



Evaluierung von REST Frameworks für Android

Basierend auf der Fallstudie von Revex2020, einer Web-Anwendung für die Revitalisierung von Wasserkraftwerken

BACHELORARBEIT

zur Erlangung des akademischen Grades

Bachelor of Science

im Rahmen des Studiums

Software & Information Engineering

eingereicht von

Elisabeth Pilz

Matrikelnummer 1225231

ausgeführt am
Institut für Rechnergestützte Automation
Forschungsgruppe Industrial Software
der Fakultät für Informatik der Technischen Universität Wien

Betreuung: Thomas Grechenig

Mitwirkung: Dominik Moser

Wien, 3. März 2016

Kurzfassung

Einer der größten Trends am Business-Markt ist die Mobilisierung der Geschäftswelt. Dabei wird unabhängig von Stakeholdern, Zeit, Ort und Geräten auf unternehmensrelevante Daten zugegriffen. Ein wesentlicher Innovationsstrang dabei ist, die Entwicklung von Business-Apps. Mobile Endgeräte werden in bestehende Geschäftsprozesse der Unternehmen integriert und greifen dabei auf definierte Schnittstellen oder angebotene Dienste zu. Großteils nutzen Unternehmen Web Services, um die Daten zur Verfügung zu stellen. Mobile Applikationen rufen diese dann oftmals über eine REST-Schnittstelle auf.

Diese Arbeit präsentiert anhand des Projekts Revex2020 verschiedene REST-Frameworks für Android. Dafür wird mit jedem Framework eine Beispiel-App implementiert, welche Daten aus dem Revex2020 Backend abrufen soll. Die Beispielimplementierung der Frameworks soll unter anderem die Unterschiede dieser aufzeigen. Die Qualität der Frameworks wird anhand vorher definierter Kriterien evaluiert. Schlussendlich soll es diese Evaluierung ermöglichen, ein geeignetes Framework für die Implementierung der Revex2020 App auszuwählen.

Schlüsselwörter

REST, Android, Java, App, Smartphone, Mobile Geräte, Revex2020, Framework, Retrofit, Jersey, Spring for Android, AndroidAnnotations

Abstract

One of the biggest trends in the business market is the mobilization of the business community. The access to business-related data is independent of stakeholders, time, place and devices. An essential innovation strand is the development of business apps. Mobile devices are integrated into existing business processes of the companies and thereby access defined interfaces or provided services. The majority of companies use web services to provide access to the data. Mobile applications often call web services up via a REST interface.

This paper presents various REST frameworks for Android on the basis of the Revex2020 project. With the aim of retrieving data from the Revex2020 backend, each framework will be implemented with a sample app. The sample implementation of the frameworks will show the differences between these among other things. The quality of the frameworks will be evaluated according to predefined criteria. Finally, this evaluation should make it possible to select a suitable framework for the implementation of the Revex2020 app.

Keywords

REST, Android, Java, App, Smartphone, Mobile devices, Revex2020, Framework, Retrofit, Jersey, Spring for Android, AndroidAnnotations

Inhaltsverzeichnis

| | | |
|----------|--|-----------|
| 1 | Einleitung | 1 |
| 1.1 | Problemstellung | 1 |
| 1.2 | Motivation | 1 |
| 1.3 | Zielsetzung | 2 |
| 1.4 | Methodik | 3 |
| 2 | State of the Art | 5 |
| 3 | Android | 8 |
| 3.1 | Überblick | 8 |
| 3.2 | Architektur | 8 |
| 3.3 | Cross Compiling | 10 |
| 3.4 | Android und Java | 10 |
| 3.5 | Android Versionen | 10 |
| 4 | Rest | 12 |
| 5 | Vorstellung der Frameworks | 13 |
| 5.1 | Retrofit | 13 |
| 5.1.1 | Allgemein | 13 |
| 5.1.2 | Funktionsweise | 14 |
| 5.2 | Spring for Android | 16 |
| 5.2.1 | Allgemein | 16 |
| 5.2.2 | Funktionsweise | 16 |
| 5.3 | Jersey | 19 |
| 5.3.1 | Allgemein | 19 |
| 5.3.2 | Verwendung unter Android | 19 |
| 5.3.3 | Funktionsweise | 20 |
| 5.4 | AndroidAnnotations | 22 |
| 5.4.1 | Allgemein | 22 |
| 5.4.2 | Funktionsweise | 23 |
| 5.4.3 | zusätzliche Dienste | 24 |
| 6 | Vergleich | 25 |
| 6.1 | Gegenüberstellung der Frameworks | 25 |
| 6.2 | Evaluierung im Kontext Revex2020 | 32 |
| 6.3 | Persönliches Fazit zu den Frameworks | 32 |
| 7 | Zusammenfassung | 34 |
| | Literatur | 35 |
| | Wissenschaftliche Literatur | 35 |
| | Online Referenzen | 36 |

Abbildungsverzeichnis

| | | |
|-----|---|----|
| 1.1 | Use-Case-Diagramm | 2 |
| 2.1 | Github Stars, aktualisiert am 01.03.2016 | 6 |
| 2.2 | Stackoverflow Questions, aktualisiert am 01.03.2016 | 6 |
| 2.3 | Github Commits, aktualisiert am 01.03.2016 | 6 |
| 2.4 | GitHub Last Commit, aktualisiert am 01.03.2016 | 7 |
| 3.1 | Marktanteil von mobilen Betriebssystemen [12] | 8 |
| 3.2 | Android Architektur [13] | 9 |
| 3.3 | Android Plattform Versionen [19] | 11 |
| 6.1 | Zeitmessung der GET Requests | 27 |
| 6.2 | Zeitmessung der POST Requests | 28 |
| 6.3 | CPU Auslastung | 28 |
| 6.4 | benötigter RAM | 29 |

Liste der Listings

| | | |
|------|--|----|
| 5.1 | Retrofit build.gradle | 14 |
| 5.2 | Retrofit Builder | 14 |
| 5.3 | Auszug aus dem PowerPlantService | 15 |
| 5.4 | Hinzufügen des Tokens, für gültigen Login | 15 |
| 5.5 | Spring for Android build.gradle | 16 |
| 5.6 | GET Request um alle Kraftwerke zu erhalten | 17 |
| 5.7 | Abfrage der Daten zu einem speziellen Kraftwerke | 17 |
| 5.8 | POST Request um ein neues Kraftwerke anzulegen | 18 |
| 5.9 | Workaround Jersey Client auf Android | 20 |
| 5.10 | GET Request | 20 |
| 5.11 | AndroidAnnotations build.gradle | 23 |
| 5.12 | Auszug aus dem PowerPlantService Interface | 23 |
| 5.13 | Interceptor für das setzen eines zusätzlichen HTTP Header Feldes | 24 |

Abkürzungen

API Application Programming Interface

CDDL Common Development and Distribution License

CPU Central Processing Unit

DI Dependency Injection

GPL GNU General Public License

HATEOAS Hypermedia as the Engine of Application State

HTTP Hypertext Transfer Protocol

JSON JavaScript Object Notation

OSS Open Source Software

POJO Plain Old Java Object

Q&A Question and Answer

RAM Random Access Memory

REST Representational State Transfer

SE Standard Edition

SOAP Simple Object Access Protocol

URI Uniform Resource Identifier

VM Virtual Machine

XML Extensible Markup Language

1 Einleitung

1.1 Problemstellung

Einer der größten Trends auf den Business-Markt ist die Mobilisierung der Geschäftswelt, die sich in den verschiedensten Unternehmensstrategien widerspiegelt. Es gibt zahlreiche Innovationen, um unabhängig von Stakeholdern, Zeit, Ort und Geräten auf Daten und Anwendungen zuzugreifen. Ein wesentlicher Innovationsstrang ist dabei die Entwicklung von Business-Apps, um beispielsweise die Arbeitszeiten auf Geschäftsreisen effektiv ausnützen zu können. Dadurch hat die Bedeutung der Informations- und Kommunikationsindustrie in den letzten Jahren in den Unternehmen stetig zugenommen [1].

Durch die immer stärkere Nachfrage nach mobilen Apps im Arbeitsalltag ist es notwendig, mobile Endgeräte in bestehende Geschäftsprozesse der Unternehmen zu integrieren. Dabei soll es vermieden werden, eine komplett neue IT-Infrastruktur unter Beteiligung von mobilen Endgeräten zu schaffen. In vielen Unternehmen wird daher die IT-Anwendungslandschaft an das Paradigma der serviceorientierten Architektur ausgerichtet. Ein wesentlicher Vorteil dabei ist, dass wohl definierte Schnittstellen vorhanden sind und angebotene Dienste flexibel und plattformunabhängig genutzt werden können. Sollen nur mobile Anwendungen in die existierende IT-Anwendungslandschaft eingegliedert werden, bedeutet dies in der serviceorientierten Architektur, das Web Services benötigt werden. In der Praxis werden Web Services entweder mit dem Kommunikationsprotokoll Simple Object Access Protocol (SOAP) oder Representational State Transfer (REST) umgesetzt [2].

Im Revex2020 Projekt wird das Kommunikationsprotokoll REST verwendet, dadurch ist es nötig ein geeignetes Framework aufseiten der mobilen App zu finden, dass eine vollständige und korrekte Anbindung an den Webservice ermöglicht. Es existieren bereits zahlreiche Frameworks, die eine REST Implementierung unterstützen, diese unterscheiden sich aber stark in der Qualität und im Funktionsumfang. Auch bieten nicht alle diese Frameworks eine Unterstützung für Android an. Daher ist die Auswahl eines geeigneten Frameworks für eine erfolgreiche Implementierung ausschlaggebend.

1.2 Motivation

Die Thematik rund um REST-Frameworks für Android ist noch relativ neu, dadurch ist es nicht möglich ohne größere Recherchen ein geeignetes Framework für das Projekt Revex2020 auszuwählen. Es gibt zwar einige Vergleiche von REST Frameworks, wie etwa die Fachstudie von Markus Fischer, Kalman Kepes und Alexander Wassiljew [3]. In dieser Studie wird allerdings nicht darauf eingegangen, ob die Frameworks eine Implementierung clientseitig mit Android unterstützen, es wird vermehrt auf die serverseitige Implementierung eingegangen. Die Möglichkeit der clientseitigen Implementierung ist aber eine essenzielle Anforderung, da eine Business-App für Android entwickelt werden soll.

Der immer stärker wachsende Bereich von mobilen Anwendungen macht das zu untersuchende

Thema besonders interessant. Herkömmliche Software rückt immer weiter in den Hintergrund, Daten sollen sofort und überall abgerufen werden können. Mobile Endgeräte wie Smartphone und Tablets verändern daher die Geschäftswelt nachhaltig, Führungskräfte und Mitarbeiter erhalten jederzeit Zugang zu Unternehmensinformationen und -prozessen. Die Unternehmen der Zukunft werden daher mobil [4].

Revex2020 ist ein Forschungsprojekt zur Revitalisierung von Wasserkraftwerken, das in Kooperation mit dem Institut für Energietechnik und Thermodynamik entwickelt wird [5]. Ein Ziel dieses Projektes ist es, Mitarbeitern zukünftig zu ermöglichen, mithilfe von mobilen Endgeräten den Zustand einzelner Kraftwerkskomponenten vor Ort erfassen zu können. Es soll eine Android-App entwickelt werden, die das bereits vorhandene Backend, über das REST-Webservices nutzt um exemplarisch den Anwendungsfall abzubilden.

1.3 Zielsetzung

Ziel dieser Bachelorarbeit ist die Evaluierung verschiedener REST-Frameworks für Android im Kontext des Revex2020 Projekts, um eine unkomplizierte Anbindung an das bereits vorhandene Backend zu ermöglichen. Dazu werden bestehende REST-Frameworks für Android getestet, indem diese in einem Anwendungsfall eingesetzt werden. Nach der Evaluierung dieser Frameworks soll eine Empfehlung abgegeben werden, welches sich am besten für das Revex2020 Projekt eignet.

Im Rahmen dieser Bachelorarbeit werden die Begriffe Framework und Bibliothek (Java Libraries) äquivalent behandelt, da die Übergänge im Android Umfeld sehr fließend sind. Ein Framework ist in der Regel komplexer. Es definiert einen Rahmen, wo die Entwickler später eigene Dienste einbauen. Frameworks legen daher zu einem gewissen Teil die Struktur der Anwendung fest, wohingegen Bibliotheken lediglich Klassen und Funktionen bereitstellen [6].

Die Evaluierung der Frameworks erfolgt anhand von Prototypen, indem die REST-Frameworks verwendet werden. Es wurden im Vorfeld verschiedene Anwendungsfälle definiert (siehe Abbildung 1.1), indem die einzelnen REST Frameworks integriert werden. Dabei werden in einem Szenario verschiedene Prozesse durchgespielt, wie Kraftwerk erstellen, löschen, bearbeiten und anzeigen. Als Vorlage dazu wurde die bestehende Web-Applikation des Projektes verwendet.

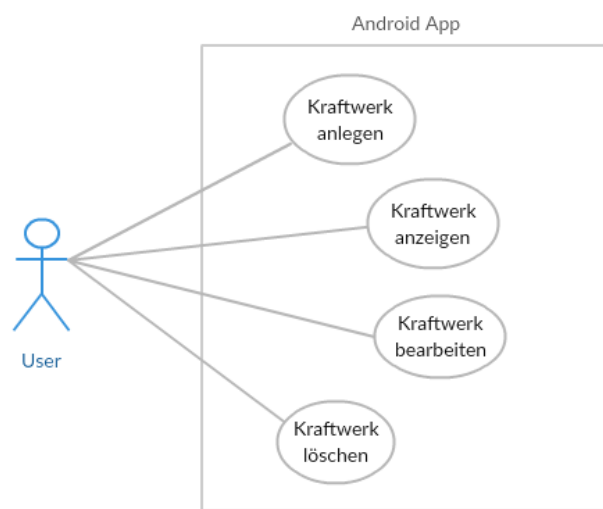


Abbildung 1.1: Use-Case-Diagramm

1.4 Methodik

Die Qualität der einzelnen Frameworks soll anhand folgender Kriterien verglichen werden, welche an dem Kriterienkatalog der Fachstudie "Vergleich von Frameworks zur Implementierung von REST-basierten Anwendungen"[3] angelehnt sind. Dieser Kriterienkatalog beschäftigt sich mit den Eigenschaften für die Evaluierung von REST Frameworks, vor allem auf serverseitiger Sicht. Der Kriterienkatalog wurde deshalb gekürzt bzw. einzelne Punkte zusammengefasst und abgeändert, um eine Evaluierung im Kontext des Projektes Revex2020 durchführen zu können.

Das Hauptaugenmerk der Evaluierung liegt auf der Clientseite, da die entwickelte App eine Client Applikation darstellt. Deswegen wurden spezifische Kriterien der Fachstudie zu einer REST Server Applikation gestrichen. Beispielsweise wurde der gesamte Kriterienblock über Ressourcentypen [7] weggelassen, da es für die clientseitige Verarbeitung irrelevant ist, welche Ressourcentypen serverseitig implementiert werden können.

Entwicklungskultur rund um die Frameworks:

- Unter welcher Lizenz steht das Projekt zur Verfügung?
- Existiert eine aktive Community?
- Ist eine Dokumentation des Codes vorhanden? (Schnittstellenbeschreibung, JavaDoc)
- Gibt es Hilfestellung für Entwicklung? (Tutorial, Codebeispiele)

Implementierung der REST-Frameworks:

- Wie aufwendig ist es das Framework ins Projekt einzubinden?
- Welche HTTP-Methoden werden unterstützt? (GET, POST, PUT, DELETE etc.)
- Gibt es Möglichkeiten den HTTP-Header zu verändern oder zu erweitern?
- Welche Medientypen werden unterstützt? (JSON, HTML, XML etc.)
- Kann die URL zum Abfragen von Ressourcen dynamisch verändert werden? (z.B. über Parameter steuern)
- Gibt es eine Möglichkeit für asynchronen Nachrichtenaustausch?
- Wird das HATEOAS-Konzept* unterstützt?
- Wird ein Error-Handling unterstützt?

Performance und benötigte Speicherplatz der Frameworks:

- Wie stark wird die CPU belastet?
- Wie viel RAM wird benötigt?
- Wie schnell erfolgt die Abwicklung einzelner Requests (GET, POST)?
- Wie groß ist die erzeugte .apk-Datei?

Erweiterte Technische Fähigkeiten der Frameworks:

- Wie wird Sicherheit gehandhabt? (Authentifizierung)
- Werden andere Protokolle fernab von HTTP unterstützt?
- Unterstützt das Framework die Entwicklung von Server Applikationen?
- Bietet das Framework zusätzliche Dienst fernab der REST-Kommunikation an?
- Wird transaktionales Verhalten vom Framework unterstützt? (ACID**-Eigenschaften)

* Das **HATEOAS**-Konzept wird in Kapitel 4 genauer beschrieben.

** **ACID** steht für Atomicity, Consistency, Isolation und Durability. Dieses Konzept beschreibt, dass alle Daten die während einer Transaktion verwendet werden gesperrt sind und sich nicht ändern dürfen, solange bis die Transaktion Committed wird oder ein Rollback durchgeführt wird. Das Einhalten dieser Eigenschaften ist wichtig, da die Kommunikation zu Servern über zustandslose REST-Schnittstellen abgewickelt wird. Durch die Zustandslosigkeit der Anfragen kann es bei Fehlern schnell zu einer Dateninkonsistenz auf dem Server kommen [8].

2 State of the Art

Um Rest Frameworks für die Evaluierung zu finden, wurde eine Technologierecherche durchgeführt. Dabei konnten folgende Projekte gefunden werden, welche eine REST-Anbindung für Android unterstützen:

- Resty (<http://beders.github.io/Resty/Resty/Overview.html>)
- Retrofit (<http://square.github.io/retrofit/>)
- RESTlet (<http://restlet.com/>)
- Spring for Android (<http://projects.spring.io/spring-android/>)
- CRest (<http://crest.codegist.org/index.html>)
- RESTeasy Mobile (<http://resteasy.jboss.org/>)
- RESTDroid (<http://pcreations.fr/me/restdroid-resource-oriented-rest-client-for-android>)
- Jersey (<https://jersey.java.net/>)

Es würde den Rahmen der Bachelorarbeit überschreiten, all diese gefundenen REST Frameworks zu evaluieren. Daher wurde einer Vorstudie gemacht, aufgrund derer die drei populärsten und den Anforderungen adäquatesten Frameworks ausgewählt wurden.

Die Popularität eines Frameworks gibt eine gewisse Auskunft über die Qualität, da für diese Frameworks oft besserer Support in Form von Dokumentation, sowie eine größere und aktivere Community zur Verfügung steht. Eine Studie von Chris Parnin[9] beschäftigen sich damit, wie “Crowd documentation“ beispielsweise auf Question and Answer (Q&A) Webseiten, die Hilfestellung zu verschiedenen Frameworks beeinflusst. Verwenden viele Entwickler ein Framework, sind dadurch mehr Fragen auf Q&A Webseiten vorhanden und dadurch können mögliche Fragen besser beantwortet werden. Durch eine Erhebung der Anzahl von Fragen auf Stack Overflow¹ und der Stars auf GitHub² wurden Rückschlüsse auf die Popularität der einzelnen Frameworks gezogen.

In dem Artikel “How to identify a strong open source project”[10] werden verschiedene Indikatoren erhoben, welche Rückschlüsse auf eine solide und gute Entwicklung eines Frameworks geben. Deswegen wurden zusätzlich noch verschiedene Aktivitäten auf GitHub verglichen, wie Datum des letzten Commits oder Anzahl der Commits.

¹ <http://stackoverflow.com/>

² <https://github.com/>

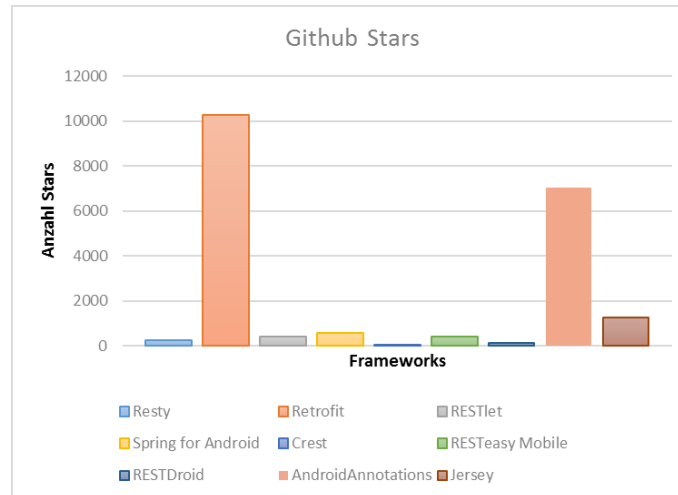


Abbildung 2.1: Github Stars, aktualisiert am 01.03.2016

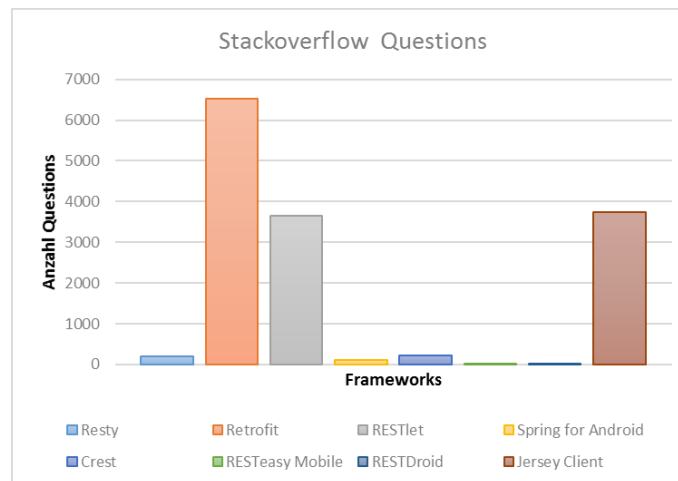


Abbildung 2.2: Stackoverflow Questions, aktualisiert am 01.03.2016

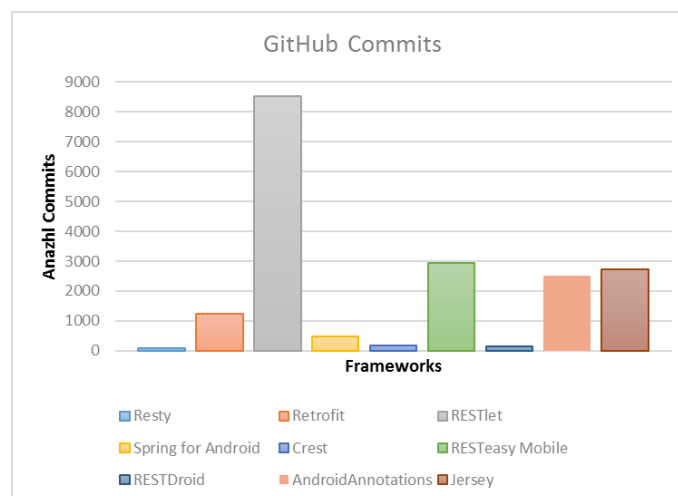


Abbildung 2.3: Github Commits, aktualisiert am 01.03.2016

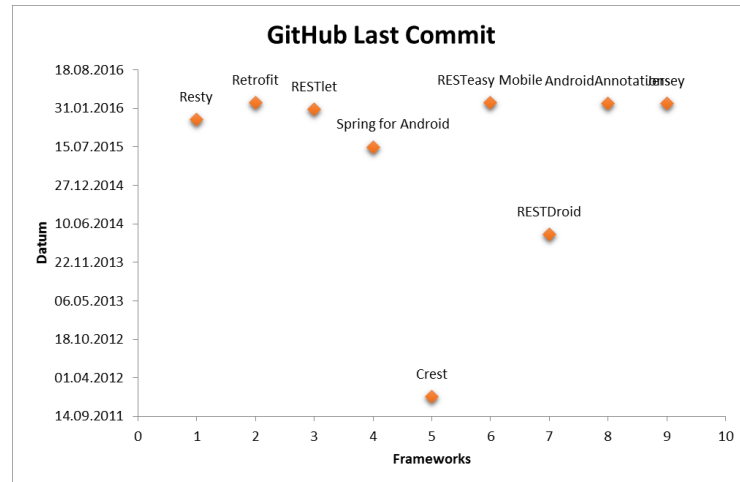


Abbildung 2.4: GitHub Last Commit, aktualisiert am 01.03.2016

Aufgrund der Vorstudie werden folgende REST-Frameworks evaluiert und miteinander verglichen:

- AndroidAnnotations
- Jersey
- Retrofit
- Spring for Android

3 Android

3.1 Überblick

Android ist ein Betriebssystem, welches primär für Smartphones und Tablets konzeptioniert ist. Das Betriebssystem basiert auf einem Linux Kernel und wird von der Open Handset Alliance (gegründet von Google) entwickelt [11]. Android ist eine freie Software und das am schnellsten wachsende mobile Betriebssystem. Der Marktanteil von Android liegt seit 2014 bei über 80% und soll sich auch in den darauffolgenden Jahren bei dieser Prozentzahl halten, wie in der Abbildung 3.1 zu sehen ist.

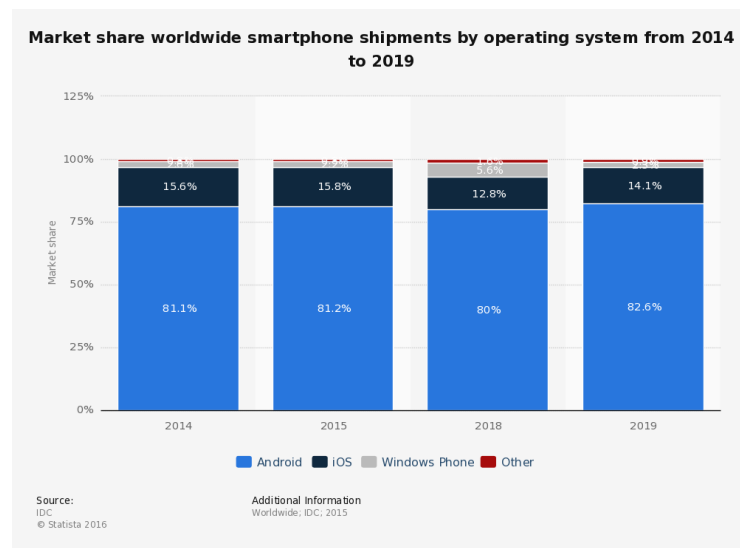


Abbildung 3.1: Marktanteil von mobilen Betriebssystemen [12]

Durch die quelloffene Struktur des Betriebssystems ist Android bei vielen Konsumenten und Entwicklern sehr beliebt, wodurch viele Unternehmen ihre mobilen Applikationen auf dieses Betriebssystem ausrichten. Gemäß einer Vorhersage von IDC, wird Android zwar eine gewisse Prozentzahl an das Windows Phone Betriebssystem verlieren, aber weiterhin der Marktführer bleiben [12]. Auf Grund dessen wurde die Evaluierung der REST Frameworks auf Android ausgerichtet, um auch in den folgenden Jahren einen hohen Nutzerkreis erreichen zu können.

3.2 Architektur

Das Android Betriebssystem ist ein Stack von Software Komponenten, welche typischerweise in vier Bereiche gegliedert werden (vgl. 3.2). Diese Bereiche sind der Linux Kernel, die Native Bibliotheken, die Laufzeitumgebung, das Application-Framework und die Applikationen selbst [13].

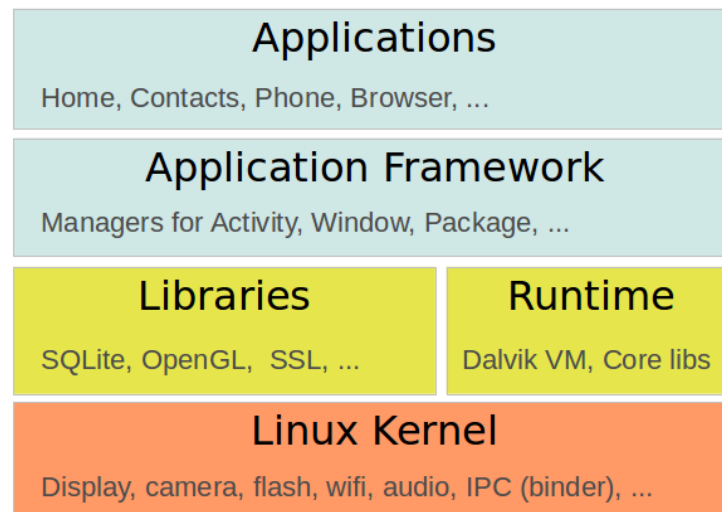


Abbildung 3.2: Android Architektur [13]

Beschreibung der Software Komponenten der Android Architektur [11], [13]:

- **Applikationen**

Die Applikationen stellen die oberste Schicht der Android Architektur dar. Einige Applikationen sind bereits auf jedem Smartphone vorinstalliert, wie beispielsweise ein SMS Client, ein Browser oder ein Kontaktmanager. Software Entwickler können ihre eigenen Applikationen schreiben und diese auf dem Smartphone installieren.

Die zu entwickelnde App, sowie die zu evaluierenden REST-Frameworks befinden sich auf dieser Schicht - das REST-Framework und die Beispielimplementierung werden gemeinsam in eine .apk-Datei zusammengefasst.

- **Application-Framework**

In der Application-Framework-Schicht befinden sich zahlreiche Java-Bibliotheken und Dienste, auf welche Software Entwickler bei der Applikationserstellung Zugriff haben. Wichtige Dienste sind dabei der Activity Manager, der Resource Manager oder der Content Manager.

- **Native Bibliotheken**

Die Native Bibliotheken stellen zahlreiche Funktionen für die Application-Framework-Schicht zur Verfügung, wie Grafik-Rendering oder Web-Browsing. Alle diese Bibliotheken sind in C oder C++ geschrieben und werden durch Java Interfaces aufgerufen, bei der Entwicklung von Applikationen.

- **Runtime**

Die Laufzeitumgebung besteht aus der Dalvik Virtual Machine (VM) und den Java Kernbibliotheken. Die Dalvik VM ist eine Java Virtual Machine, welche speziell für Android entwickelt und optimiert wurde. Durch die Dalvik VM kann jede Applikation in einem eigenen Prozess ausgeführt werden, in einer eigenen Instanz der Dalvik VM.

Durch die Java Kernbibliotheken in der Laufzeitumgebung können Software Entwickler Android Applikationen mithilfe der Programmiersprache Java entwickeln.

- **Linux Kernel**

Der Linux Kernel stellt die unterste Schicht der Android Architektur dar, welcher leicht von Google abgeändert wurde. Der Kernel ist dabei die Schnittstelle zur Geräte Hardware (Kamera, Display etc.) und ist gleichzeitig für die Speicher- und Prozessverwaltung verantwortlich.

3.3 Cross Compiling

Um eine Applikation auf Android ausführen zu können, muss eine .apk-Datei erstellt werden. Dazu wird als erstes eine .java-Datei vom Entwickler erstellt, welche den Quellcode der Applikation enthält. Danach wird mit einem Java-Compiler der Bytecode in Form von .class-Dateien erstellt. Dieser Bytecode wird mit dem dx-Tool aus dem Android SDK in eine .dex-Datei (Dalvik Executable) umgewandelt. Den Bytecode, welche die Dalvik VM ausführt, ist daher kein Java-Bytecode mehr, sondern Dalvik-Bytecode. Dieser Vorgang wird auch als Cross Compiling bezeichnet. Des Weiteren werden mehrere .class-Dateien in eine .dex-Datei zusammengefasst, um Speicherplatz zu sparen. Die .dex-Dateien werden zusammen mit einem Manifest in eine .apk-Datei verpackt. Diese .apk-Datei wird dann auf das Smartphone übertragen und installiert, wodurch die App ausgeführt werden kann [14].

3.4 Android und Java

Die meisten Android Applikationen werden in Java geschrieben und haben als Grundlage die Java 6 Standard Edition (SE). Einige Java 7 Funktionalitäten werden ab der Android Versionen 4.4 (KitKat) unterstützt, davor muss darauf geachtet werden, dass keine spezifischen Funktionen von Java 7 verwendet werden [15]. Dabei unterstützt die Android Java Application Programming Interface (API) einen Großteil der packages, welche in der Java SE Bibliothek vorhanden sind. Einige packages wurden aber weggelassen, da sie auf einer mobilen Plattform keinen Sinn machen [16], wie etwa das Drucken (javax.print). Jedoch wurde, um den Entwicklern die Arbeit zu erleichtern, zusätzlich einige Drittanbieter Bibliotheken hinzugefügt, beispielsweise die Apache HttpComponents Bibliothek (org.apache.commons.httpclient) [17].

Dadurch, dass Android nicht alle Java SE Funktionen der neuesten Version unterstützt, musste bei der Auswahl der Frameworks darauf geachtet werden, dass diese Java 6 kompatibel sind und keine ausgeklammerten packages verwenden.

3.5 Android Versionen

Android veröffentlicht in regelmäßigen Abständen neue Versionen, wie man an den zahlreichen Versionsnummern erkennen kann. Es sind vor allem jene Android-Versionsnummer wichtig, bei denen sich das API-Level ändert. Denn das API-Level bestimmt, welches Geräte eine App ausführen kann und welches nicht. Normalerweise soll eine App von möglichst vielen Geräten ausgeführt werden können. Deswegen sollte ein API-Level anvisiert werden, das so klein wie möglich ist [18].

Wie aus der Abbildung 3.3 zu entnehmen ist, nützen nicht mehr viele Besitzer eines Android-Gerätes die älteren Versionen 2.2 bis 4.0.3. Es nützen aber auch nicht viele Smartphone-Besitzer die aktuellste Android Version 6.0. Die meisten verwenden eine Version zwischen 4.1 und 5.1. Wie aus der Abbildung zu erkennen ist, muss bei der Android-Entwicklung immer einen Kompromiss eingegangen werden. Sollen neuere Funktionen der jüngeren Plattform-Versionen eingesetzt werden, muss in Kauf genommen werden, dass die App mit älteren Geräten nicht kompatibel ist. Soll die App auf so vielen Geräten wie möglich laufen, muss auf die Funktionen der jüngeren Plattform-Versionen verzichtet werden. Da die Plattform kontinuierlich weiterentwickelt wird und die Geräte Systemupdates erhalten, sollte die Entscheidung bezüglich der Versionsnummer in regelmäßigen Abständen überprüft werden [18].

| Version | Codename | API | Distribution |
|------------------|-----------------------|-----|--------------|
| 2.2 | Froyo | 8 | 0.1% |
| 2.3.3 - 2.3.7 | Gingerbread | 10 | 2.7% |
| 4.0.3 - 4.0.4 | Ice Cream Sandwich | 15 | 2.5% |
| 4.1.x | Jelly Bean | 16 | 8.8% |
| 4.2.x | | 17 | 11.7% |
| 4.3 | | 18 | 3.4% |
| 4.4 | KitKat | 19 | 35.5% |
| 5.0 | Lollipop | 21 | 17.0% |
| 5.1 | | 22 | 17.1% |
| 6.0 | Marshmallow | 23 | 1.2% |

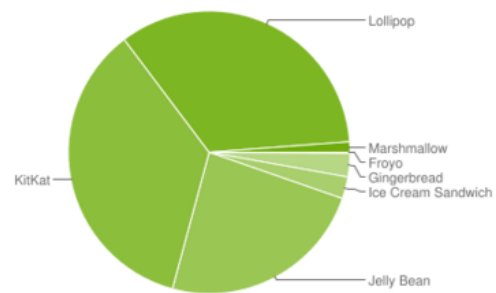


Abbildung 3.3: Android Plattform Versionen [19]

Die Beispielimplementierung ist kompatibel mit der Android Version 5.0 und höher, da für das Anzeigen der Daten eine Funktionalität benötigt wird, die erst in der API Version 21 enthalten ist. Die App kann dadurch auf mehr als einem Drittel der vertriebenen Geräte ausgeführt werden. Wenn ältere Geräte Systemupdates erhalten, erhöht sich diese Anzahl entsprechend.

4 Rest

Der Architekturstil Representational State Transfer (REST) wurde erstmals im Jahr 2000 in der Dissertation von Roy Fielding vorgestellt. REST beschreibt dabei ein Konzept, dass die Prinzipien des World Wide Web zusammenfasst. Roy Fielding abstrahierte sich dabei von konkreten Architekturen wie dem Hypertext Transfer Protocol (HTTP) oder Uniform Resource Identifier (URI). Er legte nur Kernprinzipien fest, die mit unterschiedlichen Protokollen umgesetzt werden können. Zum Beispiel wie Ressourcen im WEB identifiziert oder adressiert werden [20].

Es werden dabei folgende 5 Kernprinzipien unterschieden [21]:

1. Ressourcen mit eindeutiger Identifikation

Durch einen global definierten Namensraum wird sichergestellt, dass Ressourcen weltweit eindeutig identifiziert werden. Im Web heißt dieses Konzept für die Vergabe von IDs, *Uniform Resource Identifier* oder kurz URI.

2. Hypermedia

Mithilfe dieses Konzeptes ist es möglich andere Ressourcen zu referenzieren, um beispielsweise an zusätzliche Informationen zu gelangen. Ein weiterer wichtiger Aspekt ist die Möglichkeit die Applikation durch Links zu steuern. Ein Server kann dem Client über Hypermedia-Elemente mitteilen, welche Aktion er als Nächstes auszuführen hat - indem der Client einen Link *folgt*. Dies wird auch oft als das Hypermedia as the Engine of Application State (HATEOAS)-Konzept bezeichnet.

3. Standardmethoden

Jede Ressource unterstützt den gleichen Satz an Methoden, mit dem diese verarbeitet werden können. Bei HTTP zählen dazu folgende:

- GET, für die Darstellung von Ressourcen
- POST, für das Erstellen einer Ressource
- PUT, für das Aktualisieren einer Ressource
- DELETE, für das Löschen einer Ressource
- HEAD, um Metadaten einer Ressource Abzurufen
- OPTION, um die unterstützten Methoden einer Ressource zu erhalten

4. Unterschiedliche Repräsentationen

HTTP verfolgte einen Ansatz zur Trennung der Verantwortlichkeiten, für Daten und Operationen. Ein Client der ein bestimmtes Dateiformat verarbeiten kann, ist in der Lage jede Ressource mit diesem Format zu verarbeiten, da die Operationen dafür dieselben sind.

5. Statuslose Kommunikation

Serverseitig wird der Zustand des Clients nicht gespeichert. Der aktuelle Zustand muss vollständig auf Seiten des Clients abgespeichert werden und bei Requests müssen die nötigen Informationen an den Server übermittelt werden.

5 Vorstellung der Frameworks

In diesem Kapitel werden die einzelnen Frameworks vorgestellt. Zuerst wird ein kurzer allgemeiner Überblick gegeben, bevor genauer auf die Funktionsweise der Frameworks eingegangen wird.

Um die Frameworks beschreiben und vergleichen zu können, wurde mit jedem Framework eine eigene App entwickelt. Diese Apps unterscheiden sich dabei nur in der Form der REST Kommunikation, welche auf Basis des jeweiligen Frameworks umgesetzt wurde. Als Build-Tool kommt dabei Gradle zum Einsatz. Gradle ist ein auf Java basierendes Tool, zur Build-Automatisierung und zum Build-Management. Es dient daher der Automatisierung von Builds, Tests und Deployments. Gradle nutzt für einen einfachen Build hauptsächlich die `build.gradle` Datei, in welcher alle Tasks und Abhängigkeiten eines Projekts definiert sind.

5.1 Retrofit

5.1.1 Allgemein

Retrofit ist ein typischer HTTP Client für Android und Java, welcher von Square Open Source entwickelt wurde und aktuell in der Version 2.0 ausgeliefert wird [22]. Das Framework baut auf OkHttp auf, welches die Kommunikation auf der Netzwerkebene übernimmt [23]. Retrofit sagt über sich selbst:

“Retrofit turns your HTTP API into a Java interface.”, [22, Webseite von Retrofit]

Mithilfe von Annotation bei den Interface Methoden wird angegeben wie Request zu verarbeiten sind. Daher muss jede Interface-Methode eine HTTP Annotation besitzen, die angibt welche Request Methode zu verwenden ist [22]. Es stehen dabei fünf built-in HTTP Methoden zur Auswahl GET, POST, PUT, DELETE und HEAD.

Standardmäßig kann Retrofit nur `ResponseBody` und `RequestBody` von OkHttp serialisieren und deserialisieren. Durch das hinzufügen von Konvertern ist es jedoch möglich, weitere Formate wie JavaScript Object Notation (JSON) oder Extensible Markup Language (XML) zum Übertragen von Daten zu verwenden. Seit der Veröffentlichung von Retrofit in der Version 2, werden auch noch zusätzliche Parser zur Serialisierung und Deserialisierung von JSON-Daten unterstützt. In der Vergangenheit wurde nur die GSON Bibliothek unterstützt, welche daher die häufig verwendete Bibliothek zum Parsen von JSON-Daten darstellt [24]. Im Rahmen der Bachelorarbeit wird aber die Jackson Bibliothek zum Übertragen von Daten verwendet, da diese am schnellsten Daten serialisiert und deserialisiert [25].

Die Serverschnittstellen werden durch spezielle Annotationen bei den Interface Methoden spezifiziert, die Details über Parameter und die verwendete Request Methode beinhalten. Es stehen unter anderem folgende Annotationen zur Verfügung:

- @GET, @POST, @PUT, @DELETE

Diese Annotationen geben an, welche Request Methode zu verwenden ist.

- @Path

Durch diese Annotation ist es möglich, die URL des Endpoints dynamisch zu konfigurieren. Dabei wird ein bestimmter Teil in der URL der durch Klammern gekennzeichnet wurde, durch den dazugehörenden Parameter ersetzt (siehe Listing ??, Zeile 10).

- @Body

Jeder Parameter der über diese Annotation verfügt, wird in den Body der Request Anfrage eingefügt - nachdem das dazugehörende Java Objekt serialisiert wurde (Listing ??, Zeile 13).

5.1.2 Funktionsweise

Um Retrofit verwenden zu können, müssen die Abhängigkeiten aus dem Listing 5.1 in die `build.gradle` Datei des Projektes eingefügt werden.

```
1 dependencies {  
2     compile 'com.squareup.retrofit:retrofit:2.0.0-beta2'  
3     compile 'com.squareup.retrofit:converter-jackson:2.0.0-beta2'  
4     compile 'com.squareup.okhttp:okhttp:2.4.0'  
5     compile 'com.fasterxml.jackson.core:jackson-core:2.6.0'  
6     compile 'com.fasterxml.jackson.core:jackson-annotations:2.6.0'  
7 }
```

Listing 5.1: Retrofit build.gradle

Um Requests zur Server Schnittstelle versenden zu können, muss die Retrofit Builder Klasse verwendet werden, welche auch die Basis URL der Server REST-Schnittstelle spezifiziert (siehe Listing 5.2).

```
1 private static final String URL = "https://revex.inso.tuwien.ac.at/api/";  
2  
3 public static <S> S createService(Class<S> serviceClass) {  
4  
5     Retrofit retrofit = new Retrofit.Builder()  
6         .baseUrl(URL)  
7         .addConverterFactory(GsonConverterFactory.create())  
8         .build();  
9  
10    return retrofit.create(serviceClass);  
11 }
```

Listing 5.2: Retrofit Builder

Ist ein return Wert bei einem Request vorhanden, ist dies immer ein parametrisiertes `Call<T>` Objekt, beispielsweise `Call <PowerPlant>` (siehe Listing 5.3). Wird kein typenspezifischer Response benötigt oder erwartet, kann als return Wert `Call<Response>` angegeben werden.

```
1 public interface PowerPlantService {  
2  
3     @GET("powerplants")  
4     public Call<List<PowerPlant>> getPowerPlants();  
5  
6     @POST("powerplants")  
7     public Call<PowerPlant> createPowerPlant(@Body PowerPlant  
8         ↪ powerPlant);  
9  
10    @DELETE("powerplants/{id}")  
11    public Call<PowerPlant> deletePowerPlantById(@Path("id") int id);  
12  
13    @PUT("powerplants/{id}")  
14    public Call<PowerPlant> updatePowerPlant(@Path("id") int id, @Body  
15        ↪ PowerPlant powerPlant);  
16 }
```

Listing 5.3: Auszug aus dem PowerPlantService

Sollen bei einem Request noch zusätzlich Daten in den Header eingefügt oder dieser manipuliert werden, ist dies durch das Interface `Interceptor` möglich (siehe Listing 5.4). Dadurch kann zum Beispiel vor jeder Anfrage an den Server, ein Token hinzugefügt werden, welches vom Server zur Authentifizierung verwendet wird.

```
1 public static <S> S createServiceWithAuthToken(Class<S> serviceClass  
2     ↪ , final AuthToken token) {  
3     Interceptor interceptor = new Interceptor() {  
4         @Override  
5         public Response intercept(Chain chain) throws IOException {  
6             Request newRequest = chain.request().newBuilder()  
7                 .addHeader("Accept", "application/json")  
8                 .addHeader("X-Auth-Token", token.getToken())  
9                 .build();  
10            return chain.proceed(newRequest);  
11        }  
12    };  
13  
14    OkHttpClient client = new OkHttpClient();  
15    client.interceptors().add(interceptor);  
16  
17    Retrofit retrofit = new Retrofit.Builder()  
18        .client(client)  
19        .baseUrl(BASE_URL)  
20        .addConverterFactory(GsonConverterFactory.create())  
21        .build();  
22  
23    return retrofit.create(serviceClass);  
24 }
```

Listing 5.4: Hinzufügen des Tokens, für gültigen Login

5.2 Spring for Android

5.2.1 Allgemein

Spring gliedert sich in zahlreiche Projekte, mit dem Ziel Java Entwicklung zu vereinfachen. Die Idee zum Spring Framework hatte Rod Johnson im Jahr 2003. An den zahlreichen Open Source Projekten von Spring beteiligen sich weltweit eine große Anzahl von Entwicklern [26]. Spring for Android ist ein Projekt von Spring, dass die Entwicklung von Andorid-Apps vereinfachen soll. Aktuell wird das Projekt in der Version 2.0 ausgeliefert. Es stellt dabei ausgewählte Komponenten von Spring, gebündelt für Android zur Verfügung [27]. Dies sind im speziellen

- ein Rest Client (RestTemplate) und
- Authentifikations Unterstützung für Security APIs (OAuth).

Spring for Android unterstützt dabei keine Dependency Injection, eine der Kernfunktionen des Core Spring Frameworks. Die Bibliothek unterstützt den Entwickler daher nicht, bei der Umsetzung einer möglichst losen Kopplung in seiner Applikation [28].

Grundbaustein des Frameworks ist die Klasse `RestTemplate`, welche Teil des im Jahr 2009 erschienen Framework “Spring for MVC” ist. Diese Klasse ermöglicht Java-Entwicklern eine High-Level-Abstraktion von untergeordneten Java-APIs, wie des HTTP Clients. `RestTemplate` for Android unterstützt dabei auch die gzip Komprimierung und die Übertragung von JSON und XML Daten, indem Plain Old Java Object (POJO) Objekte – einfache Java-Objekte, die keine Business Logik enthalten [29] – automatisiert konvertiert werden [28].

Die Klasse `RestTemplate` ist für den clientseitigen Zugriff auf einen RESTful Service zuständig. Das Verhalten kann durch Callback Methoden und durch konfigurieren des `HttpMessageConverter` angepasst werden. Der `HttpMessageConverter` wird dabei verwendet um einen HTTP Request Body zu erstellen oder den Response in Java Objekte zu konvertieren. Die Klasse `MappingJackson2HttpMessageConverter` ist dabei ein Message Converter mit dem JSON Daten verarbeitet werden können.

`RestTemplate` stellt verschiedene Methoden zur Verfügung, um alle gängigen HTTP Methoden verwenden zu können, wie GET, POST, PUT, DELETE, HEAD und OPTION [30].

5.2.2 Funktionsweise

Um Spring for Android verwenden zu können, muss die Abhängigkeiten aus dem Listing 5.5 in die `build.gradle` Datei des Projektes eingefügt werden.

```
1 dependencies {  
2     compile 'org.springframework.android:spring-android-rest-template:2.0.0.M1'  
3     compile 'com.fasterxml.jackson.core:jackson-core:2.6.0'  
4     compile 'com.fasterxml.jackson.core:jackson-annotations:2.6.0'  
5 }
```

Listing 5.5: Spring for Android build.gradle

Wie bereits oben beschrieben wird die Klasse `RestTemplate` dazu verwendet um HTTP Anfragen zusammenzubauen, um mit der Server REST-Schnittstelle kommunizieren zu können. Im Listing 5.6 folgt ein kurzes Beispiel eines GET Requests, um vom Server alle vorhandenen Kraftwerke zu erhalten.

```
1 public static List<PowerPlant> getPowerPlants () {  
2     RestTemplate restTemplate = new RestTemplate(  
3         new BufferingClientHttpRequestFactory(  
4             new SimpleClientHttpRequestFactory() ));  
5  
6     restTemplate.getMessageConverters().  
7         add(new MappingJackson2HttpMessageConverter());  
8  
9     URI url = UriComponentsBuilder.  
10         fromUriString(ServiceGenerator.BASE_URL).  
11         path("/powerplants").build().toUri();  
12  
13     HttpHeaders headers = new HttpHeaders();  
14     headers.set("X-Auth-Token",  
15         UtilitiesManager.getInstance().getAuthToken().getToken());  
16  
17     headers.setContentType(MediaType.APPLICATION_JSON);  
18  
19     HttpEntity entity = new HttpEntity(headers);  
20  
21     HttpEntity<PowerPlant[]> response = restTemplate.  
22         exchange(url, HttpMethod.GET, entity, PowerPlant[].class);  
23  
24     return Arrays.asList(response.getBody());  
25 }
```

Listing 5.6: GET Request um alle Kraftwerke zu erhalten

Wie man anhand des Beispiels sehen kann, erfolgt eine automatisierte Umwandlung der erhaltenen JSON Daten zu einem Java Objekt. Für einen GET Request muss daher nur der Endpunkt des Servers und der Typ der Response Daten spezifiziert werden. Um HTTP-Header Felder hinzuzufügen wird ein `HttpHeaders` Objekt benötigt, diesem können dann die HTTP Felder über die `set`-Methode hinzugefügt werden. Dadurch ist es möglich vor jeder Anfrage das X-Auth-Token hinzuzufügen, welches vom Server für die Authentifizierung verwendet wird.

Um die Daten zu einem speziellen Kraftwerk abzufragen, ist es nötig die URL dynamisch zur Laufzeit zu verändern. Dabei wird das abzufragende Kraftwerk über den Parameter `id` (siehe Listing 5.7, Zeile 3) spezifiziert.

```
1 URI url = UriComponentsBuilder.  
2     fromUriString(ServiceGenerator.BASE_URL).  
3     path("/powerplants/" + id).  
4     build().toUri();  
5  
6 HttpEntity entity = new HttpEntity(headers);  
7  
8 HttpEntity<PowerPlant> response = restTemplate.  
9     exchange(url, HttpMethod.GET, entity, PowerPlant.class);
```

Listing 5.7: Abfrage der Daten zu einem speziellen Kraftwerke

Ein POST Request mit Spring for Android kann aus dem Listing 5.8 entnommen werden.

```
1 HttpHeaders headers = new HttpHeaders();
2
3 headers.set("X-Auth-Token",
4             UtilitiesManager.getInstance().getAuthToken().getToken());
5
6 headers.setContentType(MediaType.APPLICATION_JSON);
7
8 HttpEntity<PowerPlant> requestEntity =
9     new HttpEntity<>(powerPlant, headers);
10
11 HttpEntity<PowerPlant> response = restTemplate.
12     exchange(url, HttpMethod.POST, requestEntity, PowerPlant.class);
```

Listing 5.8: POST Request um ein neues Kraftwerke anzulegen

Im Objekt `requestEntity` sind dabei die Daten enthalten, die im Body des HTTP Requests übertragen werden.

5.3 Jersey

5.3.1 Allgemein

In Java gibt es mit JAX-RS einen Standard zum Implementieren von REST-basierten Webservices. Dieser wurde in der JSR-311 "JAX-RS: The Java API for RESTful Web Services"[31] genauer spezifiziert und ist deshalb ein offizieller Teil von Java. Das Jersey Framework ist dabei eine robuste Open-Source Referenz Implementierung der Spezifikation [32].

Die JAX-RS-Client-API Implementierung von Jersey kann mit einem beliebigen Web-Service, der auf das HTTP-Protokoll aufsetzt, kommunizieren. Der Jersey-Client wird zurzeit in der Version 2.22.1 ausgeliefert. Der serverseitige Endpoint muss nicht unbedingt mit JAX-RS implementiert sein, jedoch muss der HTTP Standard eingehalten werden.

Den Grundbaustein der Jersey-Client Implementierung bildet die Klasse `ClientBuilder`. Diese wird dazu verwendet neue `Client`-Instanzen anzulegen, über welche Requests zum Server versendet werden. Mithilfe der `ClientBuilder`-Klasse können auch zusätzliche Eigenschaften für den `Client` definiert werden, wie zum Beispiel die SSL Transport Konfiguration. Des weiteren kann ein `Client` mithilfe der `ClientConfig`-Klasse speziell konfiguriert werden, diese Konfiguration wird bei der Erstellung des `Client` einer Factory Methode übergeben. Mithilfe der `ClientConfig`-Klasse können Provider registriert oder Filter hinzugefügt werden. Ein oft verwendeter Filter ist zum Beispiel der `LoggingFilter`. Dieser ist ein Protokollierungsfilter, der die Kommunikation zwischen Server und Client aufzeichnet und dessen Aufzeichnung später für Debugging Zwecke genutzt werden kann [33].

Nachdem eine `Client`-Instanz erfolgreich erstellt wurde, kann mit dieser Instanz ein `WebTarget` Objekt erzeugt werden, welches den Endpoint zum Server spezifiziert. Um nun erfolgreich einen HTTP Request abzusetzen, muss ein `Invocation.Builder` Objekt erzeugt werden. Dieses Objekt wird dazu genutzt um die Anfrage zum Server genauer zu konfigurieren und abzusenden. Es gibt sowohl Methoden um den Media Type des Request und Response anzugeben, die HTTP Methode zu definieren und den HTTP Header genauer zu spezifizieren [33].

Die Client API von Jersey unterstützt alle gängigen HTTP Methoden, welche GET, POST, PUT, DELETE, HEAD und OPTION sind [34].

5.3.2 Verwendung unter Android

Jersey kann erst ab der Version 2.16 auf Android verwendet werden. Davor war es nicht möglich Jersey auf Android auszuführen, aufgrund der Abhängigkeit des Client Core Moduls zum package `javax.xml.stream`, welches in der Android Java API nicht vorkommt. In der neuen Jersey Version wurden alle JAXB-B Provider Abhängigkeiten in ein eigenes Modul ausgelagert [35]. Fügt man nun die Abhängigkeit zu diesem Modul in das Projekt ein, funktioniert die grundlegende Kommunikation zwischen einem Android Jersey Client und den REST Server.

```
compile 'javax.xml.bind:jaxb-api:2.1'
```

Dennoch ist es noch immer nicht möglich, den Jersey Client ohne Abhängigkeitsproblemen in einer Android Umgebung zu verwenden. Es werden noch immer Abhängigkeiten zu packages benötigt, die in der Android-Umgebung nicht verfügbar sind. Mithilfe der HK2 API ist es möglich Komponenten ohne Vorbereitung aus dem HK2 Service Locator zu entfernen. Der Workaround

aus dem Listing 5.9 hilft Fehler durch fehlende Abhängigkeiten zu unterdrücken und dennoch alle Funktionen des Jersey Clients in der Android Umgebung zu unterstützen [36].

```

1 public static class AndroidFriendlyFeature implements Feature {
2
3     @Override
4     public boolean configure(FeatureContext context) {
5         context.register(new AbstractBinder() {
6
7             @Override
8             protected void configure() {
9                 addUnbindFilter(new Filter() {
10
11                     @Override
12                     public boolean matches(Descriptor d) {
13                         String implClass = d.getImplementation();
14                         return implClass.startsWith(
15                             "org.glassfish.jersey.message.internal.DataSource")
16                             || implClass.startsWith(
17                             "org.glassfish.jersey.message.internal.RenderedImage");
18                     }
19                 });
20             }
21         });
22         return true;
23     }
24 }

```

Listing 5.9: Workaround Jersey Client auf Android

Das Feature für den Workaround muss später noch bei der Client-Instanz registriert werden.

```

client = ClientBuilder.newClient().
    register(AndroidFriendlyFeature.class);

```

5.3.3 Funktionsweise

Hat man alle erforderlichen Abhängigkeiten zum Projekt hinzugefügt, kann ein einfacher GET Request realisiert werden, der ein `PowerPlant` Objekt zurückgibt (siehe Listing 5.10). Der Media Type wird mithilfe der `accept`-Methode spezifiziert, die Art der HTTP Request Methode wird durch `get(PowerPlant.class)` angegeben. Dabei wird durch den Parameter in der Methode der Returntyp des Response festgelegt, in diesem Fall wird genau ein Objekt des Typs `PowerPlant` erwartet.

```

1 public static PowerPlant getPowerPlantById(int id) {
2
3     ClientConfig clientConfig = new ClientConfig().
4         register(JacksonFeature.class);
5
6     Client client = ClientBuilder.newClient(clientConfig).
7         register(AndroidFriendlyFeature.class);
8
9     Invocation.Builder builder = client.

```

```
10         target(BASE_URL).  
11         path("/powerplants/" + id).  
12         request(MediaType.APPLICATION_JSON);  
13  
14     PowerPlant powerPlant = builder.  
15         accept(MediaType.APPLICATION_JSON).  
16         get(PowerPlant.class);  
17  
18     return powerPlant;  
19 }
```

Listing 5.10: GET Request

Im Listing 5.3.3 wird ein POST Request abgesendet, der ein Objekt an den Server im JSON Format übergibt und auch einen Returnwert im JSON Format erwartet.

```
PowerPlant newPowerPlant = builder.  
    accept(MediaType.APPLICATION_JSON).  
    post(Entity.entity(powerPlant, MediaType.APPLICATION_JSON),  
        PowerPlant.class);
```

5.4 AndroidAnnotations

5.4.1 Allgemein

AndroidAnnotations ist ein Projekt das von Pierre-Yves Ricau gestartet wurde. Der Kern des Projektteams besteht aus aktiven und ehemaligen eBusinessInformation Mitarbeitern. Das Unternehmen unterstützt das Open Source Projekt durch seine Mitarbeiter, indem diesen Zeit und Ressourcen zur Verfügung gestellt werden, um an AndroidAnnotations zu arbeiten. Das Projekt wird aktuell in der Version 4.0 ausgeliefert [37].

Das Projekt AndroidAnnotations soll dabei helfen, Codefragmente die sich bei der Android-App Entwicklung wiederholen, zu reduzieren, indem Annotationen verwendet werden. Dadurch verringert sich der zu schreibende und wartende Code, wodurch die Entwicklung beschleunigt und die Wartbarkeit verbessert werden kann [38].

Bei der Kompilierung des Codes werden die verwendeten Annotationen aufgelöst, wodurch es zu keiner Laufzeitbeeinträchtigung durch die Verwendung von AndroidAnnotations kommt. Dies geschieht indem eine Unterklasse jeder einzelnen Aktivität erzeugt wird und die Annotations durch die zugrundeliegenden Codefragmente ersetzt werden. Wird beispielsweise eine `MyActivity` Klasse implementiert, erzeugt AndroidAnnotations die entsprechende `MyActivity_` Unterklasse. Ein kleiner Nachteil dieses Ansatzes ist, dass ein Unterstrich hinter den Namen der einzelnen Aktivitäten in der Manifest-Datei anhängt werden muss. Des weiteren muss beim Starten einer neuen Aktivität auch darauf geachtet werden, dass die entsprechend generierte Unterklasse verwendet wird [39].

Die Rest API von AndroidAnnotations ist dabei ein Wrapper rund um das Spring for Android `RestTemplate` Modul. Die `@Rest`-Annotation über einem Interface ist dabei der Grundbaustein, welche die Schnittstelle genauer spezifiziert. Die Endpoint URL vom Server wird über den `rootUrl` Parameter bei der `@Rest`-Annotation angegeben. Des weiteren muss bei der `@Rest`-Annotation mindestens ein Messagekonverter angegeben werden, welcher dem Spring `HttpMessageConverters` entspricht, da dieser dem `RestTemplate` Modul zur Verfügung gestellt wird. Dadurch werden automatisiert Java Objekte serialisiert und deserialisiert. Sind mehrere Konverter definiert, durchläuft Spring alle, bis der Content-Typ des Requests oder Response verarbeitet werden kann. Es können auch Interceptoren hinzugefügt werden, welche beispielsweise jeden Request protokollieren oder benutzerdefinierte Authentifizierung handhaben [40].

Wichtige Annotationen für die Rest Kommunikation sind [40]:

- `@Get, @Post, @Put, @Delete, @Head, @Options`
Über diese Annotationen wird die Request Methode spezifiziert, die Rest API von AndroidAnnotations unterstützt dabei die selben HTTP Methoden wie Spring for Android.
- `@Path`
Methodenparameter die mit dieser Annotation versehen werden, sind ausdrücklich als Pfadvariablen markiert, sie müssen daher einen entsprechenden Platzhalter in der URL haben.
- `@Body`
Diese Annotation wird dazu verwendet, um Daten in den Body eines Requests einzufügen. Diese Annotation kann nur in Kombination mit den HTTP-Methoden POST, PUT, DELETE oder PATCH verwendet werden. Außerdem ist nur ein `@Body` Parameter pro Request zulässig.

5.4.2 Funktionsweise

Um AndroidAnnotations verwenden zu können, müssen die Abhängigkeiten aus dem Listing 5.11 in die `build.gradle` Datei eingefügt werden.

```
1 apply plugin: 'com.android.application'
2 apply plugin: 'com.neenbedankt.android-apt'
3
4 dependencies {
5     apt "org.androidannotations:androidannotations:4.0-SNAPSHOT"
6     compile "org.androidannotations:androidannotations-api:4.0-SNAPSHOT"
7     compile "org.androidannotations:rest-spring:4.0-SNAPSHOT"
8     compile 'org.springframework.android:spring-android-rest-template:1.0.1.RELEASE'
9     compile 'com.fasterxml.jackson.core:jackson-core:2.6.0'
10    compile 'com.fasterxml.jackson.core:jackson-databind:2.6.0'
11 }
12
13 apt {
14     arguments {
15         androidManifestFile variant.outputs[0].processResources.
16             ↪ manifestFile
17     }
18 }
```

Listing 5.11: AndroidAnnotations build.gradle

Das Android-apt Plugin hilft bei der Verarbeitung und richtigen Auflösung der Annotationen. Beispielsweise muss AndroidAnnotations wissen, wo die dazugehörige Manifest-Datei liegt. Dies wird durch die `variant` angegeben. Durch die Angabe eines Indexes können auch unterschiedliche Manifest-Dateien für verschiedene Anwendungsfälle spezifiziert werden [41].

Die beispielhafte Implementierung eines AndroidAnnotation Rest Client kann man aus folgendem Listing 5.12 entnehmen. Durch Annotationen wird sowohl die Serverschnittstelle, also auch die zu verwendenden HTTP Methoden spezifiziert. Zur Konvertierung der übertragenen Daten im JSON Format wird die Klasse `MappingJackson2HttpMessageConverter` von Jackson API verwendet.

```
1 @Rest(rootUrl = "https://revex.inso.tuwien.ac.at/api",
2     converters = { MappingJackson2HttpMessageConverter.class },
3     interceptors = { HttpBasicAuthenticatorInterceptor.class })
4 public interface PowerPlantService {
5     @Get("/powerplants")
6     List<PowerPlant> getPowerPlants();
7
8     @Post("/powerplants")
9     public PowerPlant createPowerPlant(@Body PowerPlant powerPlant);
10
11     @Delete("/powerplants/{id}")
12     PowerPlant deletePowerPlantById(@Path int id);
13
14     @Put("/powerplants/{id}")
15     PowerPlant updatePowerPlant(@Path int id, @Body PowerPlant pp);
16 }
```

Listing 5.12: Auszug aus dem PowerPlantService Interface

Beim Listing 5.13 wird auch ein Interceptor registriert, der vor jeden Requests das benutzerdefinierte Authentifizierungs-Token in dem Header einfügt.

```
1 public class HttpBasicAuthenticatorInterceptor
2         implements ClientHttpRequestInterceptor {
3
4     @Override
5     public ClientHttpResponse intercept(HttpRequest request,
6         byte[] data, ClientHttpRequestExecution execution)
7         throws IOException {
8
9         request.getHeaders().add("X-Auth-Token",
10             UtilitiesManager.getInstance().getAuthToken().getToken());
11
12         return execution.execute(request, data);
13     }
14 }
```

Listing 5.13: Interceptor für das setzen eines zusätzlichen HTTP Header Feldes

5.4.3 zusätzliche Dienste

Einer der größten Vorteile dieses Frameworks ist, dass zur Verfügung stellen weiterer Dienste. Ein solcher Dienst ist die Dependency Injection (DI). Durch die Verwendung dieses Entwurfsmusters wird die Kopplung zwischen einzelnen Modulen (Objekte, Klassen) reduziert. Dabei bekommen die Module ihre Abhängigkeiten von einer zentralen Komponente zugewiesen [39]. Die anderen getesteten Frameworks beinhalten keine DI-Komponente, können aber mit entsprechenden Bibliotheken um diese Funktionalität erweitert werden.

Die Haupteigenschaften des Projektes sind unter anderem [38]:

- **Dependency Injection**
Injection von Views, System Services oder Ressourcen.
- **einfaches Threading-Modell**
Annotation über Methoden, die angeben, ob die Methode vom UI-Thread oder von einem Hintergrundthread ausgeführt werden soll.
- **Event Binding**
Annotation über Methoden um Events von Views zu behandeln, es werden keine anonymen Listener Klassen mehr benötigt.
- **Rest Client**
Durch erstellen eines Client Interfaces, wird die zugrundeliegende Implementierung generiert.

6 Vergleich

Es folgt nun eine Gegenüberstellung der einzelnen Frameworks, anhand der in Kapitel 1.4 definierten Kriterien. Als erster wird ein genereller Überblick über die einzelnen Fakten zu den Frameworks gegeben und deren Unterschiede beschrieben. Danach wird genauer auf relevanten Kriterien für das Revex Projekt eingegangen und ein persönliches Fazit zu den Frameworks gezogen.

6.1 Gegenüberstellung der Frameworks

Entwicklungskultur rund um die Frameworks

Alle Frameworks werden unter einer Open Source Lizenz zur Verfügung gestellt. Einige Lizenzen beinhalten aber ein Copyleft, was verlangt, dass die entwickelte Software ebenfalls wieder unter der gleichen Lizenz als Open Source Software (OSS) zur Verfügung gestellt wird. Die wichtigste Copyleft-Lizenz ist die GNU General Public License (GPL). Die GPL bestimmt, dass Software, welche eine Framework unter dieser Lizenz nutzt, wiederum nur unter GPL vertrieben werden darf. Die Apache License ist eine Lizenz ohne Copyleft, es ist daher zulässig Frameworks mit dieser Lizenz in proprietärer Software zu verwenden. Die Common Development and Distribution License (CDDL) beinhaltet ein abgeschwächtes Copyleft, lizenzierter Code kann in einem anderen Framework verwendet werden, solange dieser Code nicht auf Dateiebene gemischt wird. Die CDDL verbietet nicht, entwickelte Software unter einer anderen Lizenz zu veröffentlichen. Für die reinen Nutzer von OSS, welche die Frameworks weder weiterentwickeln noch vertreiben wollen, spielen die Unterschiede der verschiedenen OSS-Lizenzen kaum eine Rolle. Will ein Entwickler jedoch seine Arbeitsergebnisse nicht als OSS veröffentlichen, darf er für seine Arbeiten keine OSS Frameworks verwenden, welche mit einem strengen Copyleft lizenziert sind [42]. Die Lizenzen der einzelnen Frameworks könnten aus der Tabelle 6.1 entnommen werden.

OSS wird durch Communities entwickelt, weiterentwickelt und gewartet. Dabei übernimmt die Community zahlreiche Aufgaben, die bei Closed Software vom Hersteller oder Anbieter wahrgenommen werden. In der Regel gibt es zu jeder OSS genau eine Community. User der Software können dabei ein registriertes Mitglied der Community sein und aktiv am Projekt als Entwickler oder Tester mitarbeiten. Die Qualität einer Software hängt dabei maßgeblich von der Aktivität und der Struktur der Community ab. Sind zahlreiche aktive Mitglieder mit unterschiedlichen Rollen am Projekt beteiligt, ist dies eine Grundlage für eine gute Qualität der Software [43]. Alle untersuchten Frameworks weisen eine aktive und starke Community auf.

Um die Anwendung neuer Frameworks zu lernen, lesen 78 Prozent der Entwickler die Dokumentation, 55 Prozent verwenden Codebeispiele und 29 Prozent fragen Kollegen [44]. Dadurch ist es besonders wichtig auf eine aktuelle Dokumentation des Source-Codes Zugriff zu haben. Wird nur eine veraltete Dokumentation zur Verfügung gestellt, kann es passieren dass die entwickelte Software Sicherheitslücken hat oder mehr Zeit für eine korrekte Implementierung benötigt wird [45]. Für alle Frameworks ist eine Dokumentation vorhanden. Die Dokumentation von Retrofit deckt dabei aber nur die grundlegende Funktionsweise ab und ist im Vergleich zu den anderen schwächer. Das Problem hierbei war, dass in der zweiten Jahreshälfte 2015 ein großes Update

des Frameworks während des Evaluierungsprozesses durchgeführt wurde und die vorhandene Dokumentation dadurch teilweise veraltet war. Es konnten auch Codebeispiele für alle Frameworks gefunden werden, diese sind gut erklärt und ermöglichen es eine lauffähige Anwendung zu implementieren. Problematisch war es allerdings die gefundenen Jersey Codebeispiele unter Android zu starten, da keines dieser speziell für Android entwickelt wurde. Es musste zuerst ein Workaround (siehe Kapitel 5.3.2) durchgeführt werden, um die Beispiele auf einem Android Gerät starten zu können.

Implementierung der REST-Frameworks

Einer der relevantesten Aspekte für den Evaluierungsprozess ist die Implementierung und Verwendungsweise der Frameworks. Bis auf Jersey sind alle Frameworks leicht einzubinden und zu benutzen. Für Jersey muss zuerst ein Workaround durchgeführt werden, damit dieses Framework überhaupt unter Android genutzt werden kann. Die genaue Verwendungsweise der Frameworks wird in Kapitel 5 beschrieben.

Es werden von allen Frameworks die gängigsten HTTP-Methoden unterstützt. Retrofit unterstützt dabei als einziges nicht die HTTP-Methode OPTIONS. Es ist auch möglich mit allen Frameworks den HTTP-Header zu erweitern bzw. zu verändern. Des weiteren ist es auch möglich alle möglichen Medientypen zu unterstützen, wenn ein Konverter dafür implementiert und beim Framework registriert wird. Endpoint URLs zum Server können dynamisch während der Laufzeit bei allen Frameworks verändert werden.

Wenn zeitraubende Aktionen ausgeführt werden sollen, wie z.B. das Herunterladen einer großen Datei aus dem Internet, langweilen sich User oft. Denn diese müssen dann warten bis der Request vollständig abgearbeitet wurde und können währenddessen nicht weiterarbeiten. Dies kann aber verhindert werden, indem zeitraubende Aktion im Hintergrund ausgeführt werden - in Form von asynchronen Requests [46]. Mithilfe aller Frameworks können asynchrone Requests modelliert werden.

Das HATEOAS Konzept wird von Jersey und Spring for Android unterstützt. Jedoch muss bei Spring for Android eine eigene Bibliothek "Spring HATEOAS" eingebunden werden, damit eine Linkverfolgung möglich ist. Wenn das HATEOAS Konzept mit AndroidAnnotations verwendet werden soll, muss ebenfalls "Spring HATEOAS" eingebunden werden und die Verwendung ist nur ohne Annotations möglich. Man greift dabei auf die Basis REST Implementierung zurück, welche Spring for Android ist. Retrofit unterstützt das HATEOAS Konzept nicht.

Das registrieren von Error Handlern ist bei allen Frameworks möglich, dadurch können aussagekräftige Fehlermeldungen für den User erzeugt werden. Designrichtlinien für mögliche Geräte geben an, dass es sinnvoll ist, Rückmeldungen für Aktionen des Systems zu geben, wie beispielsweise eine Fehlermeldung, wenn der Server nicht erreichbar ist. Eine solche Rückmeldung sollte für den Benutzer verständlich sein. Zum Beispiel sind die Fehlermeldungen "HTTP404 ERROR" und "Diese Seite konnte nicht gefunden werden" äquivalent, die zweite Fehlermeldung ist aber für einen Großteil der User verständlicher [47].

Performance

Mobile Geräte haben längst Einzug in das Alltagsleben erhalten, viele Nutzer erweitern die Funktionalität der Geräte mit zahlreichen Anwendungen. Jede installierte App auf einen mobilen Endgerät hat Einfluss auf den Energieverbrauch. Je mehr Anwendungen auf eine Endgerät installiert

sind und je öfter diese genutzt werden, desto höher ist der Energieverbrauch und die Betriebszeit sinkt dadurch massiv [48]. Daher ist bei der Entwicklung darauf zu achten, dass der Energieverbrauch so gering wie möglich gehalten wird. Auswirkungen auf den Energieverbrauch eines Smartphones haben sowohl der Bildschirm, die Central Processing Unit (CPU), die Kommunikation über das Mobilfunknetz, als auch das WLAN-Modul. Werden die einzelnen Komponenten so wenig wie möglich beansprucht, ist der Energieverbrauch der Anwendung gering [49].

Umso länger und mehr Daten bei einem Request über das Mobilfunknetz oder das WLAN-Modul übertragen werden, desto mehr Energie wird verbraucht. Es sollte daher darauf geachtet werden, dass die Requestdauer möglichst gering ist und nicht unnötige Daten übertragen werden. Des Weiteren ist die Rechenleistung von Smartphones wesentlich geringer als jene von Notebooks. Daher ist bei der Entwicklung auch darauf zu achten, dass die CPU Beanspruchung der Apps gering gehalten wird, damit das System performant weiterarbeiten kann [50]. Außerdem sollte die benötigte CPU Leistung aus einem weiteren Grund möglichst gering sein, den die Hardware variiert von Smartphone zu Smartphone. Ist die benötigte CPU Leistung gering, ist es möglich die App auf nahezu allen mobilen Geräten zu installieren und zu starten [51].

Für den Performance Vergleich wurden daher sowohl die CPU-Auslastung, als auch die Dauer für GET- und POST Requests (Dauer der Netzwerkkommunikation) der einzelnen Frameworks gegenübergestellt. Um Vergleichswerte zu ermitteln, wurde jede App einzeln in einem Emulator gestartet und die gleiche Abfolge an Funktionsaufrufen durchgeführt.

Die Zeitdauer der GET-Requests bei der Abbildung 6.1 setzt sich aus 11 einzeln abgesetzte GET-Requests zusammen. Diese Requests werden benötigt, um die Details zu einem bestimmten Kraftwerk anzuzeigen. Es werden in der Praxis oft mehrere GET-Request hintereinander abgesetzt, daher wurde die Dauer ermittelt um alle Details zu einem Kraftwerk abzurufen. Es konnte dabei festgestellt werden, dass Retrofit die Daten am schnellsten überträgt, dicht gefolgt von AndroidAnnotations und Jersey am langsamsten.

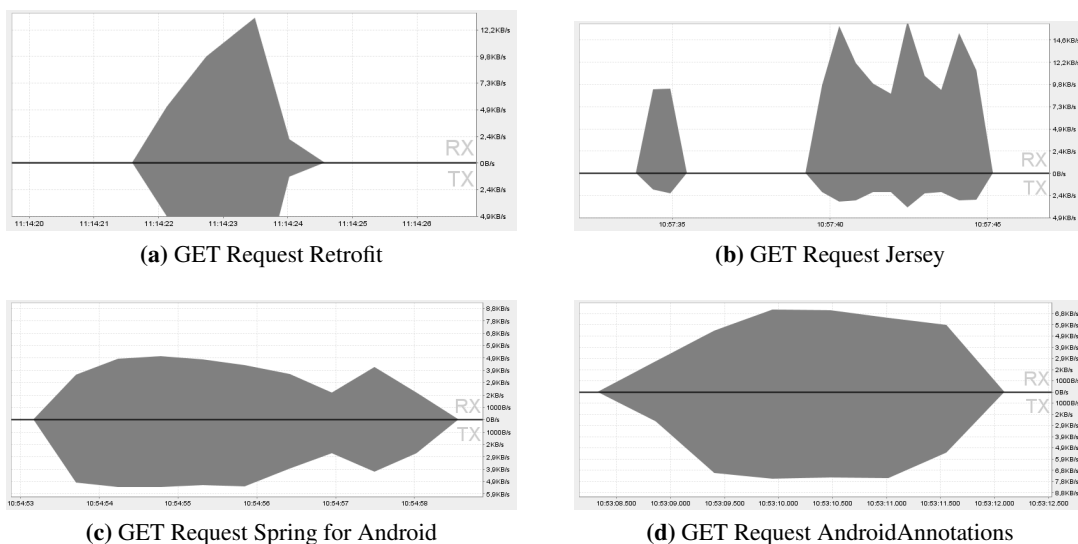


Abbildung 6.1: Zeitmessung der GET Requests

Die Messung der Zeitdauer für einen POST Request ist aus der Abbildung 6.2 zu entnehmen. Dabei wurde die Zeit gemessen bis neu eingegeben Daten, um ein Kraftwerk anzulegen erfolgreich zum Server übermittelt wurden. Es konnte dabei festgestellt werden, dass alle Frameworks circa gleich lange benötigen, um die Daten zu übertragen. Spring for Android ist dabei minimal langsamer als die anderen, dies ist aber zu vernachlässigen, da diese Verzögerung keine Auswirkung auf die Usability für einen User hat [52].

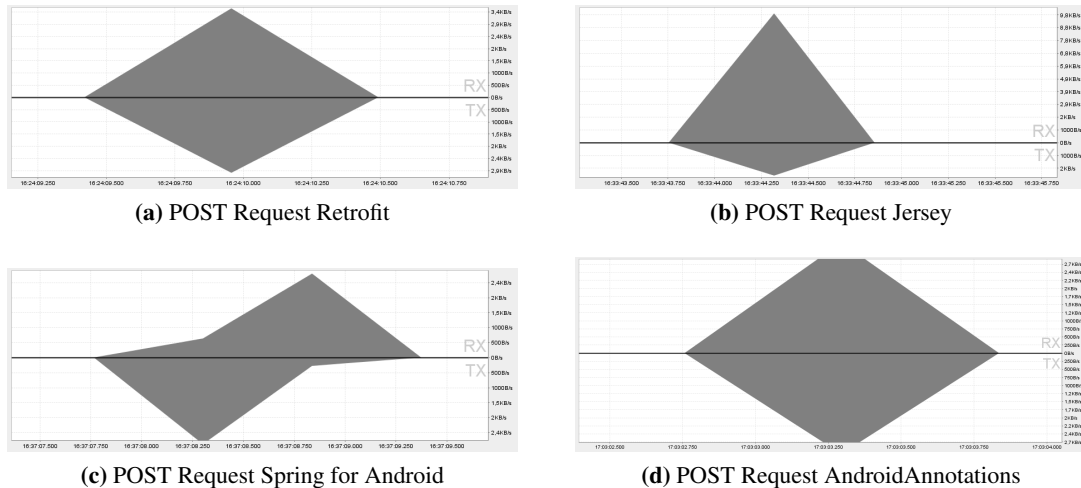


Abbildung 6.2: Zeitmessung der POST Requests

In der Abbildung 6.3 wird die CPU Auslastung der einzelnen Frameworks gegenübergestellt. Aus dieser Abbildung kann entnommen werden, dass AndroidAnnotations die CPU am geringsten in Anspruch nimmt und Jersey am stärksten. Bei der Benutzung der Apps ist auch festzustellen, dass AndroidAnnotations und Retrofit am schnellsten arbeiten und eine flüssige Bedienung der Anwendung möglich ist. Jersey ist des öfteren langsam und hat kleinere Ruckler während der Bedienung. Die Messungen konnten dabei den subjektiven Eindruck während der Bedienung der Apps bestätigen.

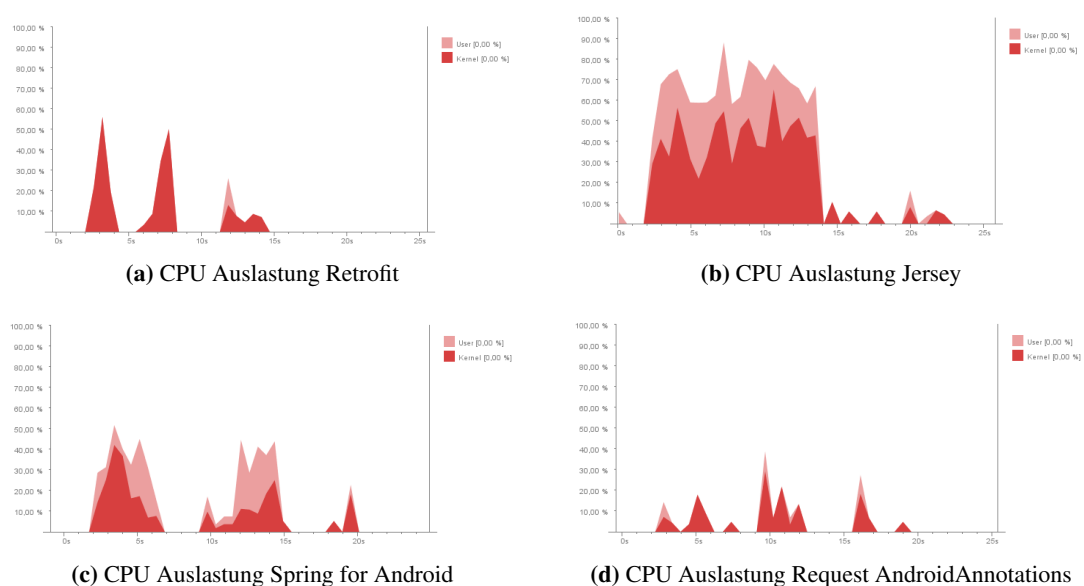


Abbildung 6.3: CPU Auslastung

Speicherplatz

Eine der größten Herausforderungen beim Implementieren von Apps ist der Hardwareunterschied der mobilen Geräte. Die Hardwarekomponenten unterscheiden sich nicht nur in der Displaygröße oder CPU Leistung, sondern auch im Speicherbereich [51]. Sei es nun der zur Verfügung gestellte Random Access Memory (RAM) oder der interne Speicher, welcher bei manchen Geräten durch Speicherkarten erweitert werden kann. Je mehr und je größere Apps installiert werden, desto mehr leidet die Performance – insbesondere auf älteren oder Hardware schwächeren Geräten. Auch spielt die Installationszeit eine wichtige Rolle. User wollen Apps so schnell wie möglich nutzen, längere Installationszeiten sorgen schneller dafür, dass die Installation abgebrochen wird [53]. Bei der Gegenüberstellung der APK Größe und der maximalen RAM Beanspruchung schnitt AndroidAnnotations am besten ab, Jersey am schlechtesten (siehe Tabelle 6.1).

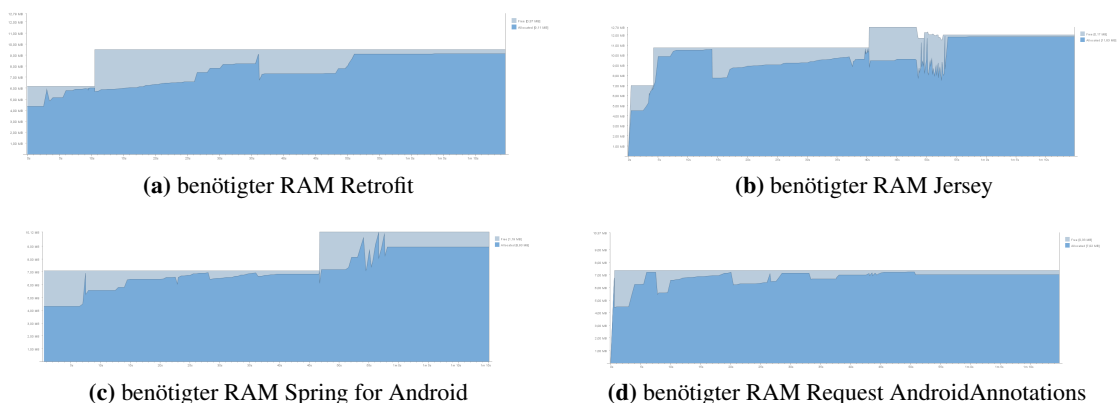


Abbildung 6.4: benötigter RAM

Erweiterte Technische Fähigkeiten der Frameworks

Sicherheit ist ein wesentliches Thema in der Informatik und auch für mobile Geräte. Denn die Apps sind nicht nur auf Informationen des Smartphones beschränkt, sondern haben auch Zugriff auf das Internet. OAuth ist ein Webstandard für den beschränkten Zugriff von Benutzer-Ressourcen auf Servern [54]. Dieser Standard wird von allen Frameworks unterstützt und ermöglicht einen sicheren und beschränkten Zugriff auf Ressourcen.

Alle evaluierten Frameworks unterstützten keine weiteren Übertragungsprotokolle außer HTTP. Die ACID-Eigenschaften werden auch von keinem der Frameworks implementiert. Soll aber ein Transaktionsmanagement clientseitig unterstützt werden, muss eine eigene Implementierung geschrieben werden. Dabei muss besonders darauf geachtet werden, dass alle Daten, die während einer Transaktion verwendet werden, gesperrt sind und sich nicht ändern dürfen, so lange bis die Transaktion Committed wird oder ein Rollback durchgeführt wird [8].

Jersey hat eine serverseitige Referenzimplementierung. Für Spring for Android existiert ebenfalls eine Referenzimplementierung aus der Spring Familie. AndroidAnnotations hat keine direkt Referenzimplementierung, aber man kann auch auf jene der Spring Familie zurückgreifen - denn die REST Implementierung von AndroidAnnotations basiert auf Spring for Android. Retrofit hat keine serverseitige Referenzimplementierung.

AndroidAnnotations stellt als einziges Framework zusätzliche Dienste bereit, welche die Implementierung einer Android App erleichtern. Diese zusätzlichen Dienste sind unter anderem Dependency Injection oder Event Binding, siehe Kapitel 5.4.

Tabelle 6.1: Gegenüberstellung der Frameworks

| | Retrofit | Jersey | Spring for Android | AndroidAnnotations |
|---------------------------------|--|--|--|--|
| Version | 2.0 | 2.0 | 2.22.1 | 4.0 |
| Entwicklungskultur | | | | |
| Lizenz | Apache License Version 2.0 | CDDL Version 1.1 GPL Version 2 | Apache License Version 2.0 | Apache License 2.0 CDDL |
| aktive Community | Ja | Ja | Ja | Ja |
| Dokumentation | grundlegend | gut | gut | gut |
| Codebeispiele | Ja | nicht speziell für Android | Ja | Ja |
| Implementierung | | | | |
| Einbindung | leicht | problematisch | leicht | leicht |
| HTTP-Methoden | GET, POST, PUT, DELETE, HEAD | GET, POST, PUT, DELETE, HEAD, OPTIONS | GET, POST, PUT, DELETE, HEAD, OPTIONS | GET, POST, PUT, DELETE, HEAD, OPTIONS |
| HTTP-Header | erweiterbar | erweiterbar | erweiterbar | erweiterbar |
| unterstützte Medientypen | alle möglichen, wenn Konverter für Typ vorhanden ist | alle möglichen, wenn Konverter für Typ vorhanden ist | alle möglichen, wenn Konverter für Typ vorhanden ist | alle möglichen, wenn Konverter für Typ vorhanden ist |
| URL veränderbar | Ja | Ja | Ja | Ja |
| asynchrone Requests | Ja | Ja | Ja | Ja |
| HATEOAS Konzept | Nein | Ja | mithilfe von Spring HATEOAS | ohne Annotations und mithilfe von Spring HATEOAS |
| Error-Handling | unterstützt | unterstützt | unterstützt | unterstützt |

| | Retrofit | Jersey | Spring for Android | AndroidAnnotations |
|--|------------------------|---------------|--------------------|-----------------------------------|
| Performance und Speicherplatz | | | | |
| GET Request | ~3s | ~12s | ~5s | ~3.5s |
| POST Request | ~1s | ~1s | ~1.5s | ~1s |
| CPU Auslastung | max. ~55% | max. ~87% | max. ~52% | max. ~40% |
| belegter RAM | max. 9.11 MB | max. 11.83 MB | max. 8.93 MB | max 7.02 MB |
| APK Größe | 1.70 MB | 2.58 MB | 1.69 MB | 1.44 MB |
| Erweiterte Technische Fähigkeiten | | | | |
| Sicherheit | OAuth2, SSL via OkHttp | OAuth2, SSL | OAuth2, SSL | OAuth2, SSL |
| andere Protokolle außer HTTP | Nein | Nein | Nein | Nein |
| ACID-Eigenschaften | Nein | Nein | Nein | Nein |
| Server Implementierung | Nein | Ja | Spring Framework | Nein (eventuell Spring Framework) |
| zusätzliche Dienste | Nein | Nein | Nein | Ja |

6.2 Evaluierung im Kontext Revex2020

Als REST Frameworks sind sowohl Retrofit, Spring for Android und AndroidAnnotations geeignet. Von Jersey als REST Framework ist abzuraten, da dieses Framework nur mithilfe eines Workarounds benutzt werden kann. Darüber hinaus schneidet Jersey im Performance- und Speichervergleich am schlechtesten ab.

Vergleicht man nun jene anderen drei ist der Unterschied minimal. Retrofit weist eine etwas schwächere Dokumentation auf, da gerade eine neue Version erschienen ist und die Dokumentation teilweise noch nicht an die neueste Version angepasst wurde (Stand Ende 2015). Spring for Android ist im Lesezugriff langsamer, was einen minimalen Nachteil darstellt - da das Anzeigen einer großen Datenmenge länger dauert. Retrofit und AndroidAnnotations haben einen leicht zu lesenden Code, da dieser Annotations basiert ist. Dadurch verringert sich der zu schreibende und wartende Code, wodurch die Entwicklung beschleunigt und die Wartbarkeit verbessert werden kann [55].

Zusammenfassend ist zu empfehlen AndroidAnnotations als REST Framework zu verwenden. Dieses Framework arbeitet Ressourcenschonend und unterstützt alle nötigen Anforderungen, welche für das Revex Projekt benötigt werden. Der HTTP-Header kann erweitert und verwendet werden und es werden alle benötigten HTTP-Methoden (GET, PUT, POST und DELETE) unterstützt. Des Weiteren sind sowohl asynchrone Requests und das behandeln von Fehlern möglich. Ein weiterer Pluspunkt von AndroidAnnotations sind die zusätzlichen Dienste, welche die Implementierung einer Android App erleichtern. Sollen diese Dienste, welche in AndroidAnnotations enthalten sind, mit den anderen Frameworks verwendet werden, müssen zusätzliche Drittanbieter Bibliotheken hinzugefügt werden. Dadurch wird, aber der benötigte Speicherplatz der App erhöht.

6.3 Persönliches Fazit zu den Frameworks

Es folgt nun ein kurzer persönlicher Eindruck zu den Frameworks. Wo sind Probleme während der Implementierung aufgetreten und welche Aspekte besonders positiv aufgefallen sind.

Retrofit

Retrofit ist ein Annotations basiertes Framework, wodurch der Code leicht verständlich und kurz zu schreiben ist. Das Einbinden in ein Projekt gestaltet sich als sehr einfach, es müssen nur die nötigen Abhängigkeiten in die gradle-Datei des Projektes eingefügt werden. Problematisch bei der Implementierung war die teilweise veraltete Dokumentation. Es konnte anfangs nicht festgestellt werden, auf welche Version des Frameworks die gefundene Beschreibung Bezug nahm. Bei mehrmaligem Zugriff auf die offizielle Dokumentation konnte aber festgestellt werden, dass diese laufend erweitert und auf die neueste Version abgeändert wurde. Dies weist auf eine aktive Community hin. Performance technisch konnten keinerlei Probleme festgestellt werden, das Framework arbeitet flüssig und es kam zu keinen Verzögerungen während der Nutzung der App.

Jersey

Jersey war am problematischsten zu implementieren, da es am längsten gedauert hat dieses Framework unter Android verwenden zu können. Es musste eine umfangreiche Recherche im Internet durchgeführt werden, bis die Probleme dieses Frameworks mit Android festgestellt werden konnten. Danach musste noch einige Zeit investiert werden, bis ein funktionierender Workaround für die Probleme gefunden werden konnte. Die clientseitige Dokumentation des Framework ist gut und liefert alle relevanten Informationen für den Implementierungsprozess. Während der Nutzung der App kam es immer wieder zu Verzögerungen, bis alle relevanten Daten geladen wurden. Dies hat wesentlich Auswirkungen auf die Usability, da eine längere Wartezeit nötig ist, bis dem User alle Daten angezeigt werden. Jersey schnitt im Performance- und Speichervergleich am schlechtesten ab. Es ist daher deutlich festzustellen, dass die clientseitige Implementierung von Jersey nicht für Android optimiert wurde.

Spring for Android

Das Einbinden von Spring for Android in ein Projekt gestaltet sich als trivial, es müssen nur die nötigen Abhängigkeiten in die gradle-Datei des Projektes eingefügt werden. Es sind sowohl eine gute Dokumentation, als auch Beispielcodes auf der offiziellen Homepage verfügbar. Der Code ist gut strukturiert, aber nicht so leicht verständlich, wie bei Annotations basiertes REST Frameworks. Daher musste mehr Zeit in das Implementieren der REST Schnittstellen investiert werden, da auch mehr Code zu schreiben war. Performancetechnisch lag dieses Framework im Mittelfeld, größtenteils arbeitet es sehr flüssig. Nur beim Abrufen großer Datenmengen kommt es zeitweise zu einer minimalen Verzögerung.

AndroidAnnotations

Das Einbinden und Verwenden von AndroidAnnotations ist sehr einfach, es sind wiederum nur alle nötigen Abhängigkeiten in die gradle-Datei des Projektes einzufügen. Die Dokumentation und gefunden Code Beispiele sind ebenfalls sehr gut und enthalten alle benötigten Informationen für die Implementierung. Da AndroidAnnotations ein Annotations basiertes Framework ist, ist der Code gut lesbar und schnell zu schreiben, was den Implementierungsprozess erheblich verkürzt. Performance technisch schnitt dieses Framework am besten ab, es arbeitet flüssig und es kam zu keinerlei Verzögerungen während der Nutzung. Ein großer Mehrwert dieses Framework ist, dass zur Verfügung stellen zusätzlicher Dienste, fernab der REST Kommunikation. Dadurch gestaltet sich das Implementieren einer Android App als einfacher. Da auch Annotation für andere Bereiche existieren, wird der zu schreiben Code für eine App stark reduziert. Zum Beispiel existiert Annotationen für das behandeln von Events bei Views, es werden keine anonymen Listener Klassen mehr benötigt.

7 Zusammenfassung

Diese Arbeit hat sich mit der Evaluierung von REST Frameworks für Android im Kontext des Revex2020 Projektes auseinandergesetzt. Die ausgewählten Frameworks wurden anhand einer Beispielimplementierung vorgestellt und verglichen.

Zu Beginn der Arbeit wurde auf einen der größten Trends am Business-Markt eingegangen. Dieser ist die Mobilisierung der Geschäftswelt, wobei unabhängig von Stakeholdern, Zeit, Ort und Geräten auf Daten und Anwendungen zugegriffen werden soll. Ein wesentlicher Innovationsstrang ist dabei die Entwicklung von Business-Apps. Mobile Endgeräte werden in bestehende Geschäftsprozesse der Unternehmen integriert und greifen dabei oft auf definierte Schnittstellen oder angebotene Dienste zu [1]. Herkömmliche Software rückt dabei immer weiter in den Hintergrund, Daten sind sofort und überall abrufbar. Mobile Endgeräte wie Smartphones und Tablets verändern daher die Geschäftswelt nachhaltig, da Unternehmensinformationen jederzeit verfügbar sind - die Unternehmen der Zukunft sind mobil [4].

In einem Teil der Arbeit werden die verwendeten Technologien Android und REST beschrieben. Android ist ein Betriebssystem, welches primär für Smartphones und Tablets konzipiert ist [11]. Durch die quelloffene Struktur des Betriebssystems ist Android bei vielen Konsumenten und Entwicklern sehr beliebt, wodurch viele Unternehmen ihre mobilen Applikationen auf dieses Betriebssystem ausrichten [12]. Aufgrund dessen wurde die Evaluierung der REST Frameworks auch auf Android ausgerichtet. In einer kurzen Zusammenfassung wurde auf REST als Architekturstil eingegangen und die 5 Kernprinzipien dieses Stils beschrieben [21].

Mithilfe der prototypisch entwickelten Android-App soll es Mitarbeitern zukünftig möglich sein, den Zustand einzelner Kraftwerkskomponenten vor Ort zu erfassen oder Details zum Kraftwerk anzuzeigen. Die App soll auf das bereits vorhandene Backend, eines REST-Webservices zugreifen und diesen verwenden. Um auf das vorhandene Backend zuzugreifen muss ein REST Framework verwendet werden, dass für Android kompatibel ist und performant arbeitet. Um geeignete REST Frameworks zu finden, wurde eine Technologierecherche durchgeführt und die drei populärsten und den Anforderungen adäquatesten Frameworks für eine Beispielimplementierung ausgewählt.

Die Qualität der einzelnen Frameworks wurde anhand definierter Kriterien evaluiert. Diese Kriterien umfassen sowohl die Entwicklungskultur, den Implementierungsprozess und einen Performance- und Speichervergleich. Die ausgewählten REST Frameworks wurden anhand einer Beispielimplementierung vorgestellt und deren Verwendungsweise beschrieben. Im Anschluss daran wurden die einzelnen Frameworks mittels der definierten Kriterien verglichen. Durch die Evaluierung der Frameworks konnte festgestellt werden, dass AndroidAnnotations am besten für die Implementierung der Revex2020 App geeignet ist. Als Abschluss wurde noch ein kurzes persönliches Resümee zu den Frameworks gezogen.

Literatur

Wissenschaftliche Literatur

- [1] Claudia Linnhoff-Popien und Stephan Verclas. *Smart Mobile Apps – Mit Business-Apps ins Zeitalter mobiler Geschäftsprozesse*. Springer, 2012. Kap. 1: Mit Business-Apps ins Zeitalter mobiler Geschäftsprozesse, S. 3–14.
- [2] Jens Bertram und Carsten Kleiner. *Smart Mobile Apps – Mit Business-Apps ins Zeitalter mobiler Geschäftsprozesse*. Springer, 2012. Kap. 17: Mobile Apps in Enterprise-Anwendungen unter Berücksichtigung von Sicherheitsaspekten, S. 253–267.
- [3] Markus Fischer, Kálmán Képes und Alexander Wassiljew. *Vergleich von Frameworks zur Implementierung von REST-basierten Anwendungen*. 2013. URL: <http://elib.uni-stuttgart.de/opus/volltexte/2013/8731>.
- [4] Michael Kern. *Smart Mobile Apps – Mit Business-Apps ins Zeitalter mobiler Geschäftsprozesse*. Springer, 2012. Kap. 7: Eine neue Generation von Geschäftsanwendungen, S. 95–106.
- [5] Eduard Doujak. “Ein Beitrag zur technisch ökonomischen Analyse von Verschlußkörpern bei der Beurteilung über Revitalisierungs- bzw. Modernisierungsmaßnahmen von Wasserkraftanlagen”. TU Wien, 2000.
- [8] Martin Mensing-Braun. “Transaktionen im Kontext von REST-basierten Web-Diensten”. FH Hagenberg, 2015.
- [9] Lars Grammel Margaret-Anne Storey Chris Parnin Christoph Treude. “Crowd documentation: Exploring the coverage and the dynamics of API discussions on Stack Overflow”. In: *Georgia Institute of Technology, Tech. Rep* (2012).
- [11] Rajinder Singh. “An Overview of Android Operating System and Its Security Features”. In: *International Journal of Engineering Research and Applications* (2014).
- [15] Ed Burnette. *Hello, Android 4th Edition - Introducing Google’s Mobile Development Platform*. 2015. Kap. Appendix 1: Java vs. the Android Language and APIs.
- [16] Dr. S. K. Shah Priyank.S.Patil. “Implementation of Android Application to Search nearby Places with GPS Navigation Technology”. In: *International Journal of Advanced Research in Electrical, Electronics and Instrumentation Engineering* (2015).
- [18] Marko Gargenta. *Einführung in die Android-Entwicklung*. O’Reilly Germany, 2011. Kap. Kapitel 1 - Android im Überblick, S. 1–6.
- [20] Roy Thomas Fielding. “Architectural Styles and the Design of Network-based Software Architectures”. Diss. 2000.
- [21] Silvia Schreier Oliver Wolf Stefan Tilkov Martin Eigenbrodt. *REST und HTTP: Entwicklung und Integration nach dem Architekturstil des Web*. dpunkt.verlag, 2015. Kap. 2: Einführung in REST.
- [25] K. Maeda. “Performance evaluation of object serialization libraries in XML, JSON and binary formats”. In: *Digital Information and Communication Technology and it’s Applications (DICTAP), 2012 Second International Conference on*. 2012, S. 177–182.

- [28] Anthony Dahanne. *Instant Spring for Android Starter*. Packt Publishing, 2013.
- [30] SpringSource. *Spring Framework Reference Documentation*. 2015. Kap. 27.10. Accessing RESTful services on the Client, S. 661–667.
- [32] Roland Kübert, Gregory Katsaros und Tinghe Wang. “A RESTful Implementation of the WS-agreement Specification”. In: *Proceedings of the Second International Workshop on RESTful Design*. WS-REST ’11. 2011, S. 67–72.
- [42] Ursula Widmer und Konrad Bähler. *Open Source Jahrbuch 2006: Zwischen Softwareentwicklung und Gesellschaftsmodell*. Lehmanns Media, 2006. Kap. Open-Source-Lizenzen - Wesentliche Punkte für Nutzer, Entwickler und Vertreiber, S. 165–179.
- [43] Alexandra Kees. *Open Source Enterprise Software*. 2015. Kap. 3.5 Communities, S. 30–32.
- [44] M.P. Robillard. “What Makes APIs Hard to Learn? Answers from Developers”. In: *IEEE Software* 26.6 (2009), S. 27–34.
- [45] J. Singer T.C. Lethbridge und A. Forward. “How software engineers use documentation: the state of the practice”. In: *IEEE Software* 20.6 (2003), S. 35–39.
- [46] Dirk Louis und Peter Müller. *Android: Der schnelle und einfache Einstieg in die Programmierung und Entwicklungsumgebung*. Hanser eLibrary: Carl Hanser Verlag GmbH & Company KG, 2014.
- [47] Jun Gong und Peter Tarasewich. “Guidelines for handheld mobile device interface design”. In: *Proceedings of DSI 2004 Annual Meeting*. 2004, S. 3751–3756.
- [48] Claas Wilke. “Energieverbrauchsermittlung von Android-Applikationen”. In: *Innovationsforum open4INNOVATION2012 regional kooperativ-global innovativ*. Technische Universität Dresden, 2012.
- [49] Marcus Vetter. “Ereigniskorrelation auf energiebeschränkten mobilen Endgeräten”. Universität Stuttgart, 2012.
- [50] Radhika Mittal, Aman Kansal und Ranveer Chandra. “Empowering Developers to Estimate App Energy Consumption”. In: *Proceedings of the 18th Annual International Conference on Mobile Computing and Networking*. Mobicom ’12. ACM, 2012.
- [51] A. Mesbah M.E. Joorabchi und P. Kruchten. “Real Challenges in Mobile App Development”. In: *Empirical Software Engineering and Measurement, 2013 ACM / IEEE International Symposium on*. 2013, S. 15–24.
- [52] Marco Glier Dr. Herbert A. Meyer Petra Vogt. “Performance–(k) ein Thema für Usability Professionals?” In: *Usability Professionals* (2005).
- [54] Mohamed Shehab und Fadi Mohsen. “Securing OAuth Implementations in Smart Phones”. In: *Proceedings of the 4th ACM Conference on Data and Application Security and Privacy*. CODASPY ’14. 2014, S. 167–170.
- [55] Robert C. Martin. *Clean Code: A Handbook of Agile Software Craftsmanship*. 2009. Kap. Chapter 11: Systems, S. 164–166.

Online Referenzen

- [6] *Library vs. Framework?* Letzter Besuch: 03.03.2016. URL: <http://goo.gl/VPpnwi>.
- [7] Martina Szybiak. *Kurz & Gut: Ressourcen*. Letzter Besuch: 31.10.2015. URL: <http://goo.gl/79eBFm>.

- [10] Ben Balter. *How to identify a strong open source project*. Letzter Besuch: 28.09.2015. URL: <http://ben.balter.com/2014/06/02/how-to-identify-a-strong-open-source-project/>.
- [12] IDC. *Market share worldwide smartphone shipments by operating system from 2014 to 2019*. Letzter Besuch: 04.01.2016. URL: <http://goo.gl/pe2QYO>.
- [13] Lars Vogel. *Introduction to Android development with Android Studio - Tutorial*. Letzter Besuch: 04.01.2016. URL: <http://www.vogella.com/tutorials/Android/article.html#android>.
- [14] Hans-Christian Dirscherl. *Unterschiede zwischen JVM und Dalvik VM*. Letzter Besuch: 26.01.2016. URL: <http://goo.gl/Hz7VMf>.
- [17] Ed Burnette. *Java vs. Android APIs*. Letzter Besuch: 26.01.2016. URL: <http://www.zdnet.com/article/java-vs-android-apis/>.
- [19] *Dashboards - Platform Versions*. Letzter Besuch: 21.02.2016. URL: <http://goo.gl/PoiPRV>.
- [22] Square. *Retrofit*. Letzter Besuch: 19.10.2015. URL: <http://square.github.io/retrofit/>.
- [23] Square. *OkHttp*. Letzter Besuch: 31.10.2015. URL: <http://square.github.io/okhttp/>.
- [24] Roger Hu. *Consuming APIs with Retrofit*. Letzter Besuch: 31.10.2015. URL: <https://guides.codepath.com/android/Consuming-APIs-with-Retrofit>.
- [26] ITWissen.info. *Spring*. Letzter Besuch: 02.01.2016. URL: <http://www.itwissen.info/definition/lexikon/Spring-Spring-framework.html>.
- [27] *Spring for Android*. Letzter Besuch: 02.01.2016. URL: <https://github.com/spring-projects/spring-android>.
- [29] Martin Fowler. *POJO: Plain Old Java Object*. Letzter Besuch: 01.03.2016. URL: <http://www.martinfowler.com/bliki/POJO.html>.
- [31] *JSR 311: JAX-RS: The Java™ API for RESTful Web Services*. Letzter Besuch: 03.01.2016. URL: <https://jcp.org/en/jsr/detail?id=311>.
- [33] Doku Jersey API. *Chapter 5. Client API*. Letzter Besuch: 28.01.2016. URL: <https://jersey.java.net/documentation/latest/client.html>.
- [34] JAX-RS 2.0 API Specification. *Interface Invocation.Builder*. Letzter Besuch: 30.01.2016. URL: <https://jax-rs-spec.java.net/nonav/2.0/apidocs/javax/ws/rs/client/Invocation.Builder.html>.
- [35] Jakub Podlesak. *Jersey 2.x Client on Android*. Letzter Besuch: 30.01.2016. URL: https://blogs.oracle.com/japod/entry/jersey_2_x_client_on.
- [36] Jakub Podlesak. *Jersey 2.x Client on Android - Take 2*. Letzter Besuch: 30.01.2016. URL: https://blogs.oracle.com/japod/entry/jersey_2_x_client_on1.
- [37] *Sponsors*. Letzter Besuch: 09.02.2016. URL: <https://github.com/excilys/androidannotations/wiki/Sponsors>.
- [38] *Introduction*. Letzter Besuch: 09.02.2016. URL: <https://github.com/excilys/androidannotations/wiki>.
- [39] Roy Clarkson. *Clean Code in Android Applications*. Letzter Besuch: 09.02.2016. URL: <http://spring.io/blog/2011/08/26/clean-code-in-android-applications>.
- [40] *Rest API*. Letzter Besuch: 09.02.2016. URL: <https://github.com/excilys/androidannotations/wiki/Rest-API>.
- [41] *Android-apt Overview*. Letzter Besuch: 09.02.2016. URL: <https://bitbucket.org/hvisser/android-apt>.
- [53] Stefanie Schäfers. *Google verdoppelt Größe für Android-APKs*. Letzter Besuch: 19.02.2016. URL: <https://goo.gl/KM6ncg>.