Digitalisierung von Reisen durch Entwicklung einer intelligenten Reiseführer App

Studienarbeit

im Rahmen der Prüfung zum

Bachelor of Science (B.Sc.)

des Studienganges Angewandte Informatik

an der Dualen Hochschule Baden-Württemberg Karlsruhe

von

Joshua Schulz und Raphael Müßeler

2020

-Sperrvermerk-

Abgabedatum:	18. Mai 2020
Bearbeitungszeitraum:	01.10.2019 - 18.05.2020
Matrikelnummer, Kurs:	4508858, 6801150, TINF15B1
Ausbildungsfirma:	SAP SE Dietmar-Hopp-Allee 16 69190 Walldorf, Deutschland
Gutachter der Dualen Hochschule:	Thorsten Schlachter

Eidesstattliche Erklärung

Ich versichere hiermit, dass ich meine Studienarbeit mit dem Thema:

Digitalisierung von Reisen durch Entwicklung einer intelligenten Reiseführer App

gemäß § 5 der „Studien- und Prüfungsordnung DHBW Technik“ vom 29. September 2017 selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe. Die Arbeit wurde bisher keiner anderen Prüfungsbehörde vorgelegt und auch nicht veröffentlicht.

Ich versichere zudem, dass die eingereichte elektronische Fassung mit der gedruckten Fassung übereinstimmt.

Karlsruhe, den 12. Februar 2020

Schulz, Joshua; Müßeler, Raphael

Sperrvermerk

Die nachfolgende Arbeit enthält vertrauliche Daten der:

SAP SE
Dietmar-Hopp-Allee 16
69190 Walldorf, Deutschland

Sie darf als Leistungsnachweis des Studienganges Angewandte Informatik 2017 an der DHBW Karlsruhe verwendet und nur zu Prüfungszwecken zugänglich gemacht werden. Über den Inhalt ist Stillschweigen zu bewahren. Veröffentlichungen oder Vervielfältigungen der Studienarbeit - auch auszugsweise - sind ohne ausdrückliche Genehmigung der SAP SE nicht gestattet.

SAP und die SAP Logos sind eingetragene Warenzeichen der SAP SE. Die Wiedergabe von Gebrauchsnamen, Handelsnamen, Warenbezeichnungen usw. in dieser Arbeit berechtigt auch ohne besondere Kennzeichnung nicht zu der Annahme, dass solche Namen im Sinne der Warenzeichen- und Markenschutz-Gesetzgebung als frei zu betrachten wären und daher von jedem benutzt werden dürfen.

Abstract

- English -

This is the starting point of the Abstract. For the final bachelor thesis, there must be an abstract included in your document. So, start now writing it in German and English. The abstract is a short summary with around 200 to 250 words.

Try to include in this abstract the main question of your work, the methods you used or the main results of your work.

Abstract

- Deutsch -

Dies ist der Beginn des Abstracts. Für die finale Bachelorarbeit musst du ein Abstract in deinem Dokument mit einbauen. So, schreibe es am besten jetzt in Deutsch und Englisch. Das Abstract ist eine kurze Zusammenfassung mit ca. 200 bis 250 Wörtern.

Versuche in das Abstract folgende Punkte aufzunehmen: Fragestellung der Arbeit, methodische Vorgehensweise oder die Hauptergebnisse deiner Arbeit.

Inhaltsverzeichnis

Abkürzungsverzeichnis

AOP	Aspect-oriented Programming
API	Application Programming Interface
HTTP	Hypertext Transfer Protocol
IoC	Inversion of Control
JPA	Java Persistence API
JTA	Java Transaction API
OAI	OpenAPI
ORM	Object-relational Mapping
POI	Point of interest
REST	Representational State Transfer
RPC	Remote Procedure Call
SOAP	Simple Objec Access Protocol
SQL	Structured Query Language
URI	Uniform Resource Identifier
WWW	World Wide Web
XML	Extensible Markup Language
YAML	YAML Ain't Markup Language

Abbildungsverzeichnis

Tabellenverzeichnis

Quellcodeverzeichnis

1 Einleitung

(Joshua)

In der heutige Zeit unterliegt die Welt der Medien einer sehr rasanten und starken Veränderung. Es werden immer mehr neuartige und innovative Techniken entwickelt, wie Medien konsumiert werden können, z.B. argumented und virtual reality. Durch diese Techniken wird versucht die Mediennutzung effektiver, intensiver und moderner zu machen. In vielen Bereichen haben diese Techniken bereits Einzug gehalten und sind für eine große Masse von Konsumenten verfügbar, z.B. virtual reality gaming mit Hilfsmitteln wie Oculus rift und anderen Produkten.

Allerdings gibt es ebenfalls Bereiche bei denen die Digitalisierung und Nutzung neuer Techniken kaum genutzt und somit große Vorteile verschenkt werden, wie z.B. beim Reisen: Viele Menschen nutzen Reiseführer, um sich einen Überblick über ihre Reisedestination zu verschaffen und einen grundlegenden Plan zu erstellen, allerdings findet man diese Reiseführer fast ausschließlich in gedruckter Buchform. Das bedeutet, dass immer schwere Printmedien mit in den Urlaub genommen werden müssen, dass Informationen in den Reiseführern nicht aktualisiert werden können, ohne eine neue Auflage herauszugeben und eine interaktive Gestaltung der Medien nicht möglich ist. Außerdem sind die Seiten in einem Buch begrenzt, d.h. es muss für jede Reise ein neuer Reiseführer erworben werden, der spezielle Informationen zum Reiseziel enthält. All dies sind Nachteile, die durch eine Digitalisierung der Inhalte ausgeglichen werden könnten: Das Smartphone ist heutzutage ein ständiger Begleiter, der ausgenutzt werden kann um Echtzeitinformationen schnell und immer aktuell zur Verfügung zu stellen. Außerdem können interaktive Elemente wie Karten, Navigation, Audiogiudes uvm. direkt integriert werden. Es wäre möglich einen Anwendung zu schaffen, die in der Lage ist für jede beliebige Stadt und Region auf der Welt Informationen bereit zu stellen, ohne dass neue Inhalte erworben werden müssen. Damit könnte eine Reise entstehen, die unkomplizierter und trotzdem viel aktueller ist als bei Benutzung eines herkömmlichen Reiseführers.

Es könnten viele Services, die aktuell parallel zum Reiseführer genutzt werden (z.B. verschiedene Bewertungsportale und Karten) direkt integriert werden, um alle Informationen auf einen Blick zur Verfügung zu stellen. Ebenso könnte eine Personalisierung der verfügbaren Daten umgesetzt werden. Im Gegensatz zum herkömmlichen Reiseführer, welcher allgemeine und damit u.U. viele für den einzelnen irrelevant Informationen enthält, werden Vorschläge anhand der vom Nutzer gesetzten Vorlieben gemacht und somit nur nützliche Informationen zur Verfügung gestellt.

Insgesamt soll ein digitaler Reiseführer entstehen, der das Reiseerlebnis auf eine bessere, digitalere und einfachere Ebene hebt und noch mehr Spaß am Reisen erzeugen kann.

1.1 Motivation

(Joshua) Wir möchten mit dieser Arbeit einen Beitrag zu einer digitalisierten und technisch geprägten Gesellschaft leisten, indem wir eine Anwendung schaffen, die mit den Nachteilen von gedruckten Reiseführern aufräumt und eine neue innovative Art des Reisen schafft. Damit soll das Reiseerlebnis der Menschen verbessert werden und ihnen die Möglichkeit geben sich noch mehr auf das Erlebte zu konzentrieren, ohne Gedanken an eine komplizierte Planung, bei der viele Medien parallel genutzt werden, zu verschwenden. Außerdem ist diese Arbeit Teil unserer Prüfung zum Bachelor of Science und dient zur praktischen Anwendung der bisher im Studium erlernten Fähigkeiten und zum Ausprobieren neuer innovativer Techniken um unser Wissen zu erweitern.

1.2 Ausblick auf die Arbeit

2 Voraussetzungen

2.1 RESTful APIs

Representational State Transfer beschreibt ein architektonisches Modell, um Web Services zu erstellen. Sogenannte Representational State Transfer (REST) Web Services bieten Interoperabilität zwischen Computersystemen im Internet – geeignet für die Kommunikation von Maschine zu Maschine. So lässt sich REST auch als eine Abstraktion der Struktur und des Verhaltens des World Wide Webs beschreiben.

Neben REST gibt es weitere Alternativen, wie Simple Objec Access Protocol (SOAP) oder Remote Procedure Call (RPC); der Vorteil von REST besteht jedoch darin, dass durch das World Wide Web (WWW) ein Großteil der Infrastruktur für die Kommunikation bereits vorhanden und implementiert ist. Während bei
acRPC in der Uniform Resource Identifier (URI) Methodeninformationen enthalten sind, gibt eine URI in der REST Architektur ausschließlich Ressourcen an und kodiert die Funktionalität mittels Hypertext Transfer Protocol (HTTP) Methoden. Dieser Ansatz entspricht dem Konzept einer URI, da bei einer HTTP Anfrage ebenso nur Ressourcen und keine Funktionalität gefragt ist.

Eine REST API muss insgesamt sechs Eigenschaften besitzen:

Client-Server Architektur Bei REST gilt im Allgemeinen, dass eine Client-Server Architektur vorliegen soll: Die Client konsumiert die vom Server bereitgestellten Dienste.

Zustandslosigkeit Eine RESTful Application Programming Interface (API) soll keine Zustände haben, sondern so konzipiert werden, dass benötigten Informationen in einer REST-Nachricht enthalten sind. Dies begünstigt außerdem die Skalierbarkeit eines solchen Dienstes, da es auf diese Weise einfacher ist, alle Anfragen auf mehrere Instanzen zu verteilen.

Caching Server sowie Client können Antworten zwischenspeichern. Es muss jedoch vorher explizit definiert werden, welche Antworten zwischengespeichert und

welche nicht zwischengespeichert werden, um zu verhindern, dass alte oder ungeeignete Daten versendet werden.

Einheitliche Schnittstelle Die REST API muss eine einheitliche Schnittstelle zur Verfügung stellen, welche den von [1] definierten Anforderungen entsprechen muss. Dies vereinfacht die Nutzung der API.

Mehrschichtige Systeme Die Struktur einer RESTful API soll mehrschichtig sein, sodass es ausreicht, dem Client lediglich eine Schnittstelle anzubieten. Die Architektur der API wird dadurch simplifiziert und die dahinterliegenden Schichten der Implementierung bleiben verborgen.

Code on Demand Fielding beschreibt diese Eigenschaft als optional: Der Server kann, durch das Übertragen von ausführbarem Code, die Funktionalität des Clients zeitweise erweitern oder anpassen. Vorstellbar wäre beispielsweise eine Übertragung von bereits kompilierten Komponenten oder Client-seitigen Skripten.

2.2 Frameworks

2.2.1 OpenAPI

Der OpenAPI Standard ist Open Source und dient der Beschreibung von RESTful APIs. Bis 2016 war OpenAPI Teil des Swagger Frameworks, wurde schließlich aber als separates Projekt unter Aufsicht der sog. *OpenAPI Initiative* ausgelagert.

Mit der deklarativen Ressourcenspezifikation von OpenAPI können Clients Dienste verstehen und konsumieren, ohne über Kenntnisse der eigentlichen Server-Implementierung bzw. Zugriff auf den Servercode zu verfügen. Dies erleichtert die Entwicklung Client-seitiger Applikationen, die RESTful APIs verwenden. Die OpenAPI-Spezifikation ist zudem sprachunabhängig und lässt sich in jeder Beschreibungssprache (YAML Ain't Markup Language (YAML), Extensible Markup Language (XML) etc.) definieren.

2.2.2 Swagger

Swagger ist ein Framework das sich des OpenAPI Standards bedient. Swagger bietet ein Tooling an, mit dessen Hilfe APIs spezifiziert und beschrieben werden können. Neben einem entsprechenden Editor, bietet Swagger die Möglichkeit, aus der OpenAPI Spezifikation Code zu generieren und zwar unter Verwendung unterschiedlicher Frameworks – z.B. eine vollständige Spring Applikation – wobei nur noch die eigentliche Implementierung der Businesslogik erforderlich ist.

Des Weiteren stellt Swagger eine Web-basierte Benutzeroberfläche zur Verfügung, welche nicht nur die direkte Anbindung von Live APIs ermöglicht, sondern auch eine visuelle Dokumentation der API darstellt. [2]

2.2.3 Spring

Spring ist ein Open Source Framework, welches in Java geschrieben wurde. Es gilt als de facto Standard bei der Entwicklung von RESTful API, da es in der Open Source Welt viel Zuspruch und Verwendung gefunden hat. Zudem integriert Spring mit fast allen Java Umgebungen und ist somit nicht nur für Anwendungen im kleinen Maßstab, sondern eben so für Anwendungen in großen Unternehmen geeignet. [3]

Bei der Entwicklung dieses Frameworks wurden die unter anderem in [4] beschriebenen Design Prinzipien mithilfe der folgenden Module und deren entsprechender Funktionalität umgesetzt:

Dependency Injection

Der von Martin Fowler 2004 definierte Begriff der *Dependency Injection* ist eine Präzisierung oder Spezialisierung des Begriffs *Inversion of Control*. Inversion of Control (IoC) bezeichnet ein Paradigma, welches den Kontrollfluss einer Applikation nicht mehr der Anwendung, sondern dem Framework – in diesem Fall Spring – überlässt. Ein Beispiel für IoC sind Listener (Beobachter Muster).
Dependency Injection beschreibt ein Entwurfsmuster, bei welchem festgelegte Abhängigkeiten nicht zur Kompilierzeit, sondern zur Laufzeit bereitgestellt werden. Dies

lässt sich einem Beispiel erläutern: Besteht bei der Initialisierung eines Objektes eine Abhängigkeit zu einem anderen Objekt, so wird diese Abhängigkeit an einem zentralen Ort hinterlegt. Wenn nun die Initialisierung dieses Objektes erfolgt, beauftragt es den sog. Injector (dt. Injezierer), die Abhängigkeit aufzulösen. [5]

In Spring bietet der IoC Container mittels Reflexion ein konsistentes Werkzeug zur Konfiguration sowie Verwaltung von Java Objekten. Diese durch den Container erstellten Objekten heißen *Beans*. Die Konfiguration des Containers erfolgt entweder über eine XML Datei oder über Java Annotationen. [3]

Aspektorientierte Programmierung

Aspect-oriented Programming (AOP) beschreibt ein Paradigma und ermöglicht die klassenübergreifende Verwendung generischer Funktionalität. Die führt zu einer starken Modularisierung und es gibt eine klare Trennung zwischen der Anwendungslogik und der Businesslogik (Cross-cutting Concern). [6]
Das Schreiben von Logs stellt ein Beispiel für ein Cross-cutting Concern dar, da eine Logging-Strategie alle protokollierten Klassen und Methoden erfasst und somit durchaus mit der Anwendungs- sowie der Businesslogik in Berührung kommt.

Transaktionsmanagement

Ein weiteres Beispiel für AOP ist sog. Transaktionsmanagement. Eine Transaktion bezeichnet in der Informatik eine logische Einheit, mit dessen Hilfe Aktionen auf einer Persistenz ausgeführt werden können. Dabei ist sichergestellt, dass sobald die Transaktion fehlerfrei und vollständig abgeschlossen ist, der Datenbestand weiterhin konsistent ist. Im Umkehrschluss bedeutet das, dass eine Transaktion entweder vollständig oder gar nicht ausgeführt wird. [7]

Das von Spring bereitgestellte Transaktionsmanagement stellt eine Abstraktion auf die Java Plattform dar und ist in der Lage mit globalen und verschachtelten Transaktionen sowie sog. Savepoints – ein Punkt innerhalb einer Transaktion, zu welchem im Fehlerfall zurück gesprungen werden kann – zu arbeiten. Außerdem lasst sich diese Abstraktion in fast allen Java Umgebungen einsetzen. Die von Java bereitgestellte Ja-

va Transaction API (Java Transaction API (JTA)) hingegen unterstützt nur globale und verschachtelte Transaktionen und erfordert zudem immer einen Applikationsserver.

Model View Controller

Das **MVC!** (**MVC!**) (Model View Controller) Pattern ist ein weit verbreiteter Mechanismus zur Entwicklung von Benutzeroberflächen. **MVC!** stellt ein Design Pattern dar, dass Kapselung sowie eine Struktur für eine Architektur von Benutzeroberflächen bietet und bei welcher jeder Bereich eine definierte Aufgabe hat. Eine Verletzung der Zuständigkeitsbereiche ist zu vermeiden. [8]

Das Pattern besagt, dass die Architektur von Benutzerschnittstellen in folgende Bereiche aufgeteilt ist: Das *Model* ist für den Zugriff auf die Datenbank und die Beschaffung von Daten zuständig. Häufig ist das Model auch für die Aufbereitung der Daten zuständig. Somit liegt die meist aufwändige Logik nicht beim Client, sondern bei den Servern, welche zumeist auch mit besserer Hardware ausgestattet sind.
Ein *Controller* definiert die Art und Weise, wie die Benutzerschnittstelle auf die Eingaben des Benutzer reagiert. Des Weiteren ist ein Controller für das Aktualisieren der Daten im Datenmodell, aber auch auf dem View zuständig.
Der *View* bestimmt ausschließlich, wie die Benutzeroberfläche aussehen soll. Er enthält – in den meisten Implementierungen – keine Logik, sondern ist lediglich eine Definition und Anordnung der Benutzeroberflächenelemente.

Spring definiert für alle Verantwortlichkeiten eigene Strategie-Interfaces, wie beispielsweise das Controller Interface, welches alle eingehenden HTTP Requests definiert und darüber hinaus auch behandelt.

2.2.4 Hibernate

Hibernate ist ein Persistenz- und Object-relational Mapping (Object-relational Mapping (ORM)) Framework, das ebenfalls unter einer Open Source Lizenz veröffentlicht und in Java geschrieben ist. Hibernate bietet eine Abstraktionsstufe gegenüber relationalen Datenbankimplementationen. Mithilfe der Sprache *Hibernate Query Language* und dem entsprechend konfigurierten Dialekt (z.B. MySQL Dialekt, MariaDB Dialekt

etc.) werden die entsprechenden Statements erzeugt und schließlich ausgeführt. Dies ermöglicht den einfach und schnellen Umstieg von einer Datenbankimplementation auf die andere, ohne Anpassung der sich im Code befindlichen Queries.

Object-relational Mapping

Eine häufig eingesetzt Technik der Persistierung sind relationale und meist auch Structured Query Language (SQL) basierte Datenbanken wie beispielsweise MySQL oder MariaDB. Wenn jedoch die Anwendung, welche die Businesslogik enthält, der Objektorientierung folgt, kommt es zu einem Widerspruch – dem sog. Object-relational impendance mismatch –, welcher in den unterschiedlichen Paradigmen begründet liegt. So beschreibt die Objektrelationale Abbildung eine Technik, bei welcher sich Objekte einer objektorientierten Sprache in einer relationalen Datenbank persistieren lassen.

Java bietet mit der sog. Java Persistence API (Java Persistence API (JPA)) eine Abstraktion genau zu diesem Zweck, dessen sich Hibernate auch bedient. Mittels Annotationen lassen sich Objekte mit Attributen und Methoden – zumeist Plain Old Java Objects (**POJO!** (**POJO!**)s) – auf Entitäten abbilden. Diese Annotation definieren, welche Tabelle auf welches Objekt und welche Spalte auf welches Attribute abgebildet wird. Es lassen sich außerdem die Relationen der Entitäten auf die Assoziationen der Objekte abbilden. Hibernate bzw.JPA unterstützt 1:1, 1:N sowie N:N Relationen. Somit wird ein vollständiges Abbild der Persistenz in der Anwendung geschaffen.

Die einzige Vorgabe bei der Definition der Objekte ist, dass ein parameterloser Konstruktor existieren muss. Hibernate greift auf die Attribute der Klasse mittels Reflexion zu.

Transaktionsmanagement

In Hibernate erfolgt der Zugriff auf die Persistenz über sogenannte *Sessions*. Eine Session repräsentiert eine physische Verbindung zwischen der Persistenz und der Anwendung und bietet Methoden für alle Datenbestandsoperationen. Der Lebenszyklus einer *Session* ist durch den Beginn und das Ende einer logischen Transaktion begrenzt. Es lässt sich konfigurieren, ob Hibernate das Sessionmanagement übernimmt, oder die Anwendung selbst die *Sessions* öffnet und wieder schließt. So werden auch par-

allele Datenbankverbindung und damit auch eine Performance Verbesserung ermöglicht.

Um eine Session mittels des Sessionmanagements zu erstellt, wird sich der *SessionFactory* bedient, von welcher meist nur eine Instanz in der Applikation existiert. Diese beinhaltet die Konfiguration, die den Verbindungsaufbau und die Verbindung selbst definiert.

Eine Session ermöglicht es eine *Transaction* – Abstraktion der Implementation von JTA – zu starten und zu beenden. Wie bereits in Abschnitt 2.2.3 beschrieben, lässt sich hierbei das Transaktionsmanagement sehr gut integrieren, sodass Spring das Starten und Beenden von Transaktionen handhabt. Dies führt zu einer Simplifizierung des Codes, der zum einen lesbarer und zum anderen einfach wird.

2.2.5 Android

2.2.6 Kotlin

2.3 Qualitätssichernde Maßnahmen

2.3.1 Code Review

2.3.2 Continuous Integration

2.3.3 Testing Frameworks

3 Datengrundlage

(Joshua)

Die Datengrundlage ist für einen Reiseführer sehr wichtig, da der gesamte Sinn eines Reiseführers darauf basiert, Informationen aufzubereiten und an den Leser/Nutzer weiter zu geben. Aus diesem Grund wurden für diese Arbeit mehrere Datenquellen zu unterschiedlichen Themen herausgesucht und verglichen. Im Folgenden sollen diese Alternativen und die angestellten Überlegungen sowie die endgültige Entscheidung, welche Daten für *travlyn* verwendet werden sollen, aufgezeigt.

3.1 Points of Interest

Travlyn soll laut Spezifikation Trips erstellen können, welche eine Abfolge von interessanten und sehenswerten Punkten in einer Stadt ist. Diese Point of interest (POI) sollen über ein API in die App integriert werden. Folgende Anforderungen sind an die Informationen und die API gestellt:

- Die abgefragten Daten sollten möglichst für einen kommerziellen Nutzen zugelassen sein, damit während der Entwicklung keine schwierigen Lizenzfragen auftreten können und ggf. höhere Datenvolumen durch Nachfragen erreicht werden können.
- Da diese Arbeit ein Studienprojekt ist, für welches sehr begrenzte Ressourcen zur Verfügung stehen sollte die API kostenfrei benutzbar sein.
- Die API sollte Daten für möglichst viele Städte/Orte zur Verfügung stellen, damit *travlyn* möglichst überall eingesetzt werden kann.
- Zu den einzelnen POIs sollten neben dem Namen und der Position weitere Daten wie Beschreibungen, Öffnungszeiten und ggf. Bilder bereitgestellt werden.

3.1.1 Google Places API

Google ist einer der größten Anbieter von Ortsbasierten Services/Diensten und stellt eine API für POIs zu Verfügung [9]. Diese API hat sehr weit gefächerte Funktionen, die von einer einfachen Suche über ausführliche Details zu interessanten Orten bis hin zu „user check-in“ an einzeln Orten reichen. Diese große Funktionalität wäre für *travlyn* sehr wertvoll. Allerdings sind die Google APIs nicht frei zugänglich und die Anzahl der Requests ist u.U. stark eingeschränkt [10]. Außerdem ist die Nutzung der erhaltenen Daten nur in Verbindung mit anderen von Google bereitgestellten Services erlaubt [11], somit wäre die ganze *travlyn* Applikation an Google gebunden.

3.1.2 Openroute service

Openroute service [12] wird vom Heidelberger Institut für Geoinformationstechnik angeboten. Es handelt sich um eine Crowd Sourced API, d.h. sie wird durch Benutzer über OpenStreetMap (OSM) [13] gespeist und ist damit frei zugänglich. Durch die Nutzung von OSM ergibt sich der weitere Vorteil, dass die API für Orte weltweit nutzbar ist. Leider sind die gelieferten Informationen nicht sehr umfangreich und beinhalten häufig keine genauere Beschreibung und keine Bewertung o.Ä.. Weiterhin sind Crowd Sourced Informationen meist nicht offiziell verifiziert und könnten u.U. falsch sein. Die Beschränkungen für diese API sind relativ gering (500 POIs requests pro Tag), allerdings können diese auf Nachfrage erhöht werden (z.B. für Bildungszwecke).

> **Crowdsourcing:** Beim Crowdsouring wird das Wissen, die Kreativität oder die Arbeitskraft der Masse ausgenutzt. Jeder leistet einen kleinen Teil und zusammen ergibt sich ein großes Ganzes. Typische Beispiele sind z.B. Wikipedia oder die Klassifikation von Daten zum Machine Learning. Allerdings können diese Daten von jedem bewusst oder unbewusst verfälscht werden und sie sind sehr schwer zu verifizieren [**Winkler.2009**].

3.1.3 Foursquare

Die Firma Foursquare bietet ebenfalls eine API an [14], über die Informationen zu interessanten Orten gelesen werden können. Die API ist weltweit einsetzbar und liefert sehr viele Informationen zu einzelnen Orten, wie Ratings, kurze Beschreibungen oder Adressen von denen Bilder in der gewünschten Auflösung abgefragt werden können. Die Anzahl der möglichen Requests kann durch die Registrierung einer Kreditkarte (trotz der kostenlosen Nutzung) auf ca. 100.000 pro Tag gesteigert werden. Allerdings können die kostenfreien Varianten dieser API nicht für kommerzielle Zwecke genutzt werden und die abgefragten Daten dürfen nicht länger als 24 Stunden persistiert werden.

3.1.4 Evaluation der Alternativen

Für das Projekt wurde aus den obigen Alternativen gewählt. Die endgültige Entscheidung fiel auf den OpenRoute service, da es für unser relativ kurzes Projekt ein sehr großes Risiko darstellt, dass langwierige Nachforschungen zu Lizenzfragen und erlaubten Einsatzmöglichkeiten angestellt müssen. Openroute service ist selbst für kommerziellen Nutzen freigegeben und aus rechtlicher Perspektive damit sehr einfach zu nutzen. Im Gegensatz dazu stellen sowohl Google als auch Foursquare starke Einschränkungen auf, die ein nicht abzusehendes Risiko darstellen, z.B. sind bestimmte Funktionen unter diesen Einschränkungen umsetzbar. Trotz diesem entscheidenden Vorteil muss der Nachteil in Kauf genommen werden, dass nur mäßig ausführliche Informationen gelesen werden können. Im weiteren Verlauf dieser Arbeit werden weitere zusätzliche Datenquellen aufgezeigt, um diesen Nachteil möglichst gut aufzufangen.

3.2 Weiterführende Informationen zu POIs und Städten

Um die eingeschränkte API Openroute service auszugleichen und dem Nutzer weitere Informationen zu seinem Reiseziel und zu besuchenden Orten zu bieten müssen weitere APIs angefragt werden.

Die wahrscheinlich bekannteste und ausführlichste Datenquelle ist Wikipedia. Dies ist einen Crowd Sourced Enzyklopädie die von allen Nutzern gespeist werden kann. Aus diesem Grund ist die Nutzung direkt im Internet aber auch per API Zugriff kostenlos und frei nutzbar. Allerdings ist zu beachten, dass die enthaltenen Inforationen durch jeden verändert und ggf. gefälscht werden können und Wikipedia deshalb keine sichere Quelle für wissenschaftliche Arbeiten o.Ä. darstellt. Für den in dieser Arbeit vorliegenden Use Case wurde entscheiden, dass dieses Risiko annehmbar ist und der Vorteil der sehr großen Wissensbasis das Risiko überwiegen.

Für den Zugriff auf Wikipedia gibt es unterschiedliche Möglichkeiten, im Folgenden werden zwei APIs beschreiben, die ausprobiert worden sind:

- **MediaWiki**: Hinter Wikipedia und vielen anderen Wiki-Seiten steht die selbe Software: MediaWiki [15]. Diese Software bietet eine sogenannte *MediaWiki action API*, die viele Informationen zu allen Artikeln eines Wiki zurückliefern kann. Leider sind die Daten nicht über einen zentralen Aufruf abrufbar sondern es werden mehrere Abfragen in folge benötigt, um z.B. die URL eines der Bilder des Artikels zu ermitteln.
- **DBpedia**: Die zweite API, die mithilfe eines Protypes getestet wurde ist *DBpedia* [16]. Auch diese API folgt dem crowd souring Prinzip und ist somit frei zugänglich. Zusätzlich können dort alle Informationen über einen zentralen Zugriff abgerufen werden, indem die benötigten Daten über URL-Parameter spezifiziert werden können. Außerdem bietet diese API die Daten in aufbereiteter Form an: Links können direkt aufgelöst werden, es wird das selbe Thumbnail ausgeliefert welches im original Artikel ausgewählt ist und es kann auf alle Daten der kompakten Infobox in der oberen rechten Ecke strukturiert zugegriffen werden.

Durch die einfachere Handhabung der *DBpedia* API wurde für den weiteren Verlauf entschieden auf diese API zu setzen und alle Informationen zu POIs und Städten von diesem Zugang abzufragen. Damit können erweiterte Infos geladen werden, die dem Nutzer während seines Trips kontinuierlich angezeigt werden können, um die Erfahrung weiter zu verbessern.

4 Konzept

4.1 Server Architektur

4.2 Client Architektur

4.3 Kommunikationsschema

5 Implementierung

5.1 Coding Conventions

5.2 Probleme

6 Evaluation

6.1 User Zufriedenheit

6.2 Aufbau

6.3 Ablauf

6.4 Ergebnis

7 Fazit

7.1 Ausblick

Literaturverzeichnis

[1] Roy Thomas Fielding. „Architectural Styles and the Design of Network-based Software Architectures“. In: (2000). URL: https://www.ics.uci.edu/~fielding/pubs/dissertation/fielding_dissertation.pdf.

[2] SmartBear. *The Best APIs are Built with Swagger Tools | Swagger*. 2020. URL: https://swagger.io/.

[3] Walls, C./ Carnell, J. *Spring Boot in action // Spring microservices in action*. Shelter Island: Manning und Manning Publications Co, 2016 // 2017. URL: https://www.nitinagrawal.com/uploads/2/1/3/6/21361954/spring_boot_in_action.pdf.

[4] Johnson, R. *Expert one-on-one J2EE design and development*. Programmer to programmer. Indianapolis, Ind. und Great Britain: Wrox, 2003.

[5] Martin Fowler. *Inversion of Control Containers and the Dependency Injection pattern*. 23.01.2020. URL: https://www.martinfowler.com/articles/injection.html#InversionOfControl.

[6] Wunderlich, L. *AOP: Aspektorientierte Programmierung in der Praxis*. S- & -S-pockets. Frankfurt [Main]: Entwickler.press, 2005.

[7] Özsu, M. T./ Valduriez, P. *Principles of distributed database systems*. 3rd ed. New York: Springer Science+Business Media, 2011.

[8] Gamma, E. *Design patterns: Elements of reusable object-oriented software / Erich Gamma ... [et al.]* Addison-Wesley professional computing series. Reading, Mass. und Wokingham: Addison-Wesley, 1995.

[9] Google. *Geo-location APIs | Google Maps Platform | Google Cloud*. 01.02.2020. URL: https://cloud.google.com/maps-platform/.

[10] Singhal, M./ Shukla, A. „Implementation of Location based Services in Android using GPS and Web Services“. In: 9.1 (2012), S. 237.

[11] Google. *Google Maps APIs Terms of Service | Google Maps Platform*. 02.12.2019. URL: https://developers.google.com/maps/terms-20180207?hl=en.

[12] The Heidelberg Institute for Geoinformation Technology. *Openrouteservice*. URL: https://openrouteservice.org/.

[13] OpenStreetMap. *OpenStreetMap Deutschland: Die freie Wiki-Weltkarte*. URL: https://www.openstreetmap.de/.

[14] Foursquare. *Foursquare Location Intelligence for Enterprise*. URL: https://enterprise.foursquare.com/products/places.

[15] MediaWiki. *MediaWiki*. 24.01.2020. URL: https://www.mediawiki.org/wiki/MediaWiki.

[16] DBpedia. *About | DBpedia*. 02.02.2020. URL: https://wiki.dbpedia.org/about.