden, um keine negativen Auswirkungen auf die Usability der Anwendung, wie z.B. Latenz, zu erzeugen. In der hier ermittelten Payloadgröße[3] zeichnet sich

[3]Bundlegröße bei Projekt ohne weitere Abhängigkeiten. Dies ist je nach bereits vorhandenen Abhängigkeiten stark abweichend von den in der Tabelle ermittelten Werten.

Polyglot wegen der fehlenden Einbindung in den React-Kontextes weit von den anderen Bibliotheken ab.

Zusammenfassend lässt sich in der Auswertung der einzelnen Bibliotheken Format.js als bester Kandidat unter Berücksichtigung der hier gewählten Kriterien ernennen.

3.2.2 Sprach-Dateiformat

Listing 3.1: Beispielausschnitt aus einer Sprachdatei

```
# Modaldialog akzeptieren
modal.ok = Ok
# Modaldialog zum Ablehnen einer Aktion
modal.cancel = Abbrechen
multiline = Ein mehrzeiliger Textabschnitt.
        Der in der Sprachdatei auch
        Zeilenumbrueche verwendet.
```

Das Sprach-Dateiformat ist eine durch Zeilenumbrüchen getrennte Auflistung der Identifier und der entsprechenden Übersetzungen in der Zielsprache, welche durch diese Datei definiert werden soll. Zeilen mit einem Whitespace Zeichen am Anfang werden als Fortsetzung der letzten Zeile interpretiert. Dadurch ist es möglich Zeilenumbrüche in Zeichenketten ohne Escaping einzufügen. Das erste Zeichen wird dabei ignoriert und ist nicht Teil der resultierenden Zeichenkette. Beschreibungen für Identifier zur Verdeutlichung des Kontext werden durch das Raute Symbol(#) am Anfang einer Zeile eingeleitet. Diese Zeilen werden jedoch in der weiteren Verarbeitung wie Kommentare behandelt. Gruppierungen sind im aktuellen Entwurf möglich, jedoch vorerst nicht vorgesehen. Als Dateiname wird der Language Code nach ISO 639-1 gewählt.

3.2.3 React-Intl

React-Intl ist eine von Yahoo entwickelte Erweiterung für das React-Framework, welche eine Spezialisierung des Format.js Frameworks ist. Es gibt entsprechende Spezialisierungen von Format.js für andere Komponenten oder Templatesysteme, wie beispielsweise Ember und Handlebars. Sie erlaubt durch das Hinzufügen neuer React-Komponenten das Übersetzen einer Anwendung. Es wird jeweils die imperative und deklarative Syntax um diese Funktion erweitert. Für die deklarative Syntax muss zunächst erst eine Vaterkomponente (IntlProvider) in den Baum der Reactanwendung eingehangen werden. Alle Kindkomponenten erben dann automatisch den Kontext dieser Komponente. Dadurch ist es möglich an

beliebigen Stellen unterhalb der Komponente im Baum das FormattedMessage-Element, welches selbst eine React-Komponente ist, zu verwenden. Diese dient funktionell ähnlich einem Platzhalter, welcher beim Rendervorgang durch die in ihm referenzierte Nachricht ersetzt wird. Nachrichten, bzw. Messageidentifier sind Javascript Objekte und bestehen aus einer eindeutigen ID, einer fallback Zeichenkette (defaultMessage) und optional einer Beschreibung (description).

Die iterative Syntax erfordert eine High-Order-Component, welche in den Kontext der Komponente, welche die iterative Syntax verwenden möchte den React-Intl Kontext injeziert. Eine solche HOC kann mit React-Intls injectIntl Funktion aus einer beliebigen React-Komponente erzeugt werden.

3.2.4 Extraktion der Identifier

Zur Analyse des Quellcodes und der Sammlung aller Messageidentifier wurde für das Auffinden der deklarativ Spezifizierten Identifier eine Erweiterung für Babel verwendet, welche sich über die Pluginschnittstelle in das Buildsystem integriert. Für die iterative Verwendung der React-Intl Erweiterung wurde eine eigenes Plugin für diese Schnittstelle in Babel entwickelt und in das Buildsystem integriert.

3.2.5 Sprachauswahl

Um dem QDAcity-User die richtige Sprache anzubieten, wird die Anwendung um eine automatisch Spracherkennung des Zielsystems erweitert. Sollte eine verfügbare Sprache gefunden werden, der User bisher jedoch keine manuelle Auswahl getroffen haben, so wird diese ausgewählt. Ist eine manuelle Auswahl verfügbar so überschreibt diese die automatische Erkennung. Sollte weder eine Wahl noch eine passende Sprache gefunden werden, wird die primäre Anwendungssprache verwendet.

4 Implementierung

In diesem Kapitel werden die näheren Implementierungsdetails und Designentscheidungen und deren Gründe dargelegt.

4.1 Lokalisierung der Anwendung

4.1.1 Einbinden der Lokalisierungsbibliothek

Die Einbindung der Bibliothek wird in mehrere Arbeitsschritte unterteilt. Zunächst werden Buildsystem und Entwicklungsumgebung an die neuen Abhänigkeiten angepasst, so dass eine erfolgreiche Übersetzung mit Verwendung der erweiterten Syntax möglich ist. Im Folgeschritt werden alle Zeichenketten in eine neue Form übergeführt, welche es erlaubt diese zur Laufzeit auszutauschen. Abschließend werden eine Sprachauswahl, sowie eine Mechanik zum ad-hoc laden dieser in die Anwendung integriert.

4.1.2 Vorbereitende Anpassungen

In die Buildumgebung werden die neuen Abhänigkeiten eingetragen und die initalisierung der Bibliothek wird in das Programm eingepflegt. Da die Bibliothek keine automatische Weitergabe des Kontextes an außerhalb des App-Kontextes gerenderte Komponenten unterstützt, wird der IntlProvider durch eine eigene High-Order-Component (HOC) ersetzt. Die HOC ist eine Spezialisierung des IntlProviders und erweitert diesen um globalen Anwendungszustand, um externen Komponenten einfachen Zugriff auf den Lokalisierungskontext zu geben. QDAcity verwendet externe Komponenten im Bereich der Dialoge, welche zur Laufzeit dynamisch erzeugt und außerhalb des virtuellen DOM Reacts existieren. Diese Komponenten haben dadurch keinen Zugriff auf den Zustand der React-Anwendung und müssen über eine eigene Schnittstelle Zugriff auf diesen Erhalten. Dadurch

dass der Lokalisierungszustand (gewählte Region und Sprache) ein globaler Zustand der Anwendung ist, wird hier ein statisch abrufbares Singleton instanziert, welches den Zustand aus der Anwendung exportiert.

Ersetzen der Zeichenketten

Es werden alle vorhandenen Zeichenketten der Anwendung analysiert und auf die Notwendigkeit der Übersetzung geprüft. Alle dadurch gefundenen Zeichenketten werden durch in Funktionsaufrufen eingefasste Messageidentifier ersetzt. Eine Zeichenkette wird also durch einen eindeutigen Bezeichner identifizierbar gemacht und dann als JSON-Objekt an einen Aufruf der formatMessage Funktion des Intl-Providerobjektes weitergegeben. Dieser kann dann zur Ausführung testen, ob für den Bezeichner eine zu verwendente Übersetzung vorliegt oder auf das Fallback zurückgegriffen werden muss. Der Aufbau des JSON ist in Abschnitt 3.2.3 als Messageidentifier genauer beschrieben.

Ähnlich der Anpassung der Zeichenketten im imperativen Teil der Anwendung, werden alle deklarativen Elemente überprüft und ersetzt. Für die deklarative Syntax werden Messageidentifier als eigene Elemente definiert, welche die Parameter der JSON-Objekte als Attribute dieses Elementes abbilden.

Durch die Notwendigkeit des globalen Zustandes der Lokalisierungschnittstelle, siehe Abschnitt 4.1.2, werden alle imperativ deklarierten Messageidentifier über die externe Schnittstelle verwendet. Dies soll die Komplexität und damit die Fehlerrate bei der Verwendung reduzieren, da eine Lösung verwendet wird, die in externen sowie internen Kontext einer React-Anwendung gültig sind.

4.1.3 Einfügen der Sprachauswahl

Es werden in die HOC Funktionen zum wechseln der Sprache sowie Region integriert. Diese werden bei der Initalisierung der Anwendung mit der Region bzw. Sprache des Browsers aufgerufen. Dadurch werden beim Start der Anwendung die Daten der Usersprache geladen. Beim wechseln der Sprache wird in einer fest definierten Liste die Existenz der Sprache geprüft. Die Liste beinhaltet alle verfügbaren Sprachen und es werden dadurch nur Sprachen geladen, welche zuvor freigegeben wurden. Zum Laden einer Sprache wird ein Request an den Webserver gesendet und die Sprachdatei angefordert. Nach dem erfolgreichen Laden wird der Zustand der Lokaliserungslösung durch Weitergabe der Sprachdatei geändert. Dieser Zustandswechsel verursacht das erneute Rendern alle davon betroffenen React-Komponenten und somit dem Wechsel der Sprache der Anwendung.

Abschließend wird eine Auswahlkomponente in die Anwendung integriert, welche dem Benutzer das manuelle wechseln der Sprache erlaubt. Dazu wird eine

React-Komponente in die Useransicht eingepflegt, welche die Liste der verfügbaren Sprachen anzeigt und dem Benutzer durch anklicken einer Sprache den Wechsel zu dieser Sprache erlaubt. Die Liste der Sprachen wird aus dem Set der verfügbaren Sprachen der Lokalisierungschnittstelle gelesen.

4.2 Extraktion und Erzeugen der Templates

Die in Abschnitt 4.1.2 eingeführten Messageidentifier sollen aus dem Quellcode softwaregestützt gesammelt werden. Dazu wird in die Buildumgebung eine Erweiterung des Javascript Compilers Babel eingepflegt. Diese sucht in Quelldateien nach allen FormattedMessage-Tags des React Syntaxes. Alle gefundenen Identifier inklusive deren Fallback Zeichenketten werden in einem Array zu den Metadaten der inspizierten Datei in Babel hinzugefügt. Für alle der Dateien werden am Ende des jeweiligen Buildvorgangs die Metadaten gesammelt. Dazu wird ein Plugin für Webpacks Babel-Loader entwickelt. Der Babel-Loader führt dann die im Plugin definierten Instruktionen nach dem Laden und Übersetzen einer Quelltextdatei aus. Das Plugin kopiert die Liste an Message-Objekten in eine im Speicher gehaltene Map an Message-Objekten. Die Map ordnert jeweils einer Quelldatei die Message-Objekte zu, so dass beim erneuten Laden eine Quellcodedatei die entsprechenden Identifier aktualisiert werden können. Am Ende des Buildvorgangs erzeugt das Plugin eine Sammlung aller Message-Objekte. Dazu werden alle gesammelten Bezeichner in einem Javscript Objekt vereint. Als Bezeichner des Attributes dienen jeweils die Identifier der Message-Objekte. Entdeckte Duplikate werden auf Gleichheit geprüft und bei Unterschieden ein Fehler ausgegeben. Dies erlaubt die mehrfache Verwendung derselben identifier an verschiedenen Stellen im Quelltext und fasst all diese zu einer zu übersetzenden Zeichenkette zusammen. Dadurch ist eine Deduplikation gleicher Zeichenketten möglich.

Zusätzlich wird aus der Gesamtheit der Message-Objekte eine Vorlage für die in Abschnitt 3.2.2 definierten Sprachdatei erstellt. Dazu verwendet das Plugin einen Transformator, welcher in beiden Richtungen, also die Sprachdatei in das JSON-Format für die Übersetzungsunterstützung des Frontends und das JSON Format in eine Sprachdatei, konvertieren kann. Die erzeugte Sprachdatei und die JSON-Datei werden als Asset in Webpack registriert und von diesem anschließend via Gulp in das Zielverzeichnis geschrieben.

Desweiteren wird ein Plugin für Babel angefertigt, welches die durch die iterative API verwendeten Message-Objekte extrahiert. Das Plugin wird als Transformator in Babel registriert und dadurch in der zweiten Phase des Kompilers ausgeführt. Jedoch werden im Gegensatz zu sonst üblichen Transformationsschritten lediglich analytische Schritte am Abstract-Syntax-Tree (AST) der Anwen-

dung durchgeführt. Eine Modifikation des Quellcodes ist nicht vorgesehen, kann jedoch jederzeit zu der Implementierung hinzugefügt werden. Um alle möglichen Message-Objekte zu finden, wird als Einstiegspunkt ein Visitor für Call-Expressions verwendet. Call-Expressions umfassen alle Funktionsaufrufe ECMA-Scripts bis Version 6. Ausnahme davon sind Tagged-Template-Expressions welche für die Verwendung im Kontext der Übersetzungsunterstützung ausgeschlossen werden. Call-Expressions bestehen aus der aufzurufenden Funktion Callee und den Argumenten des Aufrufs. Das Plugin wertet die Argumente jedes Aufrufs, dessen Callee das durch die Lokalisierungschnittstelle exportierte formatMessage ist, aus und fügt diese der Metadaten des React-Intl Babel Plugins hinzu, so dass diese durch das oben beschriebene Verfahren ebenfalls extrahiert werden können. Zur Erkennung ob ein Callee dem von der Lokaliserungslösung exportierten formatMessage entspricht, werden Hilfsfunktionen definiert, welche den AST daraufhin untersuchen.

4.2.1 Analyse des AST

Der Abstrakte Syntax Baum (AST) des ECMAScript Compilers Babel besteht unter Anderem aus Nodes, Pfaden und deren Scope[1]. Der Pfad (Path) ist der Weg im Baum hin zum aktuellen Knoten (Node). Ein Node beschreibt eine Komponente der Grammatik der Sprache. Scope ist der jeweilige Geltungsbereich eines Bezeichners. Zum Auffinden des Ursrpungs eines Bezeichners muss im aktuellen Geltungsbereich (Scope) nach der Definition des Bezeichners gesucht werden. Ist eine solche Definition gefunden muss diese auf ihren Typ hin untersucht werden und entsprechend diesem das Verfahren zum Auffinden eines Bezeichners wiederholt werden. Beispiel dafür wäre eine Zuweisung, bei der einer Variable X der Wert Y zugeordnet wird. Y kann hier wieder ein Bezeichner sein, welche demnach verfolgt werden müsste. Ist im aktuellen Scope der Bezeichner nicht zu finden, muss je nach Position im Programmkode die Suche im umfassenden Scope fortgesetzt werden. Beispielsweise der umschließende Block dieses Codeblocks. Das oberste Ende der Scopes ist der Programmscope, sprich die aktuelle Datei. Sollte in keinem der Scopes der Bezeichner exisiteren ist er invalide oder nicht gebunden. Globale Variablen zählen zu letzteren, da diese erst zur Ausführung bekannt sind. Neben Bezeichnern müssen auch Member-Expressions aufgelöst werden. Member-Expressions umfassen Arrayzugriffe und Objektzugriffe, sprich Zugriffe auf Properties eines Objekts. Member-Expressions können nicht bei dynamisch erzeugten Objekten aufgelöst werden, da diese erst zur Laufzeit entstehen und somit eine Evaluation des Quellcodes benötigen. Eine weitere Variation der aufzulösenden Konstrukte sind Imports. Imports werden im AST durch Import-Declarations beschrieben. Eine Import-Declaration ist eine import Anweisung im

[1] Weitere Metadaten werden hier nicht berücksichtigt.

Quellcode und kann keine, einen oder mehrere Bezeichner definieren. Zum Auflösen der Identifier, welche eine Import-Declaration definiert, müssen partiell auch Object-Destructors analysiert werden. Dies gilt für import Anweisungen, die aus dem importierten Namensraum Bezeichner importieren.

Für die einzelnen Schritte der zu analysierenden Bestandteile des ASTs werden Funktionen mit dem Ziel angefertigt einen beliebigen Bezeichner hin zu seiner Definition zurückzuverfolgen zu können. Dieser Pfad wird durch eine weitere Hilfsfunktionen auf die einzelnen Bestandteile eines imports der formatMessage Funktion aus der Lokalisierungschnittstelle überprüft und dessen Existenz bejaht oder abgelehnt.

Bei zustimmender Entscheidung wird der gefundene formatMessage Aufruf an eine weitere Funktion gereicht, welcher aus dem Aufruf das Message-Objekt extrahiert. Diese Analyse erfordert einen statischen formatMessage Aufruf.

Alle unerwarteten Zweige im AST werden im Entwicklermodus an den Entwickler ausgegeben. Fehler in der Verwendung oder ein nicht statischer Aufruf der formatMessage Funktion wird dem Entwickler als Fehler angezeigt.

4.3 Konvertieren der Sprachdateien

Die Buildumgebung wird um eine Aufgabe zur gesammelten Transformation der Sprachdateien hin zum Zielformat erweitert. Dazu wird der im Abschnitt 4.2 beschriebene Transformator verwendet und ein Gulp-Task definiert, welcher alle Sprachdateien in ihr Zielformat konvertiert und anschließend ins Zielverzeichnis exportiert. Der Gulp-Task verifiziert die Sprachdateien, indem er sie mit der Vorlage vergleicht und jegliche Abweichungen als Warnungen im Buildprozess dokumentiert.

5 Evaluation

Im folgenden Abschnitt soll die in Abschnitt 3 beschriebene Lösung mit den Anforderungen aus Abschnitt 2.2 evaluiert werden.

5.1 Anforderungen

5.1.1 Funktionale Anforderungen

Integration in vorhandene Webanwendung

Es wurden Erweiterungen an den verwendeten Frameworks durchgeführt und es wurden alle Zeichenketten der Anwendung durch ihr pendant der Übersetzungsunterstützung ausgetauscht. Die Verwendung der Erweiterungen wurde im Projekteigenen Wiki dokumentiert und Beispiele zur korrekten Verwendung angegeben. Im selbigen Wiki wurden außerdem die Restriktionen und Limitierungen des Systems angegeben. Es wurde jeweils eine Schnittstelle für die deklarative und imperative Syntax entworfen. Der bestehende Funktionsumfang wurde in keiner Weise beschränkt.

Definierbarkeit der zu übersetzenden Zeichenketten

Die Anwendung verwendet das durch React-Intl Library implementierte ICU-Messageformat zum definieren der Zeichenketten. Die Message-Objekte können sowohl im deklarativen als auch imperativen Syntax definiert und somit in sämtlichen UI-Komponenten der Andwendung verwendet werden.

Extraktion der Standardsprache

Durch die Verwendung sowie der Implementierung geeigneter Plugins für das Buildsystem werden Message-Objekte im Quelltext erkannt und automatisch ex-

trahiert. Aus den extrahierten Messages wird eine Vorlage erzeugt, welche als Grundlage für Übersetzungen verwendet werden kann.

Entwicklung eines Key-Value-Dateiformates

Das Key-Value Dateiformat und Parser für dieses wurden in die Buildumgebung integriert. Das Dateiformat beinhaltet keine Escaping-Sequenzen.

Verifikation der Sprachdateien

Die Anpassungen der Buildumgebung verifizieren den Inhalt der Sprachdateien auf Vollständigkeit.

Übersetzung der Anwendung

Die Anwendung wurde in die Deutsche Sprache übersetzt und die Sprachdatei in die Versionskontrolle eingepflegt.

Automatische Erkennung der Benutzerregion

Beim Besuchen der Webseite, wird dem Anwender die in seinem Browser voreingestellte Nutzersprache angezeigt, soweit diese in der Anwendung zur Verfügung steht. Ist die Sprache nicht verfügbar oder die Erkennung nicht möglich wird die Primärsprache der Anwendung angezeigt.

Auswahl der Anzeigesprache

Die React-Anwendung wurde um eine Komponente zur Auswahl der Benutzersprache erweitert, welche persistent ist und die automatische Erkennung aussetzt, bzw. überschreibt.

5.1.2 Nicht-funktionale Anforderungen

Funktionale Angemessenheit

Die Sprache kann durch den Benutzer mit drei Klicks geändert werden. Es wurde das Buildsystem um Funktionalität erweitert, welche Vorlagen der Orginalsprache erzeugt. Durch die Einbindung der React-Intl-Library wurde das einfügen neuer

Zeichenketten in beiden Syntax ermöglicht. Diese bestehen aus einem Funktionsaufruf mit einem Javascript-Objekt oder einem XML ähnlichen Syntax. In beiden Fällen muss ausschließlich ein Bezeichner und die Zeichenkette angegeben werden.

Effektivität d. Leistungsverhaltens

Messung	Zeitbedarf	
	Mit Übersetzungsextraktion	Ohne
100 Iterationen (Gesamt)	1357 s	1357 s
100 Iterationen (Einzel)	13.6 s	13.6 s
1 Iteration (S.)	13.1 s	13.3 s
1 Iteration (L.)	14.5 s	14.4 s
Standardabweichung	0.26	0.17

Tabelle 5.1: Vergleich der Laufzeiten des Frontend-Build-Tasks

Die Evaluation der Arbeitszeit ist schwierig, da keinerlei Aufzeichnungen zur Arbeitszeit existieren. Dadurch ist eine Beobachtung des Einflusses auf die Arbeitsleistung bis zum Abschluss der Arbeit nicht möglich gewesen. Zur Verifikation der Effizienz des Übersetzungsvorganges wird die Laufzeit des Prozesses mit und ohne dem aktuellen System gemessen. Dabei werden die Zeiten wie in Tabelle 5.1 ermittelt. Die Ermittlung der Zeiten erfolgt durch Messen des Start- und Endzeitpunktes der Ausführung des Buildprozesses. Der Buildprozess wird dabei in einer Schleife 100 Mal wiederholt und die Gesamtzeit gemessen. Dadurch sollen Einwirkungen auf das Messergebnis wie Caching und Hintergrunddienste des Systems minimiert werden. Es wurden aus den Messungen die Mittelwerte, Randfälle und Varianz bestimmt. Wie man der Tabelle entnehmen kann kam es zu vernachlässigbaren zeitlichen Einschränkungen.

Konfiguration	Zeitbedarf		
	Vor Impl.	Mit Übersetzung	Stubs
CPU-Handicap (6-Fach)	$3,0\ s \pm 200\ ms$	$3,2\ s \pm 200\ ms$	$3,1\ s \pm 200\ ms$
Mobile (Lighthouse)	$28\ s \pm 1\ s$	$35\ s \pm 1\ s$	$34\ s \pm 1\ s$

Tabelle 5.2: Vergleich der Laufzeiten des Frontend-Build-Tasks

Zum Analysieren des Ladeverhaltens der Webseite werden Profilinginformationen des Webbrowsers vor und nach den Modifikationen analysiert, sowie die aktuelle

Implementierung durch Stubs ersetzt. Eine Zusammenfassung ist der Tabelle 5.2 zu entnehmen. Die enormen Abweichungen in den Messungen durch Lighthouse, sind unter anderem dem Netzwerkmodus geschuldet. Diese Messungen zeigen, dass die aktuelle Lösung noch nicht auf Mobile daten optimiert ist. Da die Messung jedoch an der nicht minifizierten Codebasis durchgeführt wurde sind die Messergebnisse weitaus höher als im Produktivsystem anzunehmen. Dadurch dass die Sprachdatei beim initialen Laden der Seite nicht Teil des Requests sondern eine externe Ressource ist, die nach dem Interpretieren des Anwendungscodes nachgeladen werden muss, kommt es hier zu enormen Verzögerungen. Außerhalb der mobilen Umgebung sind die Verzögerungen im Bereich zwischen 100–200 ms, was im Rahmen der Anforderungen liegt.

Kompatibilität

Das System konnte erfolgreich in die Umgebung eingepflegt werden.

Benutzbarkeit

Durch Verifikationschritte und Analysen des AST bei der Extraktion der Bezeichner, anschließender Kollisionsprüfung wird der Entwickler vor einfachen Fehlern, siehe folgende Auflistung, geschützt und somit die Entwicklung vereinfacht. Es werden folgende Fehler in der Verwendung erkannt:

- Es wurden Platzhalter im Messageidentifier bei der imperativen Syntax definiert.
- Der erste Parameter der formatMessage-Funktion ist kein JSON-Objekt.
- Die Funktion wurde mit keinem oder mehr als zwei Parametern aufgerufen.
- Das JSON-Objekt beinhaltet zusätzliche Properties oder benötigte Properties wurden ausgelassen.
- Ein Identifier wurde mehrfach verwendet und die Fallback-Messages unterscheiden sich.

Die Verwendung der Übersetzungsunterstützung ist im Wiki dokumentiert und besteht aus zwei Komponenten, welche schnell erlernbar sind. Dadurch dass die Anwendung bereits JSX-Syntax verwendet müssen lediglich zwei neue Bezeichner und deren Verwendung gelernt werden.

Zuverlässigkeit

Dem Entwickler werden bei ungültiger Verwendung, soweit diese erkennbar ist und bei Kollisionen der Bezeichner Fehler mit Ursprung angezeigt. Das System toleriert teilweise Übersetzungen.

Wartbarkeit

Alle Komponenten der Übersetzungsunterstützung wurden in einzelne Module, je nach Verwendung eingeteilt. Die Erweiterungen für Babel, Webpack und die Unterstützung der Sprachdateien wird in jeweils dafür vorgesehenen Modulen erledigt.

Zur Sicherstellung der ordnungsgemäßen und korrekten Funktionsweise der Erweiterungen wurden Tests angefertigt. Die Testcases prüfen auf typische Verwendungsbeispiele und der erfolgreichen Extraktion der Messages aus dem Quellcode. Dies erhöht die Wartbarkeit des Systems, da selbst nach Modifikationen am System sichergestellt werden kann, dass die Funktionalität noch gewährleistet ist. Die Sprachdateien werden auf Vollständigkeit geprüft und Fehler dem Entwickler angezeigt. Die Erweiterungen der Buildprozesse sind auch außerhalb der Anwendung anwendbar.

Portabilität

Die Übersetzungsunterstützung kann durch die Definition einer Funktion sowie einer React-Komponente Vollständig aus dem Quellcode entfernt werden, da es außer diesen Anteilen keine weiteren Modifikationen an anderen Komponenten gibt, welche eine Abhängigkeit auf die Lokalisierung erzeugen.

5.2 Fazit

Im Rahmen dieser Bachelorarbeit wurde die Webanwendung QDAcity lokalisierbar und übersetzbar. Die in dieser Arbeit vorgestellte Lösung lässt sich auf andere auf dem React-Framework aufbaunde Webapplikationen anwenden. Durch die modulare Bauweise der einzelnen Komponenten, lassen sich diese einfach in andere Anwendungen itegrieren und erlauben somit die einfache Globalisierung von React-Anwendungen.

Literaturverzeichnis

Aykin, N. (2016). *Usability and internationalization of information technology*. CRC Press.

Dey, I. (2003). *Qualitative data analysis: A user friendly guide for social scientists*. Routledge.

ECMAScript® 2015 Language Specification. (2015)(ECMA Nr. 262). Ecma International. Geneva, CH.

ECMAScript® 2017 Internationalization API Specification. (2017)(ECMA Nr. 402). Ecma International. Geneva, CH.

Ferraiuolo, E. (2016). React-Intl Wiki — Home. Zugriff 10. Oktober 2017 unter https://github.com/yahoo/react-intl/wiki

Fry, D. & Lommel, A. (2003). *The Localization Industry Primer*. LISA. Zugriff unter https://books.google.de/books?id=GrTiNwAACAAJ

GDSG Contributors. (2005). Zugriff 20. Februar 2018 unter https://developer.gnome.org/gdp-style-guide/2.32/locale-6.html.en

Kyle, J. (2016). Super Tiny Compiler. Zugriff 15. Dezember 2017 unter https://github.com/jamiebuilds/the-super-tiny-compiler/blob/master/the-super-tiny-compiler.js

Kyle, J. (2017). Babel Handbook. Zugriff 27. Dezember 2017 unter https://git.io/babel-handbook

React Documentation. (2017). Zugriff 5. Oktober 2017 unter https://reactjs.org/docs/

System and software quality models. (2011)(ISO/IEC Nr. 25010:2011). International Organization for Standardization. Geneva, CH.

Tesch, R. (1991). Software for qualitative researchers: Analysis needs and program capabilities. *Using computers in qualitative research*, *16*, 37.

The JSON Data Interchange Syntax. (2017)(ECMA Nr. 404). Ecma International. Geneva, CH.

Vora, P. (2009). *Web application design patterns*. Morgan Kaufmann.

YPT-Team. (2014). FormatJS. Zugriff 10. Dezember 2017 unter https://formatjs.io/