



# 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 3. 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

<b>Abkürzungsverzeichnis</b>	<b>VI</b>
<b>Abbildungsverzeichnis</b>	<b>VII</b>
<b>Tabellenverzeichnis</b>	<b>VIII</b>
<b>Quellcodeverzeichnis</b>	<b>IX</b>
<b>1 Einleitung</b>	<b>1</b>
1.1 Motivation . . . . .	2
1.2 Ausblick auf die Arbeit . . . . .	2
<b>2 Voraussetzungen</b>	<b>3</b>
2.1 RESTful APIs . . . . .	3
2.2 Frameworks . . . . .	4
2.3 Qualitätssichernde Maßnahmen . . . . .	9
<b>3 Datengrundlage</b>	<b>10</b>
<b>4 Konzept</b>	<b>11</b>
4.1 Server Architektur . . . . .	11
4.2 Client Architektur . . . . .	11
4.3 Kommunikationsschema . . . . .	11
<b>5 Implementierung</b>	<b>12</b>
5.1 Coding Conventions . . . . .	12
5.2 Probleme . . . . .	12
<b>6 Evaluation</b>	<b>13</b>
6.1 User Zufriedenheit . . . . .	13
6.2 Aufbau . . . . .	13
6.3 Ablauf . . . . .	13
6.4 Ergebnis . . . . .	13
<b>7 Fazit</b>	<b>14</b>
7.1 Ausblick . . . . .	14
<b>Literaturverzeichnis</b>	<b>X</b>

# Abkürzungsverzeichnis

<b>AOP</b>	Aspect-oriented Programming
<b>API</b>	Application Programming Interface
<b>HTTP</b>	Hypertext Transfer Protocol
<b>IoC</b>	Inversion of Control
<b>JPA</b>	Java Persistence API
<b>JTA</b>	Java Transaction API
<b>OAI</b>	OpenAPI
<b>ORM</b>	Object-relational Mapping
<b>REST</b>	Representational State Transfer
<b>RPC</b>	Remote Procedure Call
<b>SOAP</b>	Simple Object Access Protocol
<b>SQL</b>	Structured Query Language
<b>URI</b>	Uniform Resource Identifier
<b>WWW</b>	World Wide Web
<b>XML</b>	Extensible Markup Language
<b>YAML</b>	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. augmented 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, Audioguides uvm. direkt integriert werden. Es wäre möglich eine 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 Object 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 herrschen 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 vorhanden sind. Dies begünstigt außerdem die Skalierbarkeit eines solchen Dienstes, da auf diese Weise einfach alle Anfragen auf mehrere Instanzen verteilt werden können.

**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 Auflagen entsprechen muss. Dies vereinfacht die Nutzung der API.

**Mehrschichtige Systeme** Die Struktur einer RESTful API soll mehrschichtig sein, so dass es ausreicht, dem Client lediglich eine Schnittstelle anzubieten. Die Architektur der API wird dadurch simplifiziert und die dahinterliegenden Schichten der Implementierung bleibt 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 Open-API 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, fragt 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 lässt 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 im Server, 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 auch 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 einfachen und schnellen Umstieg von einer Datenbankimplementierung auf die andere, ohne Anpassung der sich im Code befindlichen Queries.

### Object-relational Mapping

Eine häufig eingesetzte 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 impedance 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 Attribut 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.

Bei der Definition solcher Objekte ist die einzige Vorgabe, 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-

alle Datenbankverbindung und damit auch eine Performance Verbesserung ermöglicht.

Um eine Session mittels des Sessionmanagements zu erstellen, 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 einfacher 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

## 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](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](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.